Homework1

黃孟霖 103062315

1、DCT image compression

A、這題要做的就是先用 DCT 進行轉換，套用 mask 後再用 IDCT 轉回來，DCT 轉換我這題用的是 DCT transform matrix 的概念，參考 matlab documentation
https://www.mathworks.com/help/images/discrete-cosine-transform.html，
D=T*A*T'其中 T'是 T 的轉置，T、A、T'是 8*8 matrix，T 的公式如下

$$T_{pq} = \begin{cases} \frac{1}{\sqrt{M}} & p = 0, \quad 0 \le q \le M-1 \\ \sqrt{\frac{2}{M}} \cos\frac{\pi(2q+1)p}{2M} & 1 \le p \le M-1, \quad 0 \le q \le M-1 \end{cases}$$

Matlab code(my_dctmtx)

```
function mtx=dctmtx(input)
    mtx=zeros(input,input);
    for i=1:input
        for j=1:input
          if i==1
                T(i,j)=1/sqrt(8);
            else
                T(i,j)=sqrt(2/8)*cos(pi*(2*j-1)*(i-1)/16);
          end
        end
    end
    mtx=T;
end
```

之後要做的是將 img 切成 8*8 的 block 去做 D=T*A*T'，這邊用的是 matlab 的內建函數 blockproc，這部分也是參考上述網頁的 documentation。

```
mask=zeros(8,8);
mask(1:k,1:k)=1;
dct=@(block_struct)T*block_struct.data*T';
B=blockproc(imag,[8 8],dct);
B2=blockproc(B,[8 8],@(block_struct)mask.*block_struct.data);
invdct=@(block_struct)T'*block_struct.data*T;
output=blockproc(B2,[8 8],invdct);
```

一開始先將變數轉換成函數也就是 dct=@(block_struct).....這行，因為 blockproc 中第三個變數需要為 function，然後 B 就是 DCT 的結果，之後套用 mask 到 B

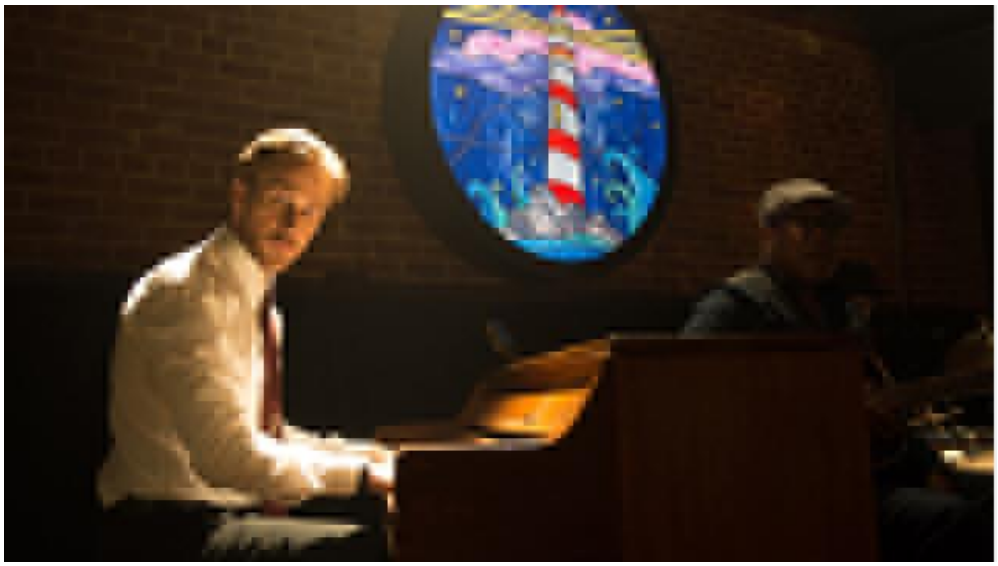
上，在做 IDCT，就是將 D=T*A*T'=>T'*D*T=A，因為 T'是正交矩陣，所以 T 的 inverse 等於 T 的轉置。

最後再去計算 PSNR，分成 RGB 去算，取平均帶進公式，code 如下

```
MSER=1/(m*n)*sum(sum((input1_s(:,:,1)-input2_s(:,:,1)).^2));
MSEG=1/(m*n)*sum(sum((input1_s(:,:,2)-input2_s(:,:,2)).^2));
MSEB=1/(m*n)*sum(sum((input1_s(:,:,3)-input2_s(:,:,3)).^2));
MSE=(MSER+MSEG+MSEB)/3;
%代公式，公式中的f_peak在此處因為是single 所以f_peak=1
 psnr=10*log10(1/MSE);
```

ResultImage:

N=2　IDCT



PSNR:

```
 26.8887
```

N=4 IDCT

PSNR:

33.3722

N=8

PSNR: Inf

從上面的結果我們可以發現，當 N 越大時也 PSNR 越大，也就代表著影像越清楚，這是因為當 N=2 時的 mask 所留下的 pixel 跟 N=8 相比是較少的，所以可以看到 n=2 時的鋸齒狀明顯的多，也較不清楚，並且當 n=8 時因為就相當於在這 8*8 的區塊中每個 pixel 都保留，所以 psnr 會為 inf，這邊要注意的是在做 PSNR 前要先將圖片從 double 轉成 uint8，雖然 T'*T 應該要是單位矩陣，但因為 MATLAB 浮點數精度問題，所以不會成單位矩陣，而會有些非 0 但很接近 0 的值在非斜對角上，所以透過轉成 uint8 的方式，把這些運算時的誤差給處理

掉，讓那些非 0 但接近 0 的點變成 0，之後做出來的 psnr 就會為 inf 了。

B、這題的話就先做 YIQ 再做上一題的步驟，之後再用 IYIQ 轉回 RGB。

YIQ 套用課本上的公式，IYIQ 則做 inverse 即可

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

```
for i=1:m
    for j=1:n
        s(1,1)=input(i,j,1);
        s(2,1)=input(i,j,2);
        s(3,1)=input(i,j,3);
        output(i,j,:)=Y*s;
    end
end
```

Code 如右:

ResultImage:

YIQ

N=2

PSNR: 26.8887

N=4

PSNR: 33.3722

N=8

PSNR: Inf

B、比較 A 跟 B 可以發現基本上是一模一樣的，這是因為假設 YIQ 矩陣為 Y，DCT=T*A*T'，而我們 B 所做的就是 inv(Y)*T'*T*Y*A*T'*T，因為 T 是正交矩陣，所以 T 的 inverse 等於 T 的轉置，所以作化簡後 B 與 A 是一樣的。

2、Image filter

A、首先利用 matlab 內建函數 fspecial 依照輸入的 size 跟 sigma 造出 gaussian filter

```
G=fspecial('gaussian',[s s],sigma);
```

之後利用兩層 for 迴圈去做 convolution，並且用兩次 sum 算出 convolution 的和，這邊我對於邊界的處理辦法是不處理，也就是當遇到邊界則維持原樣，因為高斯矩陣是對稱的，所以用 correlation 的結果會等於 convolution。

```
for i=1:m
    for j=1:n
       if (i-m2>0&&j-n2>0&&j+n2<=n&&i+m2<=m)
       p=img(i-m2:i+m2,j-n2:j+n2);
       output(i,j)=output(i,j)+sum(sum(G.*p));
       else
       output(i,j)=img(i,j);
       end
    end
end
```

ResultImage:

Sizes 3*3 sigma =0.3

Size 9*9 sigma=1

從上面兩圖比較下可以發現，mask 的 size 越大時，會讓影像變得更模糊，也會降低 pepper and salt 的 noise，而 sigma 值也占了很重要的一部份，當 sigma 值越大，會 blur 更大的範圍，也讓下圖比上圖更加模糊。然後在 size 為 9*9 時可以看到有邊框的感覺，這是因為對於邊界採不處理，所以還有些微的 pepper and salt noise，也就形成邊框的感覺。

B、這題是對圖片進行 median filter，median filter 是取中位數來當作那個 pixel 的值，因為 median 的 median 不會是 size*size 的中位數，所以先 reshape 成[size*size,1]的一維矩陣再來做 median，這題對於邊界我同樣採不處理。

```
if i-m2>0&&j-n2>0&&i+m2<=m&&j+n2<=n
    p=img(i-m2:i+m2,j-n2:j+n2);
    p=reshape(p,[s*s,1]);
    output(i,j)=median(p);
else
    output(i,j)=img(i,j);
    index=index+1;
    disp(index);
end
```

Code 如右

ResultImage:

3*3

9*9

從 A 跟 B 我們可以發現 median filter 的模糊效果是比 gaussian filter 好的，這是因為 median filter 只取中位數一個點，然而 gaussian filter 是取周圍的點去 convolution，所以模糊效果 median filter 是比 gaussian filter 強一點的。

3、Interpolation

A、 nearest neighbor 就是取最近的點，透過 round 這個函數來四捨五入取 x 跟 y 方向最接近的值即可，這部分我有加上一個值，這樣的做法好處在於可以免去一堆 if else 的邊界判斷。

```
a=round(i/s+0.499999999);
b=round(j/s+0.499999999);
```

ResultImage:

PSNR: 27.7087

B、bilinear 則是取四個點來做線性內插，透過 floor 跟 ceil(無條件捨去)來取

點，分別為(a,b),(a+1,b),(a,b+1),(a+1,b+1)，用三層 for 迴圈去跑，第三層為色彩維度，對於 a、b 小於 0 的話則讓他對回 1，c、d 超過 250 對回 250。會減去一個 0.5 的意思在於修正位移，因為我們今天將 0 的點對回 1 等於位移了一位，所以我們期望將 4n-1 對於所有正整數 n 而言的點對到 n n+1，經過簡單解聯立得知放 0.5 是最合適的。

ResultImage:

```
x=(i-0.5)/scale+0.5;
for j=1:n2
     y=(j-0.5)/scale+0.5;
     for k=1:h
          a=floor(x);
          b=floor(y);
          c=ceil(x);
          d=ceil(y);
          if a==0
               a=1;
          end
          if c>=m
               c=m;
          end
          if d>=n
               d=n;
          end
          if b==0
               b=1;
          end
```

PSNR: 29.2059

C、

從 A 跟 B 可以發現，B 在視覺上的結果較 A 好，而這樣的結果也反映在 psnr 上，這是因為 bilinear 是線性內插的結果，並且考慮到周遭的點，所以在這種連續圖的呈現會較 NN 來的好，所以若今天取更多點(bicubic)那麼在連續圖上，內插的結果會更好。