DCT image compression implementation:

Reference information:

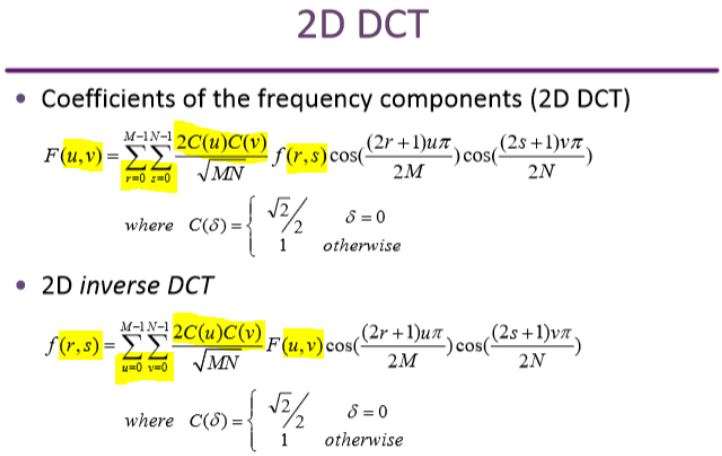
原影象與被處理影象之間的均方誤差相對於$(2^n-1)^2$的對數值(訊號最大值的平方，n是每個取樣值的位元數，例如灰度影象就是8位元，所以MAX值是255)，它的單位是dB。

$$PSNR = 10 \times log_{10}\left(\frac{(2^n - 1)^2}{MSE}\right)$$

MSE是均方誤差

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}||I(i,j) - K(i,j)||^2$$

And I also looked up to the formula in slides to write the function of DCT compression.



When I implement the DCT function, I encountered many problems.
First, the indexes of DCT formula in slide are a little bit confusing,so I took some time to understand it.
Then, following the same steps, I just wrote it into two nested for loops to implement DCT and inverse DCT function.

The DCT function is the sum from index r=0 s=0 to M-1 N-1 while iDCT is from u=0 v=0 to the end, and that is a trap I fell into.

In the beginning , I thought that DCT and inverse DCT can be implemented by the same function, only need to change data input. For example, DCT(u,v,original_img) and the inverse DCT can be written like this DCT(u,v,porcessed_img),but apparently I was wrong.
Since the indexes are not exactly the same between these two implementation.

```
for i=0: block-1
    for j=0: block-1
        % F(u,v) = f(r,s) ...with process...
        DCT_do = DCT_do + img(i+1,j+1)*2*C(u)*C(v)/sqrt(
    end
end
```

```
for i=0:block-1
    for j=0:block-1
       iDCT_do = iDCT_do + img(i+1,j+1)*2*C(i)*C(j)/s
    end
end
```

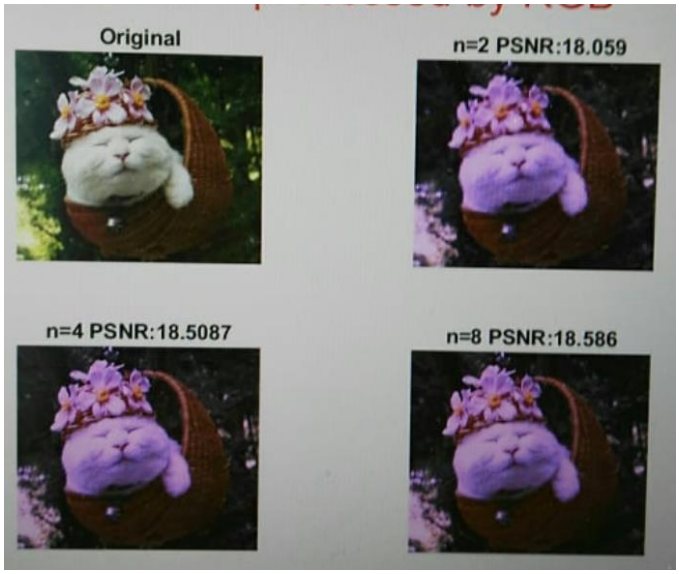
As you can notice, DCT image is prodected by 2*C(u)*C(v), while iDCT is 2*C(i)*C(j),which can't be replaced by just written like DCT(u,v,porcessed_img) , else you may get wrong result.
Another main problem was that when I wanted to cut picture into many blocks, blockproc was the first method I thought. But afraid of invalidation , I finally do it on my own ,

```
image_cut(:,:,n) = img(block*(i-1)+1:block*i,block*(j-1)+1:block*j);
```

It took me a while to develop the formula.
Moreover, the last problem I faced was this:

都是在 RGB 做 PSNR 的計算。
RGB 轉 YIQ -> 做DCT -> inverse DCT -> YIQ轉RGB -> 計算 PSNR (RGB space)

I forgot to change back to RGB space, so I fixed my code but turned out to be wrong result.



My turned back RGB images has color inaccuracy , it took a while to finally discovered that it was another data type problem!
It's can't use "int" type because the data may disappear after transformation, ("double" can save original information)

Wrong data type ,missing data!!!

```
YIQ=uint8(zeros(size(img)));
for i=1:size(img,1)
    for j=1:size(img,2)
        YIQ(i,j,1)=0.2989*img(i,j,1)+0.5870*img(i,j,2)+0.1140*img(i,j,3);
        YIQ(i,j,2)=0.596*img(i,j,1)-0.274*img(i,j,2)-0.322*img(i,j,3);
        YIQ(i,j,3)=0.211*img(i,j,1)-0.523*img(i,j,2)+0.312*img(i,j,3);
    end
end
```

Correct one !!

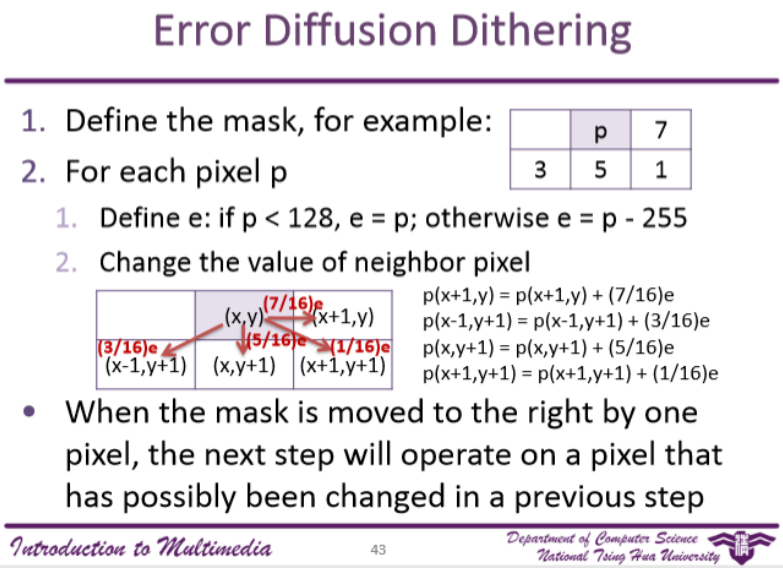
```
YIQ = im2double(zeros(size(img)));

for i=1:size(img,1)
    for j=1:size(img,2)
        YIQ(i,j,1)=0.2989*im2double(img(i,j,1))+0.5870*im2double(img(i,j,2))+0.1140*im2double(img(i,j,3));
        YIQ(i,j,2)=0.596*im2double(img(i,j,1))-0.274*im2double(img(i,j,2))-0.322*im2double(img(i,j,3));
        YIQ(i,j,3)=0.211*im2double(img(i,j,1))-0.523*im2double(img(i,j,2))+0.312*im2double(img(i,j,3));
    end
end
```

Dithering implementation:

This one is far easy than DCT compression since it has very simple formula.

As random dithering and average dithering, just understand the definitions and write it in code.

The error defussion dithering is a little bit complex,



When I was implementing it, I encountered a problem which was that I wrote just like the formula shown left,

*p(x+1,y)=p(x+1,y)+(7/16)*e*

But that turned out to be wrong result, I was very confused until asked my classmate to realized that devision must be done in *last step* since the float number may affect the afterward arithmetic.

So I should do it like this:

*p(x+1,y)=p(x+1,y)+e*7/16*

Very tricky one.

Image convolusion implementation:

It's also not very difficult, but a small problem need to be careful.

Since we have to add a circle of pixels around the picture to make it same size after processing convolusion. But since I add pixel like this:

```
padding = round(hsize/2-1);
pad_img = padarray(img,[padding padding],0,'both');
```

The round(hsize/2-1) is different from my filter size, so it affected my for loop and output answer.

This if-else condition is very important since if I don't add this line, my PSNR function occur mistake "Dimensions of input image should be the same as output image."

```
if i-1 <= m && j-1 <= n ; con(i-1,j-1,t) = tmp; end
```

Due to my padding, I add some redunctant pixels in my M*N picture sometimes(make it > M*N),which is not permitted and thus PSNR got error.

Other parts was simple, just this critical line:

```
tmp = tmp + G(x+1,y+1)*pad_img(i-padding+x,j-padding+y,t);
```

and many for loops to go through whole image to finish convolution.😊

Results discussion:

I use imshowpair(A,B, “diff”) to show difference between original pictures and processed pictures.
So it may make it easier for you to tell the differnce between them.

--Different N in RGB--

Original

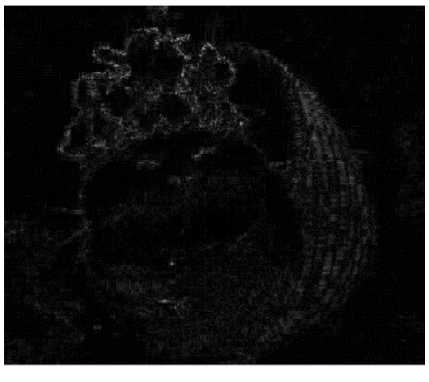
n=2 PSNR:27.2584

n=4 PSNR:35.6371

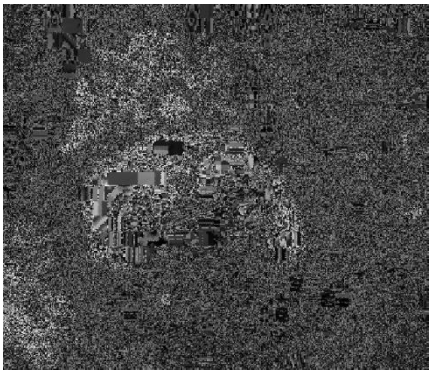
n=8 PSNR:310.6742

n=2

n=4



n=8



Since only keep the lower-frequency(upper-left n-by-n), if n is small , it means we have to abandon many good data, which cause huge critical imformation disappear.
And as you can see the PSNR value is very small when n=2 while PSNR is relatively big when

n=8.

A higher PSNR generally indicates that the reconstruction is of higher quality, but when n=2 or n=4 ,there are not much differences between them , I guess it's because PSNR is a related to *square,* so the changes are not follow linear format and that's why there's huge gap between n=4 and n=8.

--Different N processed by YIQ--

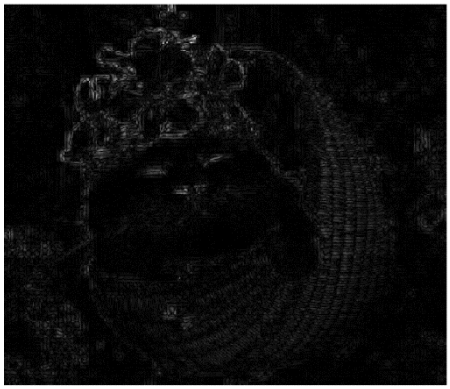
Original YIQ

n=2 PSNR:27.2583

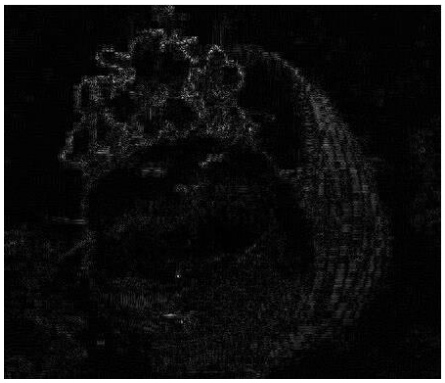
n=4 PSNR:35.6367

n=8 PSNR:75.9446

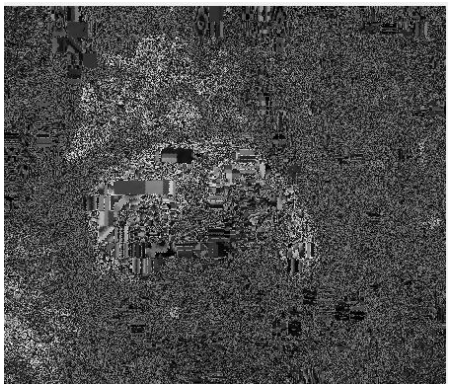
n=2



n=4



n=8



As for first turning to YIQ model and then back to RGB, there's no difference between wether or not being processed by YIQ model for n=2 and n=4.
But huge difference when n=8 and many data disappear during procedure.
Without YIQ being involved, PSNR=310.6742, PSNR=76 with YIQ involved.
I guess it's because when n is large enough ,missing data amount increases sharply and thus the quality of image becomes worse rapidly.
It can't be as high quality as the image which is without YIQ involvement.

--Different Dithering model--

Original

Random Dithering

Average Dithering

Error Diffusion

Random

avg

error

Results are shown above, we can see that random dithering is much different than other two dithering methods since it depends on random number to determine values. Much more messy but not so strong constrast like others.
But since “random” determined value, big problem appears which’s that the processed image is mostly can’t be distinguished since too many bothering pixels.
By formula, we can say that average dithering becomes so white-black-constrast type is due to we determine result values by only one standard reference “ average”. Therfore , not so much “smooth” parts in result image ,only sharp sides and white,black patches.

Error deffusion is much better than average dithering since one pixel affect four pixels beside it. Moreover,with the “error” value producted by certain numbers and is added to the result pixels , it make result image more soft and smooth than average dithering without question.
Furthermore , many details are still remain such as furry like feeling or patterns on cat skin after processing.
The “dots” (pixels to be clear) are very apparent and very disturbing in random dithering but not in other two methods , so I think random dithering isn’t a good choice while dithering.

Image convolution:

--with same sigma--

Original

hsize: 3x3 PSNR:22.5306

hsize: 5x5 PSNR:19.9865

hsize: 7x7 PSNR:18.4856

3x3 5x5 7x7

By image shown above, we can see that in same sigma=1, good quality in smaller size of gaussian filter 3x3 (PSNR=22.5306) while PSNR=18.4856 when filter size is 7x7.
That's because when filter is large, it covers more pixels than smaller ones, so the chance of being processed by filter is higher than small filter and thus more inaccuracy may occur.

--with same hsize(5)--

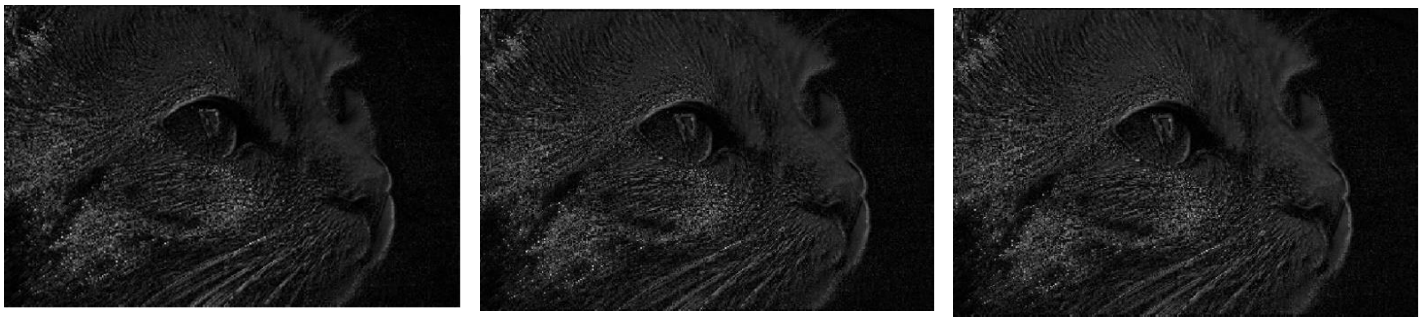
Original

Sigma:1 PSNR:19.9865

Sigma:5 PSNR:20.0625

Sigma:10 PSNR:20.0491

S=1 S=5 S=10



But in same filter size(5x5) and different sigma, there's almost no difference between three images.
PSNR values are slightly different. (range between 0.0626)
I think it's inside error range and can be regarded as the same.
So I concluded that filter size plays much more critical role in image convolution than sigma.