

1. part 1

(a) DCT compression

① n=8 以下為實做的程式碼(部分)：

程式碼	解釋
<pre>original_picture=imread('cat1.png'); picture_double=im2double(original_picture); s=size(original_picture); im_R= picture_double(:,:,1); im_G= picture_double(:,:,2); im_B= picture_double(:,:,3);</pre>	<ol style="list-style-type: none"> 1. 先將 cat1.png 的圖片讀進來。 2. 將影像矩陣轉成 double 型態。 3. s 用來存原圖的 row 和 column 的值。 4. 將原圖分成 3 個 channels 存起來。
<pre>for i=1:s(1) for j=1:s(2) if rem(i,8)==1 && rem(j,8)==1 for a=0:7 for b=0:7 if a==0 c=sqrt(1/8); else c=sqrt(2/8); end A(a+1,b+1)=c*cos(pi*(b+0.5)*a/8); end end end end</pre>	<ol style="list-style-type: none"> 1. 用 2 個 for 迴圈跑完原圖的所有 pixels，因為要把原圖切成許多 8*8 小圖，所以設一個條件： 每遇到 row 和 column 除以 8 的餘數為 1 才需要執行那一塊小圖 dct。 2. 二維 DCT 公式： 二維 DCT 是在一維的基礎下再做一次 DCT 變換 $F(u, v) = c(u)c(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i, j) \cos\left[\frac{(i+0.5)\pi}{N} u\right] \cos\left[\frac{(j+0.5)\pi}{N} v\right]$ $F = AfA^T$ $A(i, j) = c(i) \cos\left[\frac{(j+0.5)\pi}{N} i\right] \quad c(u) = \begin{cases} \sqrt{\frac{1}{N}}, & u = 0 \\ \sqrt{\frac{2}{N}}, & u \neq 0 \end{cases}$
<pre>Y_R(i:i+7,j:j+7)=A*im_R(i:i+7,j:j+7)*A'; Y_G(i:i+7,j:j+7)=A*im_G(i:i+7,j:j+7)*A'; Y_B(i:i+7,j:j+7)=A*im_B(i:i+7,j:j+7)*A'; YI_R(i:i+7,j:j+7)=A'*Y_R(i:i+7,j:j+7)*A; YI_G(i:i+7,j:j+7)=A'*Y_G(i:i+7,j:j+7)*A; YI_B(i:i+7,j:j+7)=A'*Y_B(i:i+7,j:j+7)*A; end end end</pre>	<p>根據上面公式做 DCT 變換得到 3 個 channels 分別是 Y_R、Y_G、Y_B。然後再做 inverse 2D DCT。根據以下式子做轉換。</p> $F = AfA^T$ $\therefore A^{-1} = A^T$ $\therefore f = A^{-1} F(A^T)^{-1} = A^T F A$
<pre>result_N8=cat(3,YI_R,YI_G,YI_B);</pre>	用 cat 函數構造多維矩陣。

計算 PSNR (with 原圖)： 以下為實做的程式碼：(接著上面的 code)

程式碼	解釋
<pre>%PSNR:n=8 mseR=(double(original_picture(:,:,1))-double(result_uint_N8(:,:,1))).^2; mseG=(double(original_picture(:,:,2))-double(result_uint_N8(:,:,2))).^2; mseB=(double(original_picture(:,:,3))-double(result_uint_N8(:,:,3))).^2; mR=sum(sum(mseR))/(s(1)*s(2)); mG=sum(sum(mseG))/(s(1)*s(2)); mB=sum(sum(mseB))/(s(1)*s(2)); mse=(mR+mG+mB)/3; PSNR_N8=10*log10(255^2/mse);</pre>	<ol style="list-style-type: none"> 1. 先算 MSE： 各自算三個 channel 的 MSE 再取平均。 將原圖的 image 矩陣跟做完 DCT 的 image 矩陣相減再平方。 將第 2 點算完的結果矩陣的每個 pixels 加總再除以矩陣 row*column，分別得到三個 channel 的 MSE。 三個 channel 的 MSE 取平均，得到 mse 值。 2. 再拿 mse 帶入 PSNR 公式。
<p>我用 uint8 的圖去直接用 double() 轉成 double， 因為我嘗試過直接用 double 的圖去做，但得到的 PSNR 與代 psnr() 做出來的差異頗大。</p>	

結果： 得到的 PSNR=Inf

我使用內建 function psnr 檢查我自己寫的 psnr 是否正確，得到的結果也跟自己計算的 PSNR 一致。

②n=4

基本上跟 n=8 的程式碼差不多，只是差在，row>4 || column>4 的部分要歸 0。

```
Y_R(i:i+3,j+4:j+7)=0;
Y_G(i:i+3,j+4:j+7)=0;
Y_B(i:i+3,j+4:j+7)=0;
Y_R(i+4:i+7,j:j+3)=0;
Y_G(i+4:i+7,j:j+3)=0;
Y_B(i+4:i+7,j:j+3)=0;
Y_R(i+4:i+7,j+4:j+7)=0;
Y_G(i+4:i+7,j+4:j+7)=0;
Y_B(i+4:i+7,j+4:j+7)=0;
```

計算 PSNR (with 原圖)：同 n=8 部分

結果：得到的 PSNR=35.6460

我使用內建 function psnr 檢查我自己寫的 psnr 是否正確，得到的結果也跟自己計算的 PSNR 一致。

③n=2

基本上跟 n=8 的程式碼差不多，只是差在，row>2 || column>2 的部分要歸 0。

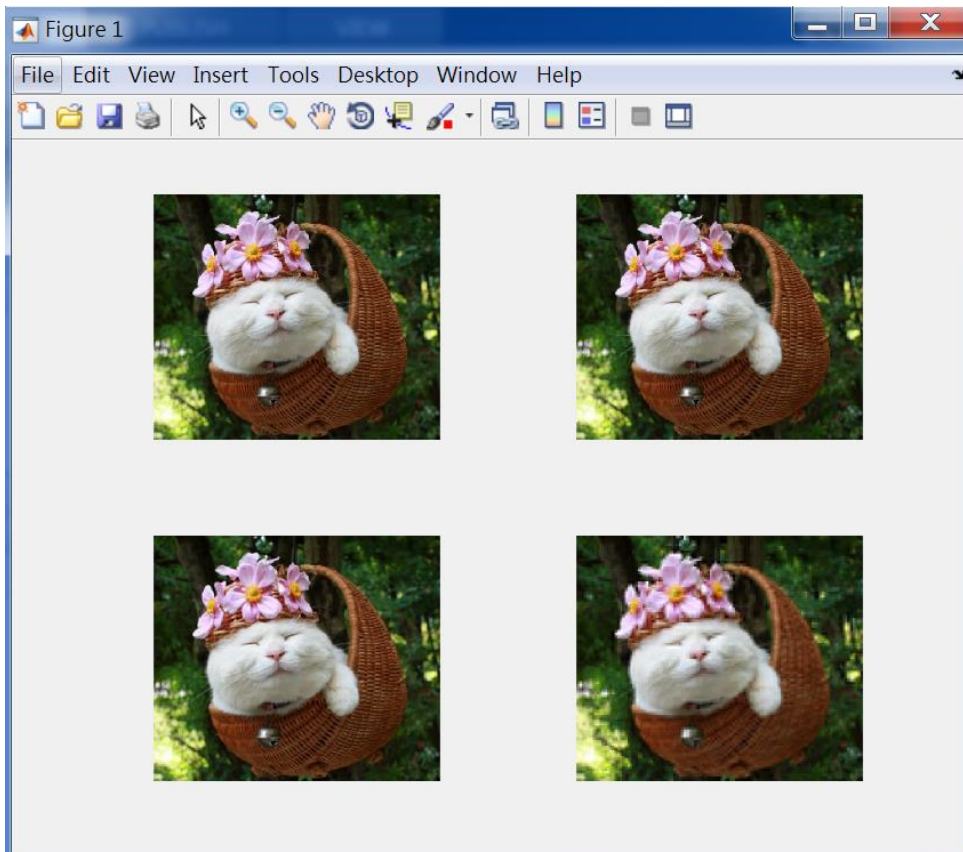
```
Y_R(i:i+1,j+2:j+7)=0;
Y_G(i:i+1,j+2:j+7)=0;
Y_B(i:i+1,j+2:j+7)=0;
Y_R(i+2:i+7,j:j+1)=0;
Y_G(i+2:i+7,j:j+1)=0;
Y_B(i+2:i+7,j:j+1)=0;
Y_R(i+2:i+7,j+2:j+7)=0;
Y_G(i+2:i+7,j+2:j+7)=0;
Y_B(i+2:i+7,j+2:j+7)=0;
```

計算 PSNR (with 原圖)：同 n=8 部分

結果：得到的 PSNR=27.2944

我使用內建 function psnr 檢查我自己寫的 psnr 是否正確，得到的結果也跟自己計算的 PSNR 一致。

④結果：左上→原圖，右上→n=8，左下→n=4，右下→n=2



(b) transformed to YIQ color model

① n=8 以下為實做的程式碼：

程式碼	解釋
<pre>original_picture=imread('cat1.png'); picture_double=im2double(original_picture); s=size(original_picture); im_R= picture_double(:,:,1); im_G= picture_double(:,:,2); im_B= picture_double(:,:,3); t=[0.299 0.587 0.114;0.596 -0.275 -0.321;0.212 -0.523 0.311]; l=[1 0.956 0.620;1 -0.272 -0.647;1 -1.108 1.705];</pre>	<p>1.先將 cat1.png 的圖片讀進來。</p> <p>2.將影像矩陣轉成 double 型態。</p> <p>3. s 用來存原圖的 row 和 column 的值。</p> <p>4. 將原圖分成 3 個 channels 存起來。</p> <p>5.定義 RGB 轉換成 YIQ 及 YIQ 轉換成 RGB 需乘上的矩陣。</p>
<pre>for i=1:s(1) for j=1:s(2) if rem(i,8)==1 && rem(j,8)==1 for a=0:7 for b=0:7 yiq(1:3)=t*[im_R(i+a,j+b);im_G(i+a,j+b);im_B(i+a,j+b)]; Y_Y(a+1,b+1)=yiq(1); Y_I(a+1,b+1)=yiq(2); Y_Q(a+1,b+1)=yiq(3); end end for a=0:7 for b=0:7 if a==0 c=sqrt(1/8); else c=sqrt(2/8); end A(a+1,b+1)=c*cos(pi*(b+0.5)*a/8); end end Y_R(i:i+7,j:j+7)=A*Y_Y*A'; Y_G(i:i+7,j:j+7)=A*Y_I*A'; Y_B(i:i+7,j:j+7)=A*Y_Q*A'; YI_R(i:i+7,j:j+7)=A'*Y_R(i:i+7,j:j+7)*A; YI_G(i:i+7,j:j+7)=A'*Y_G(i:i+7,j:j+7)*A; YI_B(i:i+7,j:j+7)=A'*Y_B(i:i+7,j:j+7)*A; end end for al=0:7 for bl=0:7 rgb(1:3)=l*[YI_R(al+i,b1+j);YI_G(al+i,b1+j);YI_B(al+i,b1+j)]; R_R(i+al,j+b1)=rgb(1); R_G(i+al,j+b1)=rgb(2); R_B(i+al,j+b1)=rgb(3); end end end end</pre>	<p>1.用 2 個 for 迴圈跑完原圖的所有 pixels，因為要把原圖切成許多 8*8 小圖，所以設一個條件： 每遇到 row 和 column 除以 8 的餘數為 1 才需要執行那一塊小圖的轉換。</p> <p>2. RGB 轉換成 YIQ：</p> $\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$ <p>YIQ 轉換成 RGB：</p> $\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.956 & 0.620 \\ 1 & -0.272 & -0.647 \\ 1 & -1.108 & 1.705 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$ <p>3.先把 RGB 的 8*8 小圖轉換成 YIQ 圖再做 DCT compression。做完後再把圖轉回 RGB。</p>
<pre>result_N8=cat(3,R_R,R_G,R_B);</pre>	<p>用 cat 函數構造多維矩陣。</p>

計算 PSNR (with 原圖)：同(a)小題 n=8 部分

結果：得到的 PSNR=Inf

我使用內建 function psnr 檢查我自己寫的 psnr 是否正確，得到的結果也跟自己計算的 PSNR 一致。

②n=4

基本上跟 n=8 的程式碼差不多，只是差在，row>4 || column>4 的部分要歸 0。

```

Y_R(i:i+3,j+4:j+7)=0;
Y_G(i:i+3,j+4:j+7)=0;
Y_B(i:i+3,j+4:j+7)=0;
Y_R(i+4:i+7,j:j+3)=0;
Y_G(i+4:i+7,j:j+3)=0;
Y_B(i+4:i+7,j:j+3)=0;
Y_R(i+4:i+7,j+4:j+7)=0;
Y_G(i+4:i+7,j+4:j+7)=0;
Y_B(i+4:i+7,j+4:j+7)=0;

```

計算 PSNR (with 原圖)：同(a)小題 n=8 部分

結果：得到的 PSNR=35.6455

我使用內建 function psnr 檢查我自己寫的 psnr 是否正確，得到的結果也跟自己計算的 PSNR 一致。

③n=2

基本上跟 n=8 的程式碼差不多，只是差在，row>2 || column>2 的部分要歸 0。

```

Y_R(i:i+1,j+2:j+7)=0;
Y_G(i:i+1,j+2:j+7)=0;
Y_B(i:i+1,j+2:j+7)=0;
Y_R(i+2:i+7,j:j+1)=0;
Y_G(i+2:i+7,j:j+1)=0;
Y_B(i+2:i+7,j:j+1)=0;
Y_R(i+2:i+7,j+2:j+7)=0;
Y_G(i+2:i+7,j+2:j+7)=0;
Y_B(i+2:i+7,j+2:j+7)=0;

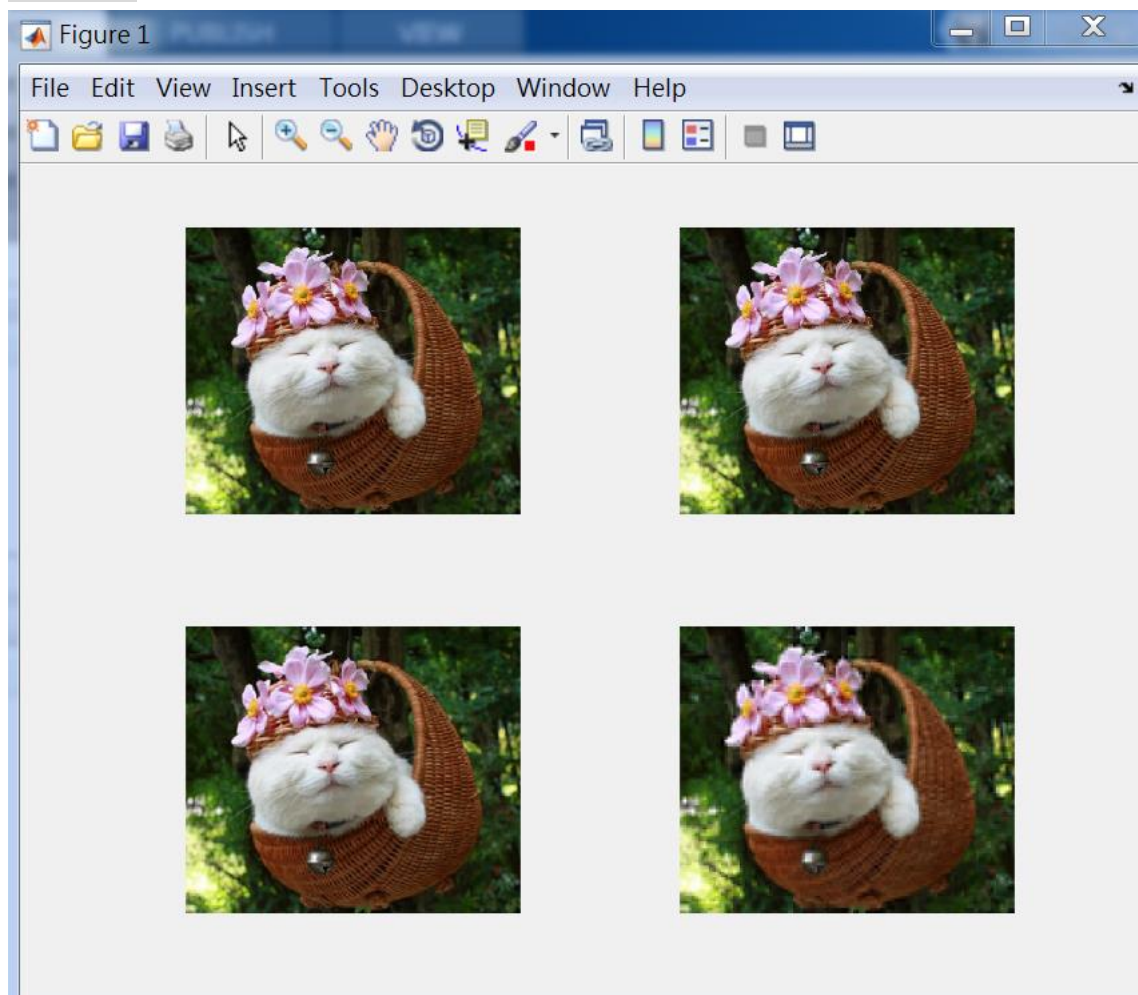
```

計算 PSNR (with 原圖)：同(a)小題 n=8 部分

結果：得到的 PSNR=27.2944

我使用內建 function psnr 檢查我自己寫的 psnr 是否正確，得到的結果也跟自己計算的 PSNR 一致。

④結果：左上→原圖，右上→n=8，左下→n=4，右下→n=2



(c) 比較 color spaces in (a) and (b)

雖然(a) and (b) PSNR 值都差不多，但經過 YIQ 的 PSNR 理論上會稍微高一點點，但可能有一下誤差所以 n=2 的一樣，n=4 的 YIQ 稍微小一點點。

	(a)RGD	(b)YIQ
n=8	PSNR=Inf 	PSNR=Inf 
n=4	PSNR=35.6460 	PSNR=35.6455 
n=2	PSNR=27.2944 	PSNR=27.2944 

2. part2

(a)noise(random) dithering 以下為實做的程式碼：

程式碼	解釋
<pre>original_picture=imread('cat2_gray.png'); result_random=zeros(size(original_picture)); s=size(original_picture);</pre>	1.先將 cat2_gray.png 的圖片讀進來。 2.產生一個維度與原圖一樣且元素均為 0 的矩陣，用來存轉換後的結果。 3. s 用來存原圖的 row 和 column 的值。

<pre> for i=1:s(1) for j=1:s(2) random=unidrnd(256); R=random-1; if original_picture(i,j)>R result_random(i,j)=1; else result_random(i,j)=0; end end end end </pre>	<p>用 2 個 for 迴圈跑完原圖的所有 pixels，拿每個 pixel 的值與 R 值比。</p> <p>每個 pixels 產生一個隨機的亂數，unidrnd(N)的做法是產生 1 到 N 之間的隨機數。</p> <p>但根據講義，欲得到的亂數應介於 0 到 255 之間，所以我把 N 設成 256 再減 1 才得到最後的 R。</p> <p>若 pixel 值較大→white(pixel 值設成 1)</p> <p>若反之→black(pixel 值設成 0)</p>
--	--

(b)average dithering 以下為實做的程式碼：

程式碼	解釋
<pre> >> original_picture=imread('cat2_gray.png'); >> result=zeros(size(original_picture)); >> total=sum(original_picture(:)); >> s=size(original_picture); >> average=total/(s(1)*s(2)); >> average_floor=floor(average); </pre>	<ol style="list-style-type: none"> 1.先將 cat2_gray.png 的圖片讀進來。 2. 產生一個維度與原圖一樣且元素均為 0 的矩陣，用來存轉換後的結果。 3.利用 sum(矩陣名(:))將 original_picture 所有元素都相加。 4. s 用來存原圖的 row 和 column 的值。 5.算出平均 pixel value，並捨去小數部分。
<pre> >> for i=1:s(1) for j=1:s(2) if(original_picture(i,j)>average_floor) result(i,j)=1; else result(i,j)=0; end end end end </pre>	<p>用 2 個 for 迴圈跑完原圖的所有 pixels，拿每個 pixel 的值與平均 pixel value 比。</p> <p>若 pixel 值較大→white(pixel 值設成 1)</p> <p>若反之→black(pixel 值設成 0)</p>

(c)error diffusion dithering 以下為實做的程式碼：

程式碼	解釋
<pre> >> original_picture=imread('cat2_gray.png'); >> result=zeros(size(original_picture)); >> s=size(original_picture); </pre>	<ol style="list-style-type: none"> 1.先將 cat2_gray.png 的圖片讀進來。 2. 產生一個維度與原圖一樣且元素均為 0 的矩陣，用來存轉換後的結果。 3. s 用來存原圖的 row 和 column 的值。
<pre> for i=1:s(1) for j=1:s(2) if original_picture(i,j)<128 e=original_picture(i,j); else e=original_picture(i,j)-255; end if j<s(2) original_picture(i,j+1)=((7/16)*e)+original_picture(i,j+1); end if i<s(1) && j>1 original_picture(i+1,j-1)=((3/16)*e)+original_picture(i+1,j-1); end if i<s(1) original_picture(i+1,j)=((5/16)*e)+original_picture(i+1,j); end if i<s(1) original_picture(i+1,j)=((5/16)*e)+original_picture(i+1,j); end if i<s(1) && j<s(2) original_picture(i+1,j+1)=((1/16)*e)+original_picture(i+1,j+1); end end end end </pre>	<p>根據講義，把 error distributes over the whole image。</p> <p>對每個 pixel：(用 2 個 for 迴圈跑)</p> <ol style="list-style-type: none"> 1.定義 e： <p>若 pixel<128→e=pixel，否則 e=pixel-255</p> 2.Change the value of neighbor pixel： <p>公式</p> $p(x+1,y)= p(x+1,y)+(7/16)*e$ $p(x-1,y+1)= p(x-1,y+1)+(3/16)*e$ $p(x,y+1)= p(x,y+1)+(5/16)*e$ $p(x+1,y+1)= p(x+1,y+1)+(1/16)*e$ <p>但需要注意邊界條件。</p>

```

for i=1:s(1)
    for j=1:s(2)
        if original_picture(i,j)<128
            result(i,j)=0;
        else
            result(i,j)=1;
        end
    end
end
end

```

error distributes over the whole image 之後，
用 2 個 for 迴圈跑完原圖的所有 pixels，拿
每個 pixel 的值與 128 比。
若 pixel 值較小→(pixel 值設成 0)
若反之→black(pixel 值設成 1)

(d)比較 3 種 dithering 的結果

方法	轉換後圖片	特點
noise(random) dithering		將原圖的每一個 pixels 與其隨機產生的一個值比較。 若 pixel 值較大→white(pixel 值設成 1) 若反之→black(pixel 值設成 0)
average dithering		將原圖的每一個 pixel 與平均 pixel 值比較。 若 pixel 值較大→white(pixel 值設成 1) 若反之→black(pixel 值設成 0) 原圖偏暗的地方可能整塊變成黑色的，而偏亮部分可能整塊變成白色的。 轉換後的圖對比明顯。
error diffusion dithering		因為 1 個算出來的 pixel 會影響到下一個 pixels，此方法可修正一些亮度判定錯誤的問題，圖片看起來比較接近原圖，較有漸層感。

3. part3

(a) nearest-neighbor interpolation 以下為實做的程式碼：

程式碼	解釋
<pre> >> original_picture=imread('cat3_LR.png'); >> s=size(original_picture); >> picture_double=im2double(original_picture); </pre>	<ol style="list-style-type: none"> 1.先將 cat3_LR.png 的圖片讀進來。 2. s 用來存原圖的 row 和 column 的值。 3.將影像矩陣轉成 double 型態。

<pre> for i=1:s(1)*4 for j=1:s(2)*4 x=round(i/4); y=round(j/4); if x==0 x=1; end if y==0 y=1; end result_neighbor(i,j,1)=picture_double(x,y,1); result_neighbor(i,j,2)=picture_double(x,y,2); result_neighbor(i,j,3)=picture_double(x,y,3); end end end </pre>	<p>nearest-neighbor interpolation，將放大後的圖像映射到原圖，值取離映射點最接近的 pixel 值。</p> <p>將放大 4 倍後的 pixel 一個個映射到原矩陣。</p> <p>$x=\text{round}(i/4) \rightarrow x$ 會取四捨五入的值，y 亦然。</p> <p>邊界條件是 x 和 y 都不可為 0，若四捨五入完 x,y 為 0 則改成 1。</p> <p>最後把離映射點最接近的 pixel 值存到 result 對應的 pixel。</p>
--	--

計算 PSNR (with cat3_HR.png)：以下為實做的程式碼：(接著上面的 code)

程式碼	解釋
<pre> >> cat_HR=imread('cat3_HR.png'); >> mseR=(double(cat_HR(:,:,1))-double(result_uint(:,:,1))).^2; >> mseG=(double(cat_HR(:,:,2))-double(result_uint(:,:,2))).^2; >> mseB=(double(cat_HR(:,:,3))-double(result_uint(:,:,3))).^2; </pre>	<p>1.先將 cat3_HR.png 的圖片讀進來。</p> <p>2.先算 MSE：</p> <p>各自算三個 channel 的 MSE 再取平均。</p> <p>將 cat3_HR.png 的 image 矩陣跟做完 nearest-neighbor interpolation 的 image 矩陣相減再平方。</p>
<pre> >> mR=sum(sum(mseR))/(s(1)*4*s(2)*4); >> mG=sum(sum(mseG))/(s(1)*4*s(2)*4); >> mB=sum(sum(mseB))/(s(1)*4*s(2)*4); >> mse=(mR+mG+mB)/3; >> PSNR=10*log10(255^2/mse); </pre>	<p>1.將第 2 點算完的結果矩陣的每個 pixels 加總再除以矩陣 row*column，分別得到三個 channel 的 MSE。</p> <p>2. 三個 channel 的 MSE 取平均，得到 mse 值。再拿 mse 帶入 PSNR 公式。</p>

結果：得到的 PSNR=19.3705

我使用內建 function psnr 檢查我自己寫的 psnr 是否正確，得到的結果也跟自己計算的 PSNR 一致。

(b) bilinear interpolation 以下為實做的程式碼：

程式碼	解釋
<pre> >> original_picture=imread('cat3_LR.png'); >> s=size(original_picture); >> picture_double=im2double(original_picture); </pre>	<p>1.先將 cat3_LR.png 的圖片讀進來。</p> <p>2. s 用來存原圖的 row 和 column 的值。</p> <p>3.將影像矩陣轉成 double 型態。</p>
<pre> >> for i=1:s(1)*4 for j=1:s(2)*4 r=i/4; c=j/4; rf=floor(r); cf=floor(c); if(rf<1) rf=1; end if(cf<1) cf=1; end if(rf>249) rf=249; end if(cf>375) cf=375; end deltaR=r-rf; deltaC=c-cf; result(i,j,1)=picture_double(rf,cf,1)*(1-deltaR)*(1-deltaC) result(i,j,2)=picture_double(rf,cf,2)*(1-deltaR)*(1-deltaC) result(i,j,3)=picture_double(rf,cf,3)*(1-deltaR)*(1-deltaC) end end end </pre>	<p>bilinear interpolation，將放大後的圖像映射到原圖，映射點的四個鄰近的 pixels 值，依照各自與映射點圍成的四邊形所占比例與 pixel 值相乘再相加得到放大後圖像 pixel 的值。</p> <p>將放大 4 倍後的 pixel 一個個映射到原矩陣。</p> <p>$rf=\text{floor}(i/4) \rightarrow rf$ 會取捨棄小數的值，cf 亦然。</p> <p>邊界條件是 x 和 y 都不可為 0，以及 x 不能為 row(即 250)，y 不能為 column(即 376)。</p> <p>最後把算出來的值存到 result 對應的 pixel。</p> <p>這裡因為程式碼太長所以放到下面(*註解)處。</p>

(*註解)


```

result(i,j,1)=picture_double(rf,cf,1)*(1-deltaR)*(1-deltaC)+
    picture_double(rf+1,cf,1)*(deltaR)*(1-deltaC)+
    picture_double(rf,cf+1,1)*(1-deltaR)*(deltaC)+picture_double(rf+1,cf+1,1)*(deltaR)*(deltaC);
result(i,j,2)=picture_double(rf,cf,2)*(1-deltaR)*(1-deltaC)+
    picture_double(rf+1,cf,2)*(deltaR)*(1-deltaC)+
    picture_double(rf,cf+1,2)*(1-deltaR)*(deltaC)+picture_double(rf+1,cf+1,2)*(deltaR)*(deltaC);
result(i,j,3)=picture_double(rf,cf,3)*(1-deltaR)*(1-deltaC)+
    picture_double(rf+1,cf,3)*(deltaR)*(1-deltaC)+
    picture_double(rf,cf+1,3)*(1-deltaR)*(deltaC)+picture_double(rf+1,cf+1,3)*(deltaR)*(deltaC);

```

計算 PSNR (with cat3_HR.png)：跟 nearest-neighbor interpolation 做法一樣。

結果：得到的 PSNR=20.3959

我使用內建 function psnr 檢查我自己寫的 psnr 是否正確，得到的結果也跟自己計算的 PSNR 一致。

(c) bicubic interpolation 以下為實做的程式碼：

程式碼	解釋
<pre> K>> original_picture=imread('cat3_LR.png'); picture_double=im2double(original_picture); s=size(original_picture); image_R=picture_double(:, :, 1); image_G=picture_double(:, :, 2); image_B=picture_double(:, :, 3); result_r=zeros(s(1)*4,s(2)*4); result_g=zeros(s(1)*4,s(2)*4); result_b=zeros(s(1)*4,s(2)*4); </pre>	<ol style="list-style-type: none"> 1.先將 cat3_LR.png 的圖片讀進來。 2.將影像矩陣轉成 double 型態。 3. s 用來存原圖的 row 和 column 的值。 4.分別取出 3 channels 的值存在 image_r、image_g、image_b。 5.產生 3 個維度為 s(1)*4xs(2)*4，構成元素全為 0 的矩陣。
<pre> K>> for i=1:s(1)*4 for j=1:s(2)*4 x=i/4; y=j/4; rf=floor(x); cf=floor(y); u=x-rf; v=y-cf; gtm=4-8*abs(u+1)+5*abs(u+1)^2-abs(u+1)^3; gtm2=4-8*abs(u-2)+5*abs(u-2)^2-abs(u-2)^3; gun=4-8*abs(v+1)+5*abs(v+1)^2-abs(v+1)^3; gun2=4-8*abs(v-2)+5*abs(v-2)^2-abs(v-2)^3; ftm=1-2*abs(u)^2+abs(u)^3; ftm2=1-2*abs(u-1)^2+abs(u-1)^3; fun=1-2*abs(v)^2+abs(v)^3; fun2=1-2*abs(v-1)^2+abs(v-1)^3; </pre>	<p>bicubic interpolation，將放大後的圖像映射到原圖，映射點的 16 個鄰近的 pixels 值，依照各自公式所算出來的結果與 pixel 值相乘再相加得到放大後圖像 pixel 的值。</p> <ol style="list-style-type: none"> 1.將放大 4 倍後的 pixel 一個個映射到原矩陣。 rf=floor(i/4)→rf 會取捨棄小數的值，cf 亦然。 2.根據講義 81 頁的公式算出對應位置的 f(t(m))、f(u(n))、g(t(m))、g(u(n))。
<pre> xi = max(1,rf-1); xx = max(1,rf); xa = min(250,rf+1); xaa = min(250,rf+2); yi = max(1,cf-1); yy = max(1,cf); ya = min(376,cf+1); yaa = min(376,cf+2); result_r(i,j)=image_R(xi,yi)*gtm* result_g(i,j)=image_G(xi,yi)*gtm* result_b(i,j)=image_B(xi,yi)*gtm* end end </pre>	<p>邊界條件是 16 個 pixels 的 row 和 column 值都不可為 0，以及 row 不能大於 250，column 不能大於 376。最後把算出來的值存到 result_r、result_g、result_g 對應的 pixel。這裡因為程式碼太長所以放到下面(*註解)處。</p>
<pre> result_bicubic= cat(3,result_r,result_g,result_b); </pre>	<p>用 cat 函數構造多維矩陣。</p>

(*註解)

```

result_r(i,j) = image_R(xi,yi)*gtm*gun + image_R(xi,yy)*gtm*fun + image_R(xi,ya)*gtm*fun2;
+ image_R(xi,yaa)*gtm*gun2 + image_R(xx,yi)*ftm*gun + image_R(xx,yy)*ftm*fun;
+ image_R(xx,ya)*ftm*fun2 + image_R(xx,yaa)*ftm*gun2 + image_R(xa,yi)*ftm2*gun;
+ image_R(xa,yy)*ftm2*fun + image_R(xa,ya)*ftm2*fun2 + image_R(xa,yaa)*ftm2*gun2;
+ image_R(xaa,yi)*gtm2*gun + image_R(xaa,yy)*gtm2*fun + image_R(xaa,ya)*gtm2*fun2;
+ image_R(xaa,yaa)*gtm2*gun2;
result_g(i,j) = image_G(xi,yi)*gtm*gun + image_G(xi,yy)*gtm*fun + image_G(xi,ya)*gtm*fun2;
+ image_G(xi,yaa)*gtm*gun2 + image_G(xx,yi)*ftm*gun + image_G(xx,yy)*ftm*fun;
+ image_G(xx,ya)*ftm*fun2 + image_G(xx,yaa)*ftm*gun2 + image_G(xa,yi)*ftm2*gun;
+ image_G(xa,yy)*ftm2*fun + image_G(xa,ya)*ftm2*fun2 + image_G(xa,yaa)*ftm2*gun2;
+ image_G(xaa,yi)*gtm2*gun + image_G(xaa,yy)*gtm2*fun + image_G(xaa,ya)*gtm2*fun2;
+ image_G(xaa,yaa)*gtm2*gun2;
result_b(i,j) = image_B(xi,yi)*gtm*gun + image_B(xi,yy)*gtm*fun + image_B(xi,ya)*gtm*fun2;
+ image_B(xi,yaa)*gtm*gun2 + image_B(xx,yi)*ftm*gun + image_B(xx,yy)*ftm*fun;
+ image_B(xx,ya)*ftm*fun2 + image_B(xx,yaa)*ftm*gun2 + image_B(xa,yi)*ftm2*gun;
+ image_B(xa,yy)*ftm2*fun + image_B(xa,ya)*ftm2*fun2 + image_B(xa,yaa)*ftm2*gun2;
+ image_B(xaa,yi)*gtm2*gun + image_B(xaa,yy)*gtm2*fun + image_B(xaa,ya)*gtm2*fun2;
+ image_B(xaa,yaa)*gtm2*gun2;

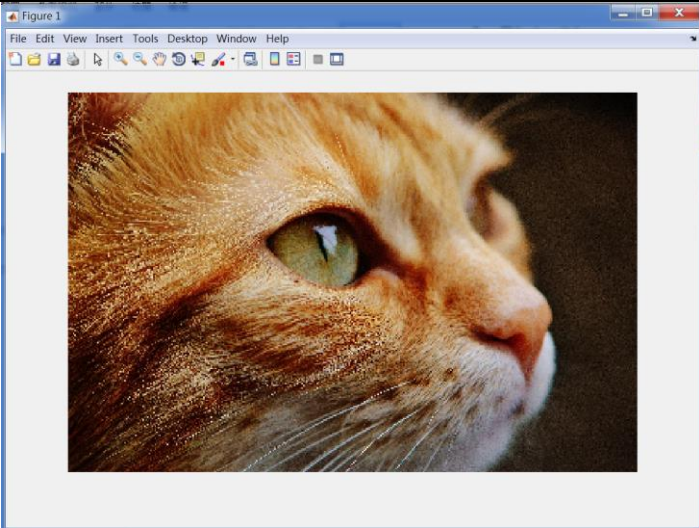
```

計算 PSNR (with cat3_HR.png)：跟 nearest-neighbor interpolation 做法一樣。

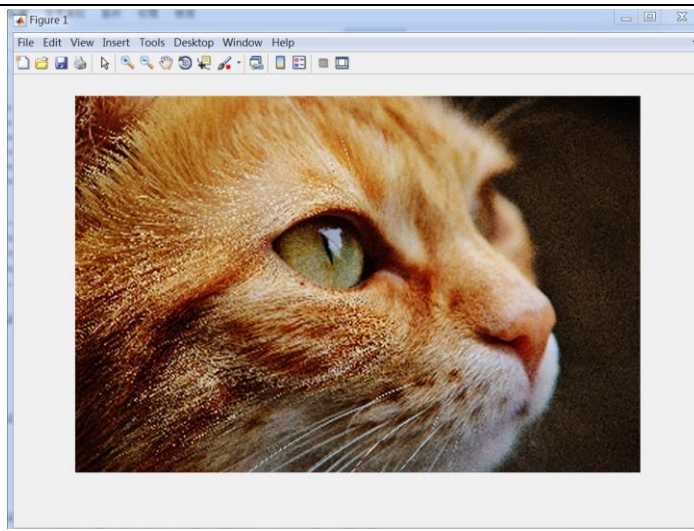
結果：得到的 PSNR=19.5994

我使用內建 function psnr 檢查我自己寫的 psnr 是否正確，得到的結果也跟自己計算的 PSNR 一致。

(d)比較 3 種 dithering 的結果

方法	轉換後圖片	特點
nearest-neighbor interpolation		PSNR=19.3705 PSNR 值越大，代表失真越少，所以 nearest-neighbor interpolation 失真最多，bicubic interpolation 次之，bilinear interpolation 失真最少。 通常 PSNR 值越高表示品質越好。
bilinear interpolation		PSNR=20.3959

**bicubic
interpolation**



PSNR=19.5994