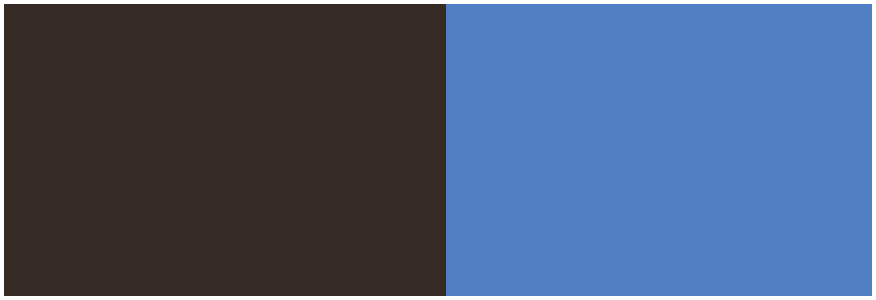
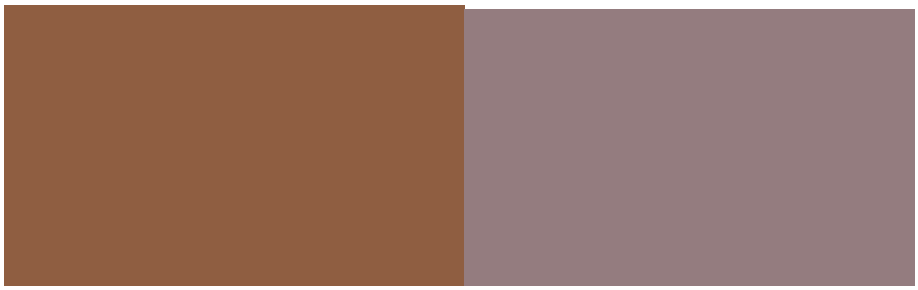
HW1-1 color quantization

102062138 陳嘉茹

(a)Show the eight color patches(按照順序):





Code implementation:

```
color = [54.4604265315182,42.1025451316958,37.0031814146197;
    84.6089999570613,70.5372600798660,78.1668706256173;
    80.3121344222492,126.229435676710,195.942693963139
    ;212.816621647431,170.490682382473,109.018853510136;
    178.386394827131,192.823489047242,211.188123515439;
    228.518085331409,229.685625054436,233.265226642113;
    143.931762510206,94.3801275321747,65.5584781678914;
    148.673459039177,124.089415167337,127.7188521203471;
for i = 1:8
    matrix = color(i,:); %A([1 4], :) = []
    matrix = matrix /256;
    set(gcf,'Color',matrix);
    imsave(gcf)
end
```

把原本的 color.mat 讀入，為一 8x3RGB 的 array，故一個 row 恰好代表一個顏色，共有 8 個 indexed color。Normalize 成[0,1]之間的值後，套用 set 畫出顏色，依序八個顏色如上所示。

(b)implement the color quantization and show the image.

After color quantization:

Code implementation:

```
clc;
clear all;
close all;
img = imread('clash1.png');
%[imgQ,map]= rgb2ind(img,8,'nodither');
%imshow(imgQ,map);
imgVec=[reshape(img(:,:,1),[],1) reshape(img(:,:,2),[],1) reshape(img(:,:,3),[],1)];
imgVecCenter = [54.4604265315182,42.1025451316958,37.0031814146197;
    84.6089999570613,70.5372600798660,78.1668706256173;
    80.3121344222492,126.229435676710,195.942693963139;
    212.816621647431,170.490682382473,109.018853510136;
    178.386394827131,192.823489047242,211.188123515439;
    228.518085331409,229.685625054436,233.265226642113;
    143.931762510206,94.3801275321747,65.5584781678914;
    148.673459039177,124.089415167337,127.718852120347];
imgVecQ=pdist2(imgVec,imgVecCenter); %choosing the closest centroid to each pixel,
[~,indMin]=min(imgVecQ,[],2);   %avoiding double for loop
imgVecNewQ=imgVecCenter(indMin,:);  %quantizing
imgNewQ=img;
imgNewQ(:,:,1)=reshape(imgVecNewQ(:,1),size(img(:,:,1))); %arranging back into image
imgNewQ(:,:,2)=reshape(imgVecNewQ(:,2),size(img(:,:,1)));
imgNewQ(:,:,3)=reshape(imgVecNewQ(:,3),size(img(:,:,1)));
figure,imshow(imgNewQ,[]);
```

一開始使用 reshape 把原本讀入的 image 轉成三個 column 多個 row 的形式。imgVecCenter 則為一開始題目給定的八個 indexed color。接下來就套用了講義上的 popularity algorithm 定義，要把每個原始點對應到 indexed color，需用 minimum mean square distance 的求法，pdist2 這個 function 求距離，後面 min 取最小，最後把每個原始點對應到 indexed color，再次 reshape 後輸出。

(c)calculate the frequency of these colors, also use Shannon's entropy equation to determine the number of bits to encode the color.

Code Implementation:

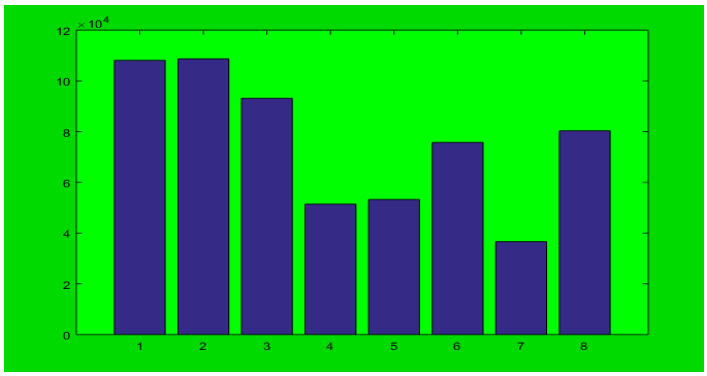
```
imgVecQ=pdist2(imgVec,imgVecCenter); %choosing the closest centroid to each pixel,
[~,indMin]=min(imgVecQ,[],2);   %avoiding double for loop
imgVecNewQ=imgVecCenter(indMin,:);  %quantizing
imgVecQCount = imgVecNewQ(:,1);
testCount = hist(imgVecQCount, numel(unique(imgVecQCount)));
totalCount = histc(imgVecQCount, unique(imgVecQCount));
reshapeCount = reshape(totalCount,[1,8])
bar(reshapeCount)
```

顏色出現頻率如下：

```
reshapeCount =

      108128      108732       93156       51438       53246       75780       36651       80369
```

用 bar 作圖，Y 軸為出現頻率，X 軸為一到八個 indexed color，如下：



為帶入 shannon's entropy，須轉為 probability distribution 的形式，normalize 後帶入公式，可得 optimal 的 encoding bit 數量。

Code Implementation:

```
imgVecQCount = imgVecNewQ(:,1);
testCount = hist(imgVecQCount, numel(unique(imgVecQCount)));
totalCount = histc(imgVecQCount, unique(imgVecQCount));
reshapeCount = reshape(totalCount,[1,8]);
total = sum(reshapeCount);
prob = reshapeCount/total
optNum = -log2(prob(prob>0))
H = sum(-(prob(prob>0).*(log2(prob(prob>0)))))
```

根據公式：

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

其中 pi 為 probability distribution，而 log2(pi)即為最佳的 encoding bits。
全部相乘取和則為整體而言，要表達一個 pixel，平均需要幾個 bits。
主要集中在下列三行：

```
prob = reshapeCount/total
optNum = -log2(prob(prob>0))
H = sum(-(prob(prob>0).*(log2(prob(prob>0)))))
```

可得輸出為：

```
prob =

    0.1780    0.1790    0.1533    0.0847    0.0876    0.1247    0.0603    0.1323


optNum =

    2.4901    2.4821    2.7052    3.5620    3.5121    3.0030    4.0510    2.9182


H =

    2.9168
```

optNum 即為每個顏色需要幾個 bit 來 encode，而 H 為整體平均。
每個顏色需要幾個 bit 來 encode(經 optNum 取 ceil 後) =
3 3 3 4 4 4 5 3
平均每個 symbol 需要 2.9168 個 bits 來 encode

(c)Using Shannon-Fano's algorithm instead.

Code Implementation:

整個 algo 的 idea 是出現機率高的用少一點 bits 來 encode，反之少出現的則用多一點 bits 來表達。而利用機率之和兩邊越接近越好，來作 encode。

```
ss = reshapeCount/total;
ss=sort(ss,'descend');  %the probabilities are sorted in descending order
siling=ceil(log2(1/ss(1))); %initial length is computed
sf=0;
fano=0;
%initializations for Pk
n=1;Hs=0; %initializations for entropy H(s)
for iii=1:length(ss)
    Hs=Hs+ ss(iii)*log2(1/ss(iii)); %solving for entropy
end
```

開始需要先 sort 整個 array，並進行初始化，並且一樣需要計算 entropy。

```
for o=1:length(ss)-1
   fano=fano+ss(o);
   sf=[sf 0]+[zeros(1,o) fano]; %solving for Pk for every codeword
   siling=[siling 0]+[zeros(1,o) ceil(log2(1/ss(o+1)))]; %solving for length every codeword
end
for r=1:length(sf)
    esf=sf(r);
    for p=1:siling(r)
        esf=mod(esf,1)*2;
        h(p)=esf-mod(esf,1); %converting Pk into a binary number
    end
    hh(r)=h(1)*10^(siling(r)-1); %initializtion for making the binary a whole number
    for t=2:siling(r)
        hh(r)=hh(r)+h(t)*10^(siling(r)-t);    %making the binary a whole number
    end                                       %e.g. 0.1101 ==> 1101
end
tao=siling(1)*ss(1); %initialization for codeword length
for u=1:length(ss)-1 %computing for codeword length
   tao=tao+siling(u+1)*ss(u+1);
end
```

整體實作如上。

Result:

每個不同的 color 對定的 encoding bits，如下：

prob	encoding bits
0.1790	3.0000
0.1780	3.0000
0.1533	3.0000
0.1323	3.0000
0.1247	4.0000
0.0876	4.0000
0.0847	4.0000
0.0603	5.0000

```
Hs = 2.9168
T = 3.4177bits/symbol
2.9168 <= 3.4177 <= 3.9168
```

每個顏色需要幾個 bit 來 encode(經 optNum 取 ceil 後) =

3 3 3 3 4 4 4 5

平均每個 symbol 需要 3.4177 個 bits 來 encode

(e)compare the coding efficiency

比較 shannon entropy 跟 shannon-fano 的 coding efficiency，我以每個

symbol 平均需要幾個 bits 來 encode 來最為衡量標準。其中 entropy 需要 2.9168，而 fano 需要 3.4177，可見 entropy 較 fano 快了 3.4177/2.9168 = 1.1717 倍。

直觀而言亦是如此，entropy 是理論值，fano 則是實驗值，我想差異可能來自 Fano 要盡力確保 tree 的 2 個 node 相等，但是可能 frequency 本來就無法等分。

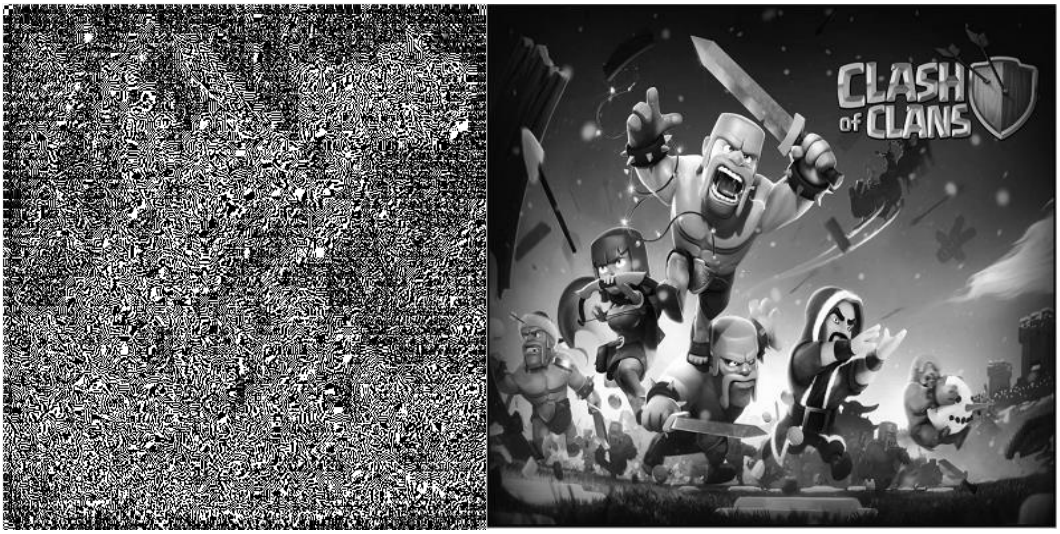
HW1-2 DCT image compression

(a)implement the dct with upper-left n*n filter

N = 2

N = 4

N = 8



(b)

N = 2

Psnr = 20.4930

N = 4

Psnr = 26.5295

N = 8

Psnr = 313.12(依照理論，mse 應該為 0，但檢測算出來的結果為 E-34 次方，因而 psnr 非預期中的趨近無限大，但相去不遠。)

PSNR value:

峰值信噪比（PSNR），一種評價圖像的客觀標準。它具有局限性，PSNR 是「Peak Signal to Noise Ratio」的縮寫。peak 的中文意思是頂點，而 ratio 的意思是比率或比列的，整個意思就是到達噪音比率的頂點信號，PSNR 一般是用於最大值信號和背景噪音之間的一個工程項目。通常在經過影像壓縮之後，輸出的影像在某種程度都會與原始影像不同。為了衡量經過處理後的影像品質，我們通常會參考 PSNR 值來衡量某個處理程序能否令人滿意。它是原圖像與被處理圖像之間的均方誤差相對於(2^n-1)^2 的對數值(信號最大值的平方，n 是每個採樣值的比特數)，它的單位是 dB。其中，MSE 是原圖像與處理圖像之間均方誤差。

PSNR 值越大，就代表失真越少。這是一個客觀的評比數據。但有時候並不能完全代表人的主觀感受。

Code implementation:

```
for i = 0: blocksize - 1
  for j = 0: blocksize - 1
    if i == 0
        DCT_trans(i+1,j+1) = sqrt(1/blocksize);
        %disp(i);
    else
        DCT_trans(i+1, j+1) = sqrt(2 / blocksize)* cos ((2 * j + 1) * i * pi / (2 * blocksize));
        %disp(j)
    end
  end
end
```

這個公式主要來自 matlab 網站上的公式：

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

接著開始以 8x8 一個區塊進行 transform：

```
for i = 0: cols-1
    for j = 0: rows-1
        DCT_matrix = img([i*8+1: (i+1)*8],[j*8+1: (j+1)*8]);
        temp = (DCT_trans)*(DCT_matrix)*(DCT_trans');
        temp = temp.* mask_4;
        output([i*8+1: (i+1)*8],[j*8+1: (j+1)*8]) = temp;
    end
end
```

其中的數學是來自官網的介紹：

```
T = dctmtx(8);
dct = @(block_struct) T * block_struct.data * T';
B = blockproc(I,[8 8],dct);
```

Upper-left 的 nxn matrix 設計如下，在需要留下的地方設為 1：

```
mask_2 = [1   1   0   0   0   0   0   0
          1   1   0   0   0   0   0   0
          0   0   0   0   0   0   0   0
          0   0   0   0   0   0   0   0
          0   0   0   0   0   0   0   0
          0   0   0   0   0   0   0   0
          0   0   0   0   0   0   0   0
          0   0   0   0   0   0   0   0];
```

接著進行反轉換，轉回原圖：

```
for i = 0: cols-1
    for j = 0: rows-1
        inverseDCT_matrix = output([i*8+1: (i+1)*8],[j*8+1: (j+1)*8]);
        temp = (DCT_trans')*(inverseDCT_matrix)*(DCT_trans);
        temp = (temp);
        reconstruct_output([i*8+1: (i+1)*8],[j*8+1: (j+1)*8]) = temp;
    end
end
```

最後計算 PSNR，一樣套用公式：

```
mse=0;
mse=mse+sum(sum((img-reconstruct_output).^2));
mse=mse/(rows*cols*64);
psnrx=20*log10(1)-10*log10(mse);
```

即可獲得。並且可以由結果看到，mask = 2 時，留下來的比例較少，轉換完較模糊，一路 mask 放大後，轉換比例即較好，PSNR 漸大，並且 DCT 為完全沒有失真的轉換，因而轉完後與原圖比較，PSNR 應趨近無窮大。

HW1-3 image interpolation

(a) nearest-neighbor interpolation

Result

Code implementation:

```
inputImage = imread('clash3.png');
figure, imshow(inputImage)
multiple = [4 4];              %# The resolution scale factors: [rows columns]
oldSize = size(inputImage);                  %# Get the size of your image
newSize = max(floor(multiple.*oldSize(1:2)),1);  %# Compute the new image size
rowIndex = min(round(((1:newSize(1))-0.5)./multiple(1)+0.5),oldSize(1));
colIndex = min(round(((1:newSize(2))-0.5)./multiple(2)+0.5),oldSize(2));
figure, imshow(inputImage(rowIndex,colIndex,:))
```

其中放大為原圖長寬各四倍，設定在 multiple 這個 array 中。並且計算新的 newSize。把原圖做點乘後放大，並且用 floor，取至最近的整數點，最後再取 max 以防發生計算完對應到 0 的情形。

接著再作把每個點的座標軸轉換，把新座標的(x,y)對到原本座標的值，再找最近的，matlab 內建的函數 round 即可找到最近值。

(b) bilinear interpolation

Result:

Code Implementation:

主要的 coding 是按照這張圖：

Let $\mathbf{I}$ be an $R \times C$ image.
We want to resize $\mathbf{I}$ to $R' \times C'$.
Call the new image $\mathbf{J}$.

Let $s_R = R / R'$ and $s_C = C / C'$.

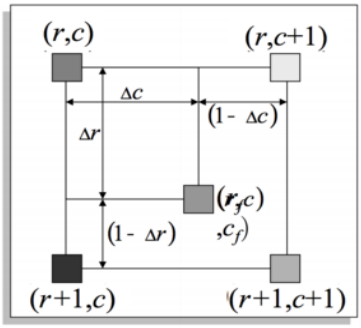
Let $r_f = r' \cdot s_R$ for $r' = 1, \ldots, R'$

and $c_f = c' \cdot s_C$ for $c' = 1, \ldots, C'$.

Let $r = \lfloor r_f \rfloor$ and $c = \lfloor c_f \rfloor$.

Let $\Delta r = r_f - r$ and $\Delta c = c_f - c$.

Then $\mathbf{J}(r', c') = \mathbf{I}(r, c) \cdot (1 - \Delta r) \cdot (1 - \Delta c)$
$+ \mathbf{I}(r+1, c) \cdot \Delta r \cdot (1 - \Delta c)$
$+ \mathbf{I}(r, c+1) \cdot (1 - \Delta r) \cdot \Delta c$
$+ \mathbf{I}(r+1, c+1) \cdot \Delta r \cdot \Delta c.$




```
img = imread('clash3.png');
in_rows = size(img,1);
in_cols = size(img,2);
out_rows = size(img,1)*4;
out_cols = size(img,1)*4;
ratio_X = in_rows / out_rows;
ratio_Y = in_cols / out_cols;
[newY, newX] = meshgrid(1 : out_cols, 1 : out_rows);
newX = newX * ratio_X;
newY = newY * ratio_Y;
x = floor(newX);
y = floor(newY);
```

一開始的部分，同樣計算座標軸轉換的倍率，新的(x,y)座標會落在哪。

```
x(x < 1) = 1;
y(y < 1) = 1;
x(x > in_rows - 1) = in_rows - 1;
y(y > in_cols - 1) = in_cols - 1;
delta_X = newX - x;
delta_Y = newY - y;
```

接著處理邊界的情形，並且計算 delta_X 跟 delta_Y，分別對應到圖上，現在座標點到左上端點的距離。

```
in1_ind = sub2ind([in_rows, in_cols], x, y);
in2_ind = sub2ind([in_rows, in_cols], x+1,y);
in3_ind = sub2ind([in_rows, in_cols], x, y+1);
in4_ind = sub2ind([in_rows, in_cols], x+1, y+1);
out = zeros(out_rows, out_cols, size(im, 3));
out = cast(out, class(im));
```

接著利用 sub2ind 可以把值取出後，把 out 先初始化為 0，接著下面進入了 bilinear 的計算。

```
for idx = 1 : size(img, 3)
    chan = double(img(:,:,idx)); %// Get i'th channel
    %// Interpolate the channel
    tmp = chan(in1_ind).*(1 - delta_X).*(1 - delta_Y) + ...
                  chan(in2_ind).*(delta_X).*(1 - delta_Y) + ...
                  chan(in3_ind).*(1 - delta_X).*(delta_Y) + ...
                  chan(in4_ind).*(delta_X).*(delta_Y);
    out(:,:,idx) = cast(tmp, class(img));
end
figure,imshow(out)
```

內部計算即是根據定義而來，以區塊大小代表機率，並且乘以對角的顏色。

(c)discuss the difference

由 output 出來的圖形可以明顯的看出 nearest neighbor 的演算法對於原圖的保留較差，而 bilinear 反之較佳，失真程度較小。

並且其實由直觀來想也是如此，nearest neighbor 的情形是很武斷的以一個最相近的顏色來填充縮放所造成的新區域。反之 blinear 的方式則是使用了以距離為權重的方式，適當地融合了每個鄰近區域的顏色。

並且根據查到的資料，nearest neighbor interpolation 的情形會有鋸齒狀出現，並且轉換畫質欠佳，適用於圖示、簡單圖檔或線條。反之 Bilinear 則畫質普通，適用於風景照等，但少了前述鋸齒狀的邊，由這次作業也可以觀察到這個現象。

參考資料：

http://stackoverflow.com/questions/22074941/shannons-entropy-calculation
https://en.wikipedia.org/wiki/Entropy_%28information_theory%29#Definition
http://www.mathworks.com/matlabcentral/fileexchange/41727-shannon-fano-encoder/content/sfencoderkasan.m
http://stackoverflow.com/questions/2880933/how-can-i-count-the-number-of-elements-of-a-given-value-in-a-matrix
http://stackoverflow.com/questions/26142288/resize-an-image-with-bilinear-interpolation-without-imresize
http://www.mathworks.com/help/images/discrete-cosine-transform.html#f21-16137