HW3

Hongru Li 95318168

1 Redundancy in the Color Space

```
I = imread('peppers.png');
Ycbcr = rgb2ycbcr(I);
Y = Ycbcr(:,:,1);
Cb = Ycbcr(:,:,2);
Cr = Ycbcr(:,:,3);
Y_down = imresize(Y, 0.5, 'bilinear');
Y_up = imresize(Y_down, 2, 'bilinear');
Cb_down = imresize(Cb, 0.5, 'bilinear');
Cb_up = imresize(Cb_down, 2, 'bilinear');
Cr_down = imresize(Cr, 0.5, 'bilinear');
Cr_up = imresize(Cr_down, 2, 'bilinear');
immse(Y,Y_up)
immse(Cb,Cb_up)
immse(Cr,Cr_up)
I_recon_Y = ycbcr2rgb(cat(3, Y_up, Cb, Cr));
I_recon_C = ycbcr2rgb(cat(3, Y, Cb_up, Cr_up));
subplot(1,3,1);imshow(I);title('original');
subplot(1,3,2);imshow(I_recon_Y);title('reconstruct Y downsample');
subplot(1,3,3);imshow(I_recon_C);title('reconstruct Chroma downsample');
```

To compare the result let me put the result of the three questions together:

For Y component, the MSE after down scaling and reconstruction is 16.5003
For Cb component, the MSE after down scaling and reconstruction is 3.9795
For Cr component, the MSE after down scaling and reconstruction is 4.2271

From my understanding, there are mainly two reasons explain why we always to down sample UV(Cb, Cr) channel in video or image compression.

Firstly, as is shown by the MSE value, it is very likely that in a natural picture, the fluctuation of Luminance is morn than Chroma. The consistency in chroma values make Cb and Cr survive better after down sampling.

Another reason is that human eyes is more sensitive to luminance info compared to color info. Thus the loss in color info is hard to be observed by us.

For this experiment, it is clear the picture after chroma down sampling seems much better than Luma down sampling. In real word application, we usually down sample chroma channel (to 4:2:2 or 4:2:0) before compression because this remove redundant info in color channels and do less harm to the video quality.

2 Reconstruction from Projection & CT

(a) According to the class slides,

$$\ln\left(\frac{I_0}{I}\right)\Big|_{line(r,\theta)} = \int_{line(r,\theta)} \mu(x,y)ds = Radon\ transform\ at\ (r,\theta) = Rad(r,\theta)$$

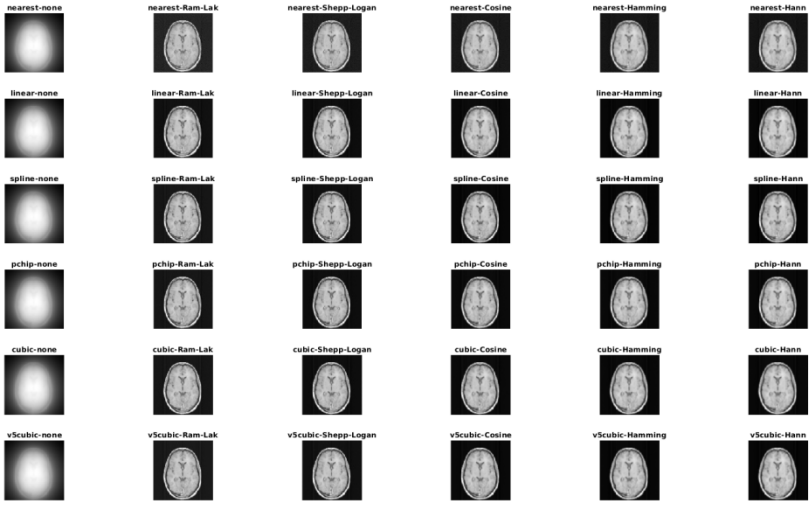
(b) Point a: Rad(7, pi/4)≈ln(5/1)=1.6094
Point b: Rad(6, pi/2)=ln(8/6)=0.2877
Point c: Rad(3, 0)=ln(7/5)=0.3365
Point d: Rad(3√2, 3pi/4)=ln(5/3)=1.6667

(c)

```
interpolators = {'nearest';'linear';'spline';'pchip';'cubic';'v5cubic'};
filters = {'none';'Ram-Lak';'Shepp-Logan';'Cosine';'Hamming';'Hann'};
bestMse = 9999999;
bestSNR = 0;
bestMse_i = 0;
bestMse_j = 0;
bestSNR_i = 0;
bestSNR_j = 0;
for i=1:6
   for j=1:6
      I = iradon(R,angles,char(interpolators(i)),char(filters(j)),128); %back
projection reconstruction
      subplot(6,6,6*(i-1)+j);imshow(I,[]);
title(strcat(char(interpolators(i)),'-',char(filters(j))));
      SE = (I-obj).^2;
      MSE = mean(SE(:)); %Mean Squared Error
      if MSE < bestMse
         bestMse = MSE;
         bestMse_i = i;
         bestMse_j = j;
      end
      SNR = 20*log(norm(obj,'fro')/norm(obj-I,'fro')); %Signal-to-Noise Ratio
      if SNR > bestSNR
         bestSNR = SNR;
         bestSNR_i = i;
         bestSNR_j = j;
      end
   end
end
```

When using ‘spline’+’Ram-Lak’ I got best MSE=3.1112E-4 and SNR=43.5491. Visually I think when the filter is ‘Ram-Lak’ the quality is obviously better than using other filters. But I am not able to distinguish the difference between

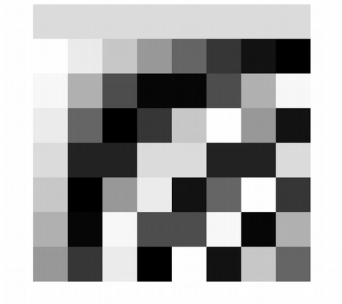
using different interpolation method.



(d)
Now the best choice is ‘linear’ interpolation and ‘Hann’ filter; But I don’t know why. All the reconstruction are bad since the degree being scanned is too small.

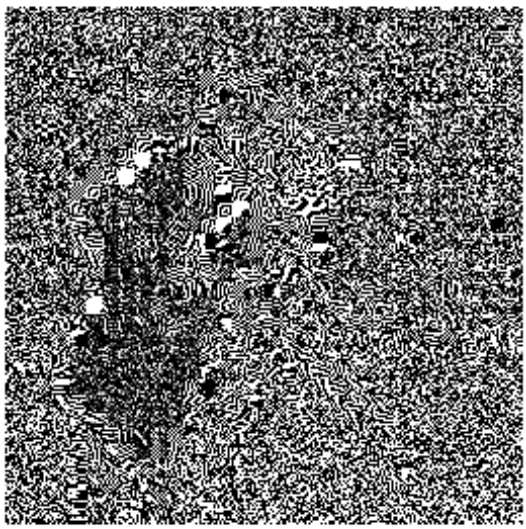
3 JPEG Compression

(a) dctmtx(n) returns the n-by-n discrete cosine transform (DCT) matrix, which you can use to perform a 2-D DCT on an image. (dct = D*A*D’)



(b) Because the DCT represent original image with linear combination of ‘cos’ patterns. Make the original image values centered zero reduce the dynamic ranged need in DCT

(c)



```
D = dctmtx(8);
I = imread('cameraman.tif');
I = double(I);
I = I-128;
dct = @(block_struct) D * block_struct.data * D';
G = blockproc(I,[8 8],dct);
```

(d) The quantization matrix will make the high frequency component has smaller values and thus spend fewer bits. This is because human eyes are less sensitive to those high frequency components.

4 JPEG Decoding

(a)

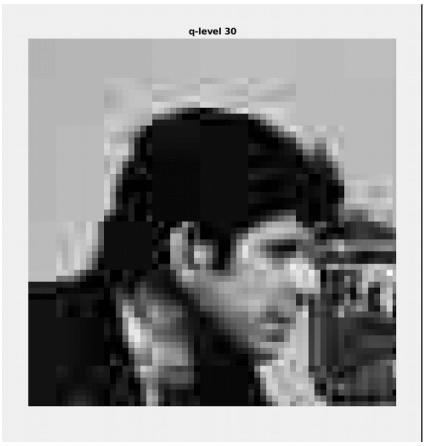
There are much more zero values after quantization. And they are mostly high frequency components.

(b) There are clear artifacts around the edges in the original images.

```
D = dctmtx(8);q_level=10;
Q = [1   1   1   2   2   2   4   4;
       1   1   2   2   2   4   4   4;
       1   2   2   2   4   4   4   8;
       2   2   2   4   4   4   8   8;
       2   2   4   4   4   8   8   8;
```

```
        2   4   4   4   8   8   8   16;
        4   4   4   8   8   8   16 16;
         4   4   8   8   8   16 16 16];
O = imread('cameraman.tif');
I = double(O);
I = I-128;
dct = @(block_struct) round((D * block_struct.data * D')./(q_level*Q));
idct = @(block_struct) round((D' * (block_struct.data.*(q_level*Q)) * D));
G = blockproc(I,[8 8],dct);
A = blockproc(G, [8 8], idct);
A = A +128;
A = uint8(A);
figure;
subplot(1,3,1);imshow(O);title('orignal');
subplot(1,3,2);imshow(A);title('q-level 10');
subplot(1,3,3);imshow(abs(A-O), []);title('absolute diff');
```

(c) The artifacts becomes more severe when q_level increases. When the q_level becomes very high, there are clear block artifact.(The 8x8 block boundaries are very clear after reconstruction) The block artifact can be reduced by applying de-blocking filters.



5 Loseless Huffman Encoding

(a) $-\sum_{i=0}^{n-1} p_i \log_b p_i$

entropy is the average uncertainty of information source. As for compression, the entropy indicate ideally how many bits should be spent for each pixel on average.
(b) The entropy of cameraman.tif is 7.0097, means we need to spend at least 7.0097 bits/pixel
(c) Use huffman encoding, we can achieve 7.0048 bits/pixel, very close the the ideal value.

```
I = imread('cameraman.tif');
e = entropy(I);
y = sort(I(:));
p = find([true;diff(y)~=0;true]);%find the pos of first element
values = y(p(1:end-1));%find different values
instances = diff(p);%find occurance of each value
p = instances/numel(I);%frequency of each value
[dict, avglen] = huffmandict(values,p);%huffman code and average length
```

6 Predictive Encoding

(a)

It looks like edge detection result.

```
f = double(imread('cameraman.tif'));
I = imread('cameraman.tif');
[m, n] = size(f);
for j=n:-1:2
    for i = 1:m
        f(i, j) = f(i, j) - f(i, j-1);
    end
end
figure;
subplot(2,1,1);imshow(I);title('Original');
subplot(2,1,2);imshow(f, []);title('Predicted');
e = entropy(f);
```

(b) Most of the predicted image is black(the uncertainty of information reduced) and thus it need less bits per pixel to compress.

The entropy of original image is 7.0097

The entropy of predicted image is 0.9825

(c)

```
for j=2:n
    for i = 1:m
        f(i, j) = f(i, j) + f(i, j-1);
    end
end
figure;
subplot(2,1,1);imshow(I);title('Original');
subplot(2,1,2);imshow(f, []);title('Predicted');
```

Original

Predicted

I think this is not lossless since you need to do rounding when alpha is not 1;
Say that alpha is integer, I think it is lossless.