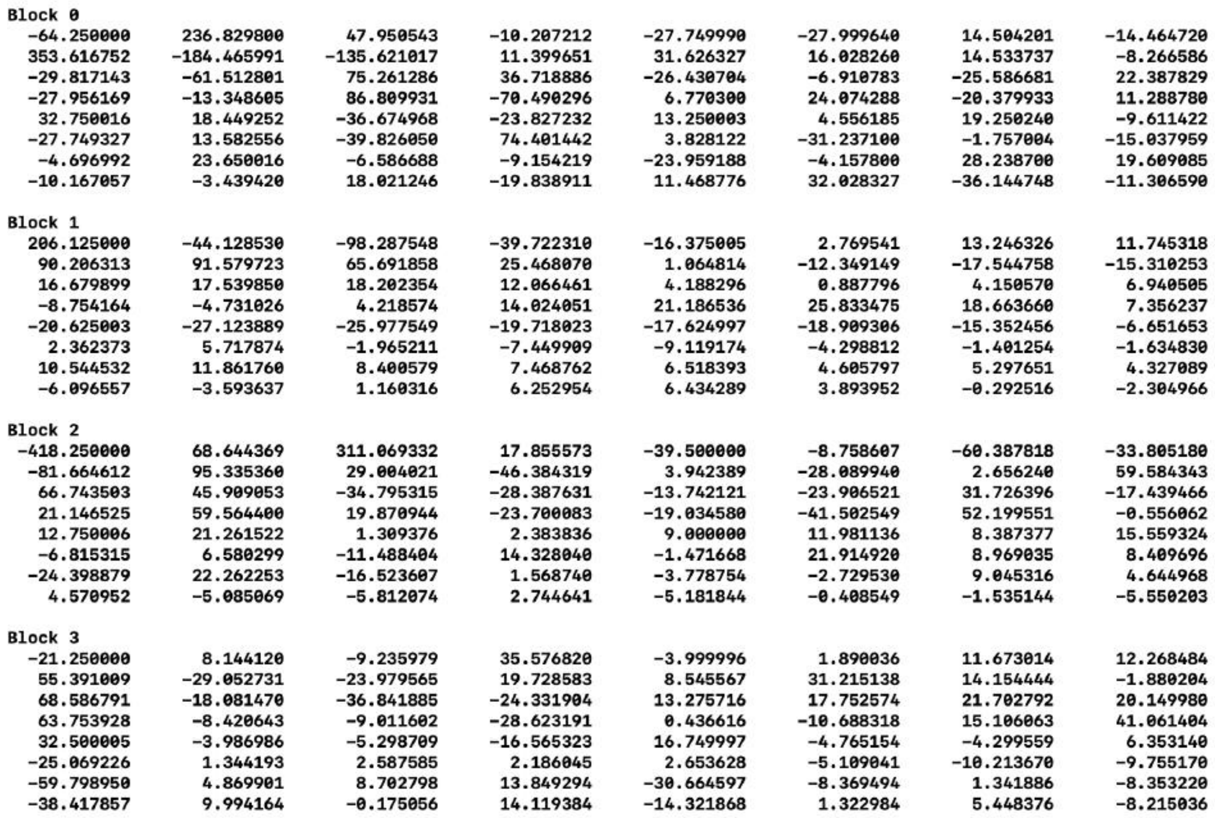
EE669 Homework #4

Yifan Wang
wang608@usc.edu

October 25, 2019

1 Discrete Cosine Transform (DCT) and Quantization for JPEG

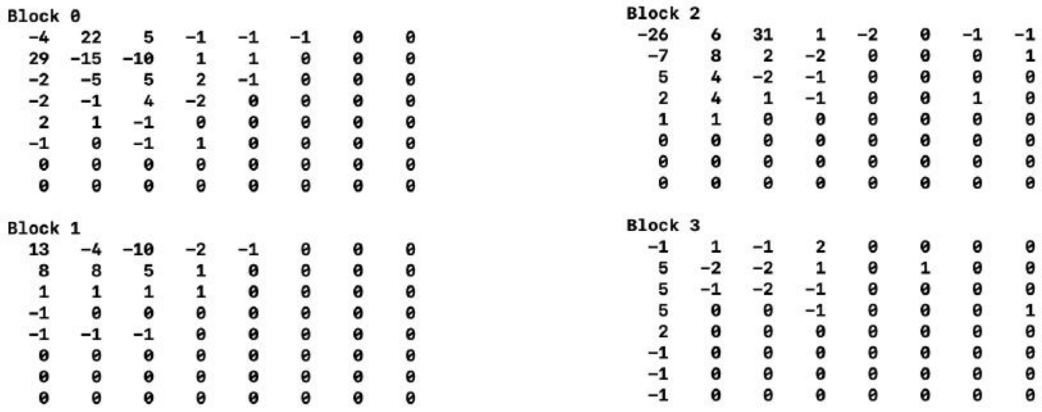
(3)



```
Block 0
  -64.250000   236.829800    47.950543   -10.207212   -27.749990   -27.999640    14.504201   -14.464720
  353.616752  -184.465991  -135.621017    11.399651    31.626327    16.028260    14.533737    -8.266586
  -29.817143   -61.512801    75.261286    36.718886   -26.430704    -6.910783   -25.586681    22.387829
  -27.956169   -13.348605    86.809931   -70.490296     6.770300    24.074288   -20.379933    11.288780
   32.750016    18.449252   -36.674968   -23.827232    13.250003     4.556185    19.250240    -9.611422
  -27.749327    13.582556   -39.826050    74.401442     3.828122   -31.237100    -1.757004   -15.037959
   -4.696992    23.650016    -6.586688    -9.154219   -23.959188    -4.157800    28.238700    19.609085
  -10.167057    -3.439420    18.021246   -19.838911    11.468776    32.028327   -36.144748   -11.306590

Block 1
  206.125000   -44.128530   -98.287548   -39.722310   -16.375005     2.769541    13.246326    11.745318
   90.206313    91.579723    65.691858    25.468070     1.064814   -12.349149   -17.544758   -15.310253
   16.679899    17.539850    18.202354    12.066461     4.188296     0.887796     4.150570     6.940505
   -8.754164    -4.731026     4.218574    14.024051    21.186536    25.833475    18.663660     7.356237
  -20.625003   -27.123889   -25.977549   -19.718023   -17.624997   -18.909306   -15.352456    -6.651653
    2.362373     5.717874    -1.965211    -7.449909    -9.119174    -4.298812    -1.401254    -1.634830
   10.544532    11.861760     8.400579     7.468762     6.518393     4.605797     5.297651     4.327089
   -6.096557    -3.593637     1.160316     6.252954     6.434289     3.893952    -0.292516    -2.304966

Block 2
 -418.250000    68.644369   311.069332    17.855573   -39.500000    -8.758607   -60.387818   -33.805180
  -81.664612    95.335360    29.004021   -46.384319     3.942389   -28.089940     2.656240    59.584343
   66.743503    45.909053   -34.795315   -28.387631   -13.742121   -23.906521    31.726396   -17.439466
   21.146525    59.564400    19.870944   -23.700083   -19.034580   -41.502549    52.199551    -0.556062
   12.750006    21.261522     1.309376     2.383836     9.000000    11.981136     8.387377    15.559324
   -6.815315     6.580299   -11.488404    14.328040    -1.471668    21.914920     8.969035     8.409696
  -24.398879    22.262253   -16.523607     1.568740    -3.778754    -2.729530     9.045316     4.644968
    4.570952    -5.085069    -5.812074     2.744641    -5.181844    -0.408549    -1.535144    -5.550203

Block 3
  -21.250000     8.144120    -9.235979    35.576820    -3.999996     1.890036    11.673014    12.268484
   55.391009   -29.052731   -23.979565    19.728583     8.545567    31.215138    14.154444    -1.880204
   68.586791   -18.081470   -36.841885   -24.331904    13.275716    17.752574    21.702792    20.149980
   63.753928    -8.420643    -9.011602   -28.623191     0.436616   -10.688318    15.106063    41.061404
   32.500005    -3.986986    -5.298709   -16.565323    16.749997    -4.765154    -4.299559     6.353140
  -25.069226     1.344193     2.587585     2.186045     2.653628    -5.109041   -10.213670    -9.755170
  -59.798950     4.869901     8.702798    13.849294   -30.664597    -8.369494     1.341886    -8.353220
  -38.417857     9.994164    -0.175056    14.119384   -14.321868     1.322984     5.448376    -8.215036
```

Figure 1: DCT result of 4 blocks

(4)



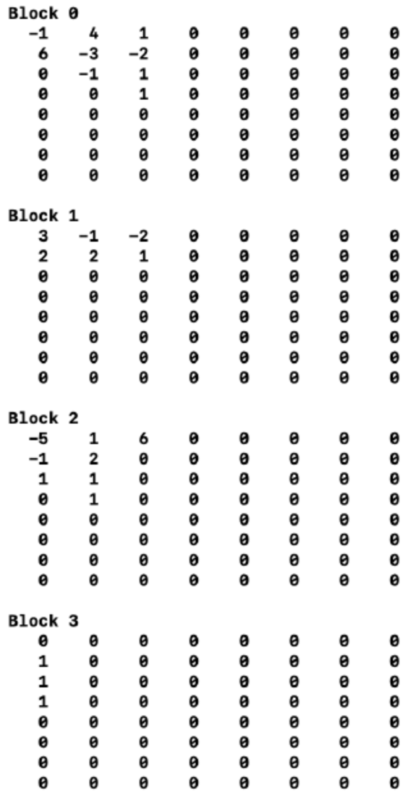
```
Block 0
  -4   22    5   -1   -1   -1    0    0
  29  -15  -10    1    1    0    0    0
  -2   -5    5    2   -1    0    0    0
  -2   -1    4   -2    0    0    0    0
   2    1   -1    0    0    0    0    0
  -1    0   -1    1    0    0    0    0
   0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0

Block 1
  13   -4  -10   -2   -1    0    0    0
   8    8    5    1    0    0    0    0
   1    1    1    1    0    0    0    0
  -1    0    0    0    0    0    0    0
  -1   -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
```

```
Block 2
 -26    6   31    1   -2    0   -1   -1
  -7    8    2   -2    0    0    0    1
   5    4   -2   -1    0    0    0    0
   2    4    1   -1    0    0    1    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

Block 3
  -1    1   -1    2    0    0    0    0
   5   -2   -2    1    0    1    0    0
   5   -1   -2   -1    0    0    0    0
   5    0    0   -1    0    0    0    1
   2    0    0    0    0    0    0    0
  -1    0    0    0    0    0    0    0
  -1    0    0    0    0    0    0    0
  -1    0    0    0    0    0    0    0
```

Figure 2: Quantized DCT result of 4 blocks use $Q = 50$

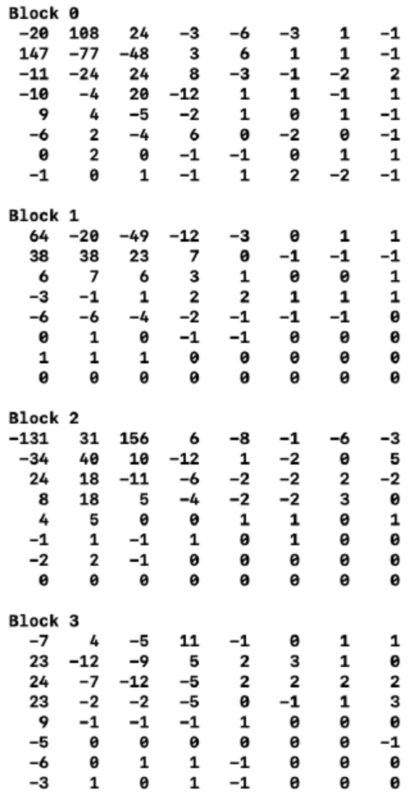
Since human eye is more sensitive to low frequency part than high frequency part, we can compress (reduce) more to high frequency part without influence much about visual experience, since some high frequency parts we are beyond human eyes' ability. In DCT DC and low frequency parts locate in left top while high frequency part locates in right bottom region. That's the reason why we can have larger quantization factor in that part. While on the other hand, we need to keep more on DC and low frequency part to give a nice image.

(5)



```
Block 0
  -1    4    1    0    0    0    0    0
   6   -3   -2    0    0    0    0    0
   0   -1    1    0    0    0    0    0
   0    0    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

Block 1
   3   -1   -2    0    0    0    0    0
   2    2    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

Block 2
  -5    1    6    0    0    0    0    0
  -1    2    0    0    0    0    0    0
   1    1    0    0    0    0    0    0
   0    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

Block 3
   0    0    0    0    0    0    0    0
   1    0    0    0    0    0    0    0
   1    0    0    0    0    0    0    0
   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
```

(a) $Q = 10$ Quantized Result



```
Block 0
 -20  108   24   -3   -6   -3    1   -1
 147  -77  -48    3    6    1    1   -1
 -11  -24   24    8   -3   -1   -2    2
 -10   -4   20  -12    1    1   -1    1
   9    4   -5   -2    1    0    1   -1
  -6    2   -4    6    0   -2    0   -1
   0    2    0   -1   -1    0    1    1
  -1    0    1   -1    1    2   -2   -1

Block 1
  64  -20  -49  -12   -3    0    1    1
  38   38   23    7    0   -1   -1   -1
   6    7    6    3    1    0    0    1
  -3   -1    1    2    2    1    1    1
  -6   -6   -4   -2   -1   -1   -1    0
   0    1    0   -1   -1    0    0    0
   1    1    1    0    0    0    0    0
   0    0    0    0    0    0    0    0

Block 2
-131   31  156    6   -8   -1   -6   -3
 -34   40   10  -12    1   -2    0    5
  24   18  -11   -6   -2   -2    2   -2
   8   18    5   -4   -2   -2    3    0
   4    5    0    0    1    1    0    1
  -1    1   -1    1    0    1    0    0
  -2    2   -1    0    0    0    0    0
   0    0    0    0    0    0    0    0

Block 3
  -7    4   -5   11   -1    0    1    1
  23  -12   -9    5    2    3    1    0
  24   -7  -12   -5    2    2    2    2
  23   -2   -2   -5    0   -1    1    3
   9   -1   -1   -1    1    0    0    0
  -5    0    0    0    0    0    0   -1
  -6    0    1    1   -1    0    0    0
  -3    1    0    1   -1    0    0    0
```

(b) $Q = 90$ Quantized Result

```
  80.000000    55.000000    50.000000    80.000000   120.000000   200.000000   255.000000   305.000000
  60.000000    60.000000    70.000000    95.000000   130.000000   290.000000   300.000000   275.000000
  70.000000    65.000000    80.000000   120.000000   200.000000   285.000000   345.000000   280.000000
  70.000000    85.000000   110.000000   145.000000   255.000000   435.000000   400.000000   310.000000
  90.000000   110.000000   185.000000   280.000000   340.000000   545.000000   515.000000   385.000000
 120.000000   175.000000   275.000000   320.000000   405.000000   520.000000   565.000000   460.000000
 245.000000   320.000000   390.000000   435.000000   515.000000   605.000000   600.000000   505.000000
 360.000000   460.000000   475.000000   490.000000   560.000000   500.000000   515.000000   495.000000
```

(c) $Q = 10$

```
   3.200000     2.200000     2.000000     3.200000     4.800000     8.000000    10.200000    12.200000
   2.400000     2.400000     2.800000     3.800000     5.200000    11.600000    12.000000    11.000000
   2.800000     2.600000     3.200000     4.800000     8.000000    11.400000    13.800000    11.200000
   2.800000     3.400000     4.400000     5.800000    10.200000    17.400000    16.000000    12.400000
   3.600000     4.400000     7.400000    11.200000    13.600000    21.800000    20.600000    15.400000
   4.800000     7.000000    11.000000    12.800000    16.200000    20.800000    22.600000    18.400000
   9.800000    12.800000    15.600000    17.400000    20.600000    24.200000    24.000000    20.200000
  14.400000    18.400000    19.000000    19.600000    22.400000    20.000000    20.600000    19.800000
```

(d) $Q = 90$

Figure 2: Different quantization matrices

Comparing different quantized matrices, when N is larger, more information is retained (especially high frequency part, which means less successive 0 in high frequency part, better quality and larger file size). In that case, a relative low N would be prefect for entropy coding (in our case $N = 10$ would be best, where we have more successive 0 than other 2 cases since lots of high frequency information is ignored, and would result a higher compression ratio for entropy coding. However, due to the fact that lots of information is ignore, image quality would be poorest. It can be observed from above result ($Figure2(a, b)$), $N = 10$ would have more zeros when sweep through diagonal.

2 JPEG Compression Quality Factor

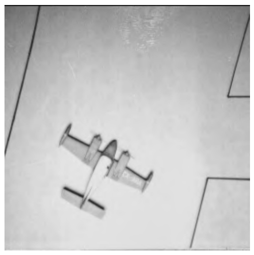
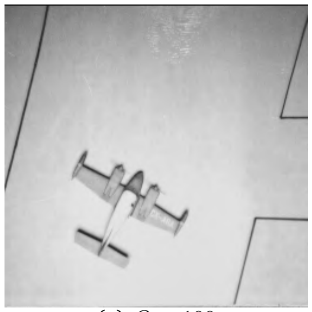


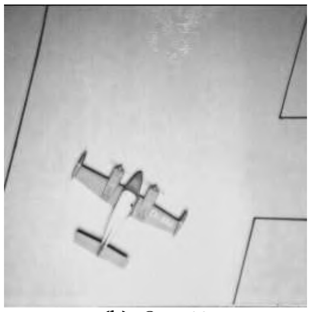
Figure 3: Raw Image

Quality Factor	100	60	40	20	10	1
JPEG size (byte)	33420	5233	4145	3171	2570	1803
MSE	0.093	8.690	13.058	25.084	45.612	884.793
PSNR (dB)	58.436	38.740	36.971	34.136	31.539	18.662
Compression Ratio	0.5026	0.0785	0.0622	0.0476	0.0385	0.0270

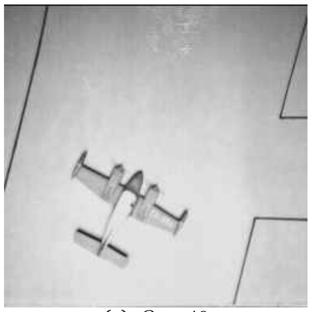
Table 1: Result of JPEG with different quality factor
$airplane.bmp$ size: 66614 bytes



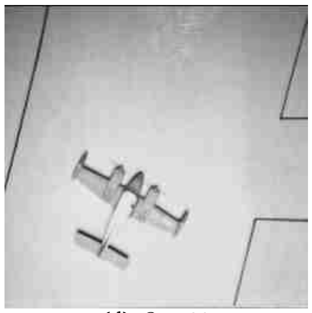
(a) $Q = 100$

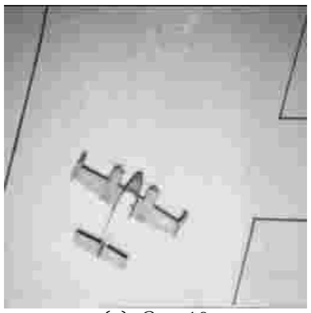


(b) $Q = 60$



(c) $Q = 40$


(d) $Q = 20$


(e) $Q = 10$

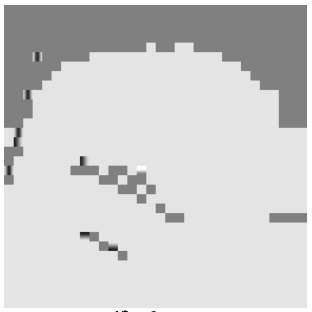
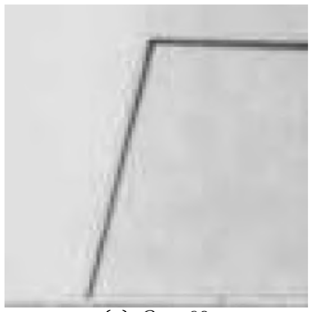
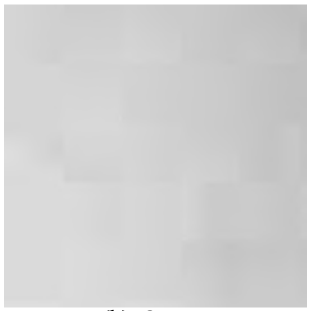

(f) $Q = 1$

Figure 3: Result of JPEG with different quality factor


(a) $Q = 60$


(b) $Q = 40$

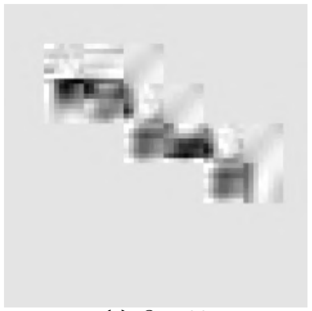

(c) $Q = 20$

Figure 4: Artifacts in JPEG

Quality factor would affect image quality by having different quantize level, a large quality factor would have a small quantize gap which would retain more detail, especially in high frequency part. It would keep more information and has less zero.

When we decreasing quality factor, image quality is decreasing. When $N = 100$, jpeg image seem almost the same as raw image. From MSE we can found there are little difference which human eye can not observe. While $N = 60$, image seems nice at first glance, especially background regions there seems nothing different. From $Figure4(a)$ it's the right bottom corner of image, it can be found black line is not so prefect as original one. It is blurred, for the reason that some high frequency part is ignored. Edges are generally the high frequency part, so that it is blurred. When N goes down to 40, the background is no longer smooth any more, it is blocked and gave a mosaic feeling as $Figure4(b)$, while edge becomes more blurred since more high frequency part is removed. When N became 1, only small part of low frequency is retained so that result is super bad, $Figure4(c)$ (plane tail).

3 Post-Processing of JPEG Encoded Images

3.1 Filtering

Since the blocking effect would always occur at boundary of 8×8 block, in my case, I separate the boundary pixels into 3 different situations. As $Figure6$ shows a boundary case, blue line is horizontal boundary while red line is vertical boundary. Let H represents the set of pixels in blue dotted region, let V represents the set of pixels in red dotted region. Three situations are:

Case 1. Pixels $\in H \cap V$.

Case 2. Pixels $\in H - V$

Case 3. Pixels $\in V - H$

(a) $boat1.jpg$

(b) $boat2.jpg$

(c) $boat3.jpg$

(d) $boat4.jpg$

(e) $boat5.jpg$

(f) $jellybeans1.jpg$

(g) $jellybeans2.jpg$

(h) $jellybeans3.jpg$

(i) $jellybeans4.jpg$

(j) $jellybeans5.jpg$

Figure 5: Raw Images

Then define similar pixels in a local region, let $P(k,i,j)$ represents value at image k^{th} channel in location (i,j), if

$$P(k,i,j) - P(k,i',j') < THRESHOLD$$

call these 2 pixels are similar.

For [Case 1.] Compute mean in a square window (5×5) only consider pixels similar with center pixel.

For [Case 2.] Compute mean in a rectangular window (3×5) only consider pixels similar with center pixel.

For [Case 3.] Compute mean in a rectangular window (5×3) only consider pixels similar with center pixel.

Use the computed mean to replace pixel value at center pixel as processed value. Result is shown in $Figure 7$ and PSNR and SSIM is given in $Table 2$ (THRESHOLD=10).

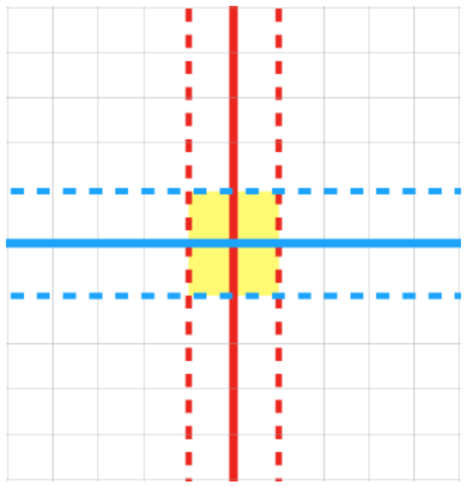


Figure 6: Explain of Filtering

Image	Before		After		Image	Before		After	
	PSNR	SSIM	PSNR	SSIM		PSNR	SSIM	PSNR	SSIM
$boat1$	28.131	0.034478	28.2139	0.034553	$jellybeans1$	26.385	0.001889	26.4314	0.001890
$boat2$	30.493	0.034560	30.5944	0.034630	$jellybeans2$	29.003	0.001897	29.0961	0.001899
$boar3$	32.753	0.034579	32.8322	0.034641	$jellybeans3$	30.831	0.001896	30.9362	0.001898
$boat4$	34.203	0.034560	34.2328	0.034611	$jellybeans4$	31.988	0.001900	32.0957	0.001902
$boat5$	36.431	0.034530	36.3155	0.034586	$jellybeans5$	33.724	0.001899	33.8413	0.001901

Table 2: Deblocking result using filtering

(a) $boat1.jpg$

(b) $boat2.jpg$

(c) $boat3.jpg$

(d) $boat4.jpg$

(e) $boat5.jpg$

(f) $jellybeans1.jpg$

(g) $jellybeans2.jpg$

(h) $jellybeans3.jpg$

(i) $jellybeans4.jpg$

(j) $jellybeans5.jpg$

Figure 7: Deblocking result using filtering

It can be found that using filtering method can remove some blocking effect and enhance PSNR slightly. By using similarity, we can also retain edge part and make image visually better, otherwise edge would be blurred during this low pass filtering. But blocking effects remains a problem for image with poor quality, even if we can increase the similarity threshold to improve the performance, truncated high frequency part can not be recovered. If zoom into a particular region, blocking remains can be observed which is a problem that we only consider a small local region while doing average process, a more careful average in a large region would definitely help.

3.2 Reapplying JPEG

Image	Before		After		Image	Before		After	
	PSNR	SSIM	PSNR	SSIM		PSNR	SSIM	PSNR	SSIM
$boat1$	28.131	0.034478	28.1299	0.034476	$jellybeans1$	26.385	0.001889	26.6722	0.0018885
$boat2$	30.493	0.034560	30.4900	0.034554	$jellybeans2$	29.003	0.001897	29.0800	0.0018973
$boat3$	32.753	0.034579	32.7388	0.034571	$jellybeans3$	30.831	0.001896	30.6642	0.0018958
$boat4$	34.203	0.034560	34.1832	0.034548	$jellybeans4$	31.988	0.001900	31.5461	0.0018995
$boat5$	36.431	0.034530	36.3536	0.034513	$jellybeans5$	33.724	0.001899	32.6353	0.0018987

Table 3: Deblocking result using filtering

(a) $boat1.jpg$

(b) $boat2.jpg$

(c) $boat3.jpg$

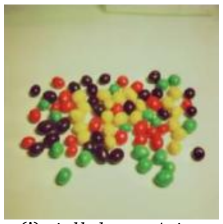
(d) $boat4.jpg$

(e) $boat5.jpg$

(f) $jellybeans1.jpg$

(g) $jellybeans2.jpg$

(h) $jellybeans3.jpg$


(i) $jellybeans4.jpg$

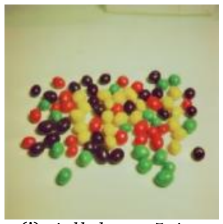

(j) $jellybeans5.jpg$

Figure 8: Deblocking result using reapply jpeg

This method is relative easy to understand, just shift jpeg image and do jpeg compression again on all possible shift combination, according to raw paper, it would perform best when using same quality factor doing jpeg on shifted image. Then shift image back and do averaging on all images. Doing average would help to reduce blocking factor since the region of blocking in different image would be different doing average would reduce it which is the same idea as de-noising using stack of image. After averaging, blocking would be reduce to $\frac{1}{64}$ compare original 1 jpeg image. However, it need to compute JPEG compression for many times and do average among these images which would need more memory and time to finish one single process. PSNR would almost the same, especially on low quality image would have some increase. But visualization experience is much better than that using filtering and raw image, especially on low quality image, blocking effect is not so obvious any more.