EE669 Homework #4

Yifan Wang
wang608@usc.edu

October 21, 2019

1 Discrete Cosine Transform (DCT) and Quantization for JPEG

(3)

Block 0

-64.250000	244.165499	25.841959	3.323738	-46.181585	-4.976727	-2.500331	-1.525660
361.278369	-220.415763	-107.792071	-0.053886	51.507229	-5.819149	34.981667	-35.960061
-64.357883	-33.661204	98.034182	20.492855	-26.448634	-12.448195	-22.239056	28.763469
7.170990	-28.028663	64.443158	-87.876486	36.462842	7.618478	-1.731139	-0.492352
2.498080	39.669789	-38.151511	4.093715	1.422310	2.193837	11.981416	-12.685386
-4.504526	-8.958151	-41.587830	74.463012	-12.063947	-17.560708	-4.576224	-5.736994
-27.272458	47.720611	4.998736	-20.182782	-29.064225	10.870818	21.720316	20.730576
15.416400	-35.301256	9.359944	-16.273519	35.210936	19.518091	-47.614346	-10.754252

Block 1

206.125000	-55.705972	-88.343746	-43.222419	-6.524141	-2.683619	22.428699	0.511047
82.830514	94.256566	69.260646	23.512335	0.221125	-15.262493	-17.771424	-14.635903
18.561527	8.002369	9.976397	8.527346	4.856986	3.518081	8.932724	7.535950
-15.200229	0.429052	9.442997	15.983935	20.739447	23.408747	12.357912	3.681617
-14.828468	-33.841458	-34.175448	-24.799171	-22.064029	-21.657818	-13.922186	-4.190697
1.536759	19.413757	10.881847	0.714935	-3.082186	2.017056	0.346525	-0.789033
12.761283	-1.399242	-2.812207	3.082867	4.570156	1.929043	5.208829	3.924346
-12.852999	6.403608	11.362210	10.868825	7.698792	5.578127	-2.719743	-3.670927

Block 2

-418.250000	94.459291	292.051554	4.449380	-38.893138	-7.365871	-71.726281	-12.856723
-63.169729	99.223703	1.653515	-39.403815	1.130761	-26.832572	16.706972	62.900651
56.449560	42.228604	-25.526750	-22.369202	-20.702961	-16.148749	30.598007	-34.908755
28.186095	60.753559	6.579325	-22.703356	-20.051379	-30.934701	59.830463	-8.975933
1.807303	13.560245	7.162595	9.087432	9.086095	19.360134	-1.025739	12.248300
-0.959476	12.032977	-22.939285	12.424966	-4.541825	20.996224	9.932410	17.603619
-33.043078	18.571615	-8.062513	6.961837	-9.016977	3.155002	-2.524717	0.002725
21.701299	-7.826871	-12.975553	-0.461671	-3.777678	-1.029789	3.091169	-0.013462

Block 3

-21.250000	8.701888	-8.992595	38.007806	-11.261500	9.796365	6.677989	16.863815
58.255085	-33.862076	-18.852332	16.007055	12.633356	31.201567	10.238386	-3.890264
66.551232	-19.597744	-33.732932	-25.458332	18.268047	10.932941	28.367751	14.817606
64.506827	-12.210158	-4.773963	-30.366847	6.459506	-16.806291	27.754784	28.981007
23.650384	-2.838309	-6.035141	-13.579403	20.817470	-10.408171	-0.318921	-1.345970
-31.258581	1.816403	4.459854	5.666473	-6.383455	-4.477465	-11.763501	-6.448774
-62.284923	10.394253	4.563334	15.028250	-36.949655	3.114832	-4.904059	-1.982919
-26.771578	8.883228	-1.393274	13.568464	-12.511919	8.668734	-1.811813	-3.661490

Figure 1: DCT result of 4 blocks

(4)

Block 0

-4	22	3	0	-2	0	0	0
30	-18	-8	0	2	0	1	-1
-5	-3	6	1	-1	0	0	1
1	-2	3	-3	1	0	0	0
0	2	-1	0	0	0	0	0
0	0	-1	1	0	0	0	0
-1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Block 1

13	-5	-9	-3	0	0	0	0
7	8	5	1	0	0	0	0
1	1	1	0	0	0	0	0
-1	0	0	1	0	0	0	0
-1	-2	-1	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

Block 2

-26	9	29	0	-2	0	-1	0
-5	8	0	-2	0	0	0	1
4	3	-2	-1	-1	0	0	-1
2	4	0	-1	0	0	1	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-1	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	-3	-1	1	0	1	0	0
5	-2	-2	-1	0	0	0	0
5	-1	0	-1	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

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	-4	-2	0	0	0	0	0
-1	-1	1	0	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

Block 1

3	-1	-2	-1	0	0	0	0
1	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	2	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	-1	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	111	13	1	-10	-1	0	0
151	-92	-38	0	10	-1	3	-3
-23	-13	31	4	-3	-1	-2	3
3	-8	15	-15	4	0	0	0
1	9	-5	0	0	0	1	-1
-1	-1	-4	6	-1	-1	0	0
-3	4	0	-1	-1	0	1	1
1	-2	0	-1	2	1	-2	-1

Block 1

64	-25	-44	-14	-1	0	2	0
35	39	25	6	0	-1	-1	-1
7	3	3	2	1	0	1	1
-5	0	2	3	2	1	1	0
-4	-8	-5	-2	-2	-1	-1	0
0	3	1	0	0	0	0	0
1	0	0	0	0	0	0	0
-1	0	1	1	0	0	0	0

Block 2

-131	43	146	1	-8	-1	-7	-1
-26	41	1	-10	0	-2	1	6
20	16	-8	-5	-3	-1	2	-3
10	18	1	-4	-2	-2	4	-1
1	3	1	1	1	1	0	1
0	2	-2	1	0	1	0	1
-3	1	-1	0	0	0	0	0
2	0	-1	0	0	0	0	0

Block 3

-7	4	-4	12	-2	1	1	1
24	-14	-7	4	2	3	1	0
24	-8	-11	-5	2	1	2	1
23	-4	-1	-5	1	-1	2	2
7	-1	-1	-1	2	0	0	0
-7	0	0	0	0	0	-1	0
-6	1	0	1	-2	0	0	0
-2	0	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

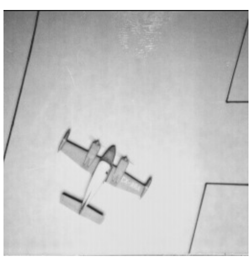
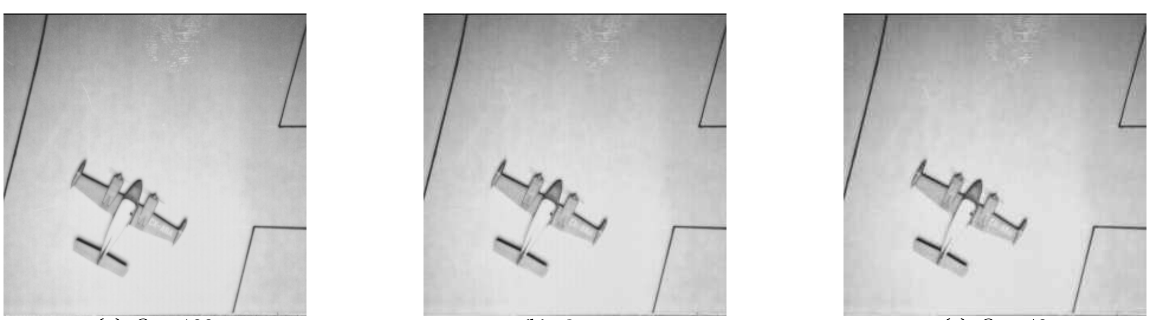


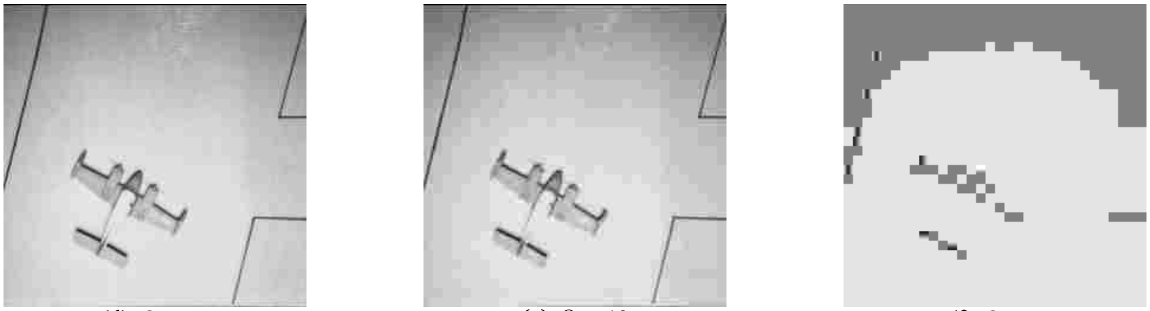
Figure 3: Raw Image

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

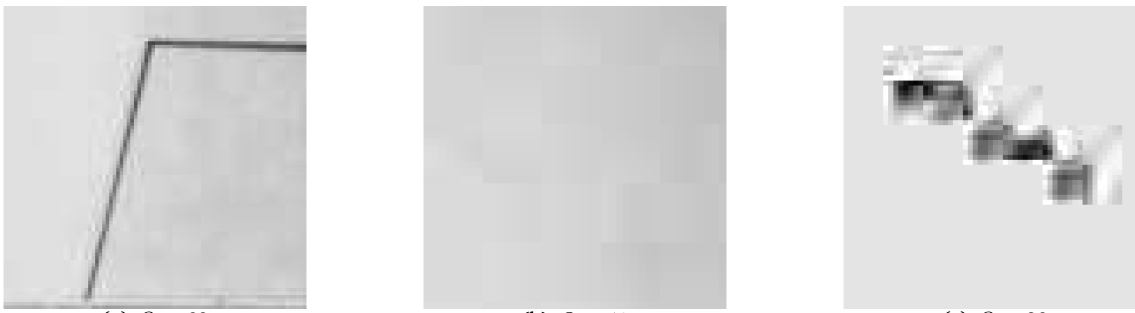


(a) $Q = 100$ (b) $Q = 60$ (c) $Q = 40$



(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

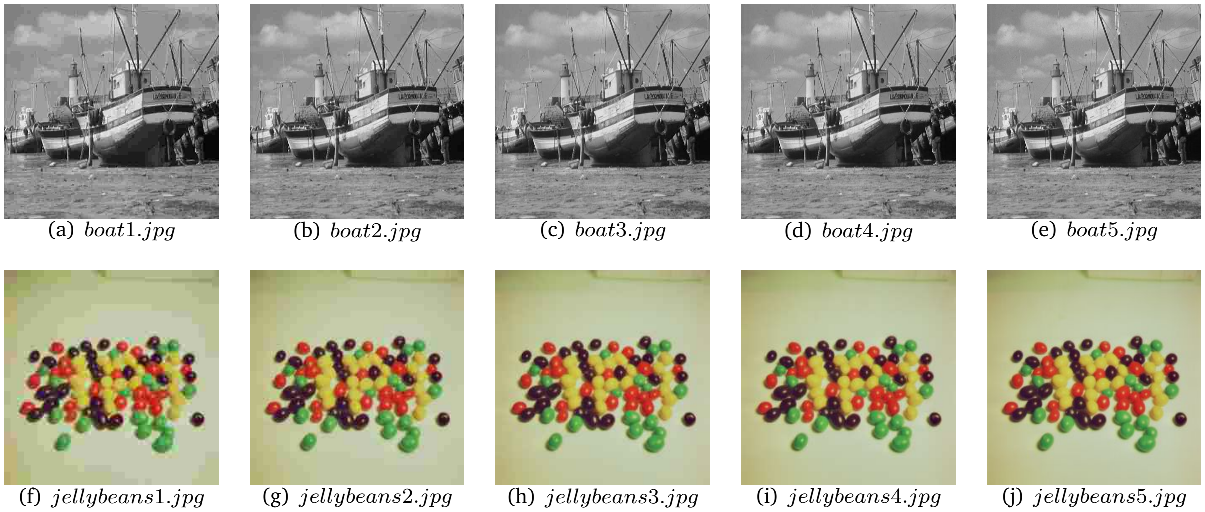
3.1 Filtering

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).

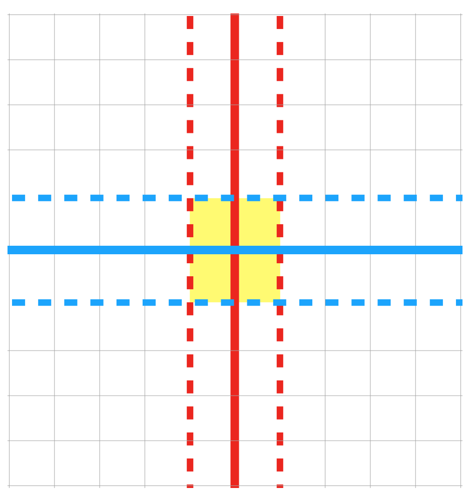


Figure 6: Explain of Filtering

Image	Before		After		Image	Before		After	
	PSNR	SSIM	PSNR	SSIM		PSNR	SSIM	PSNR	SSIM
*boat*1	28.131	0.034478	28.2139	0.034553	*jellybeans*1	26.385	0.001889	26.4314	0.001890
*boat*2	30.493	0.034560	30.5944	0.034630	*jellybeans*2	29.003	0.001897	29.0961	0.001899
*boar*3	32.753	0.034579	32.8322	0.034641	*jellybeans*3	30.831	0.001896	30.9362	0.001898
*boat*4	34.203	0.034560	34.2328	0.034611	*jellybeans*4	31.988	0.001900	32.0957	0.001902
*boat*5	36.431	0.034530	36.3155	0.034586	*jellybeans*5	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*

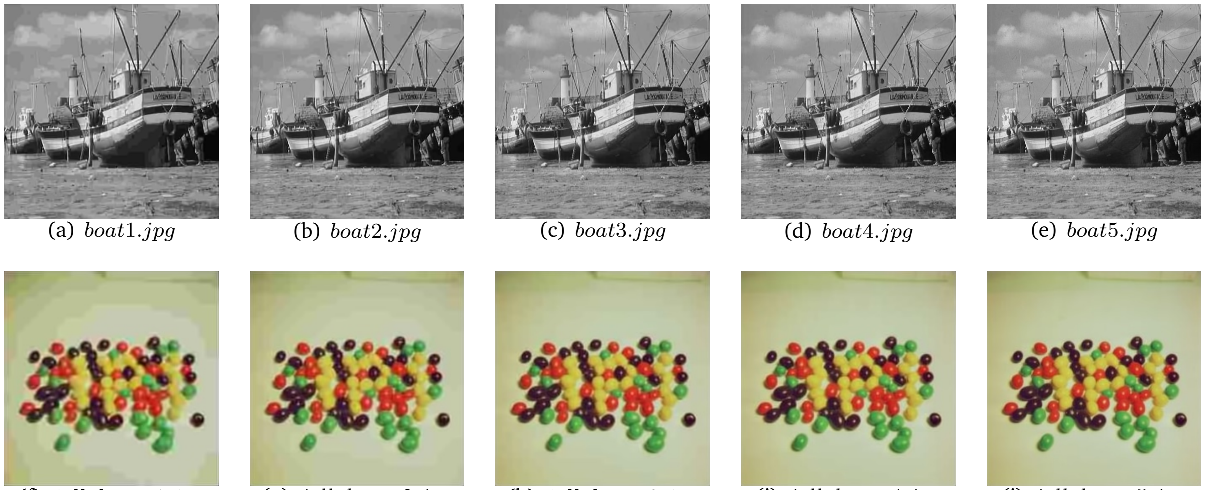
Figure 7: Deblocking result using filtering

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
*boat*1	28.131	0.034478	28.1299	0.034476	*jellybeans*1	26.385	0.001889	26.6722	0.0018885
*boat*2	30.493	0.034560	30.4900	0.034554	*jellybeans*2	29.003	0.001897	29.0800	0.0018973
*boat*3	32.753	0.034579	32.7388	0.034571	*jellybeans*3	30.831	0.001896	30.6642	0.0018958
*boat*4	34.203	0.034560	34.1832	0.034548	*jellybeans*4	31.988	0.001900	31.5461	0.0018995
*boat*5	36.431	0.034530	36.3536	0.034513	*jellybeans*5	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 164 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.