Image Compression

Results and Experiments

Participants:

- Pratik Tiwari (2019201023)
- Jay Krishna (2019201019)
- Tirth Pandit (2019201017)

Brief Description

- Purpose : To gain a very low bits per Pixel and achieve a high visual quality of de-compressed images.
- Goal : To save storage space and to reduce transmission time.
- Aim : To achieving a high compression ratio (CR) while preserving good visual fidelity of decoded images.

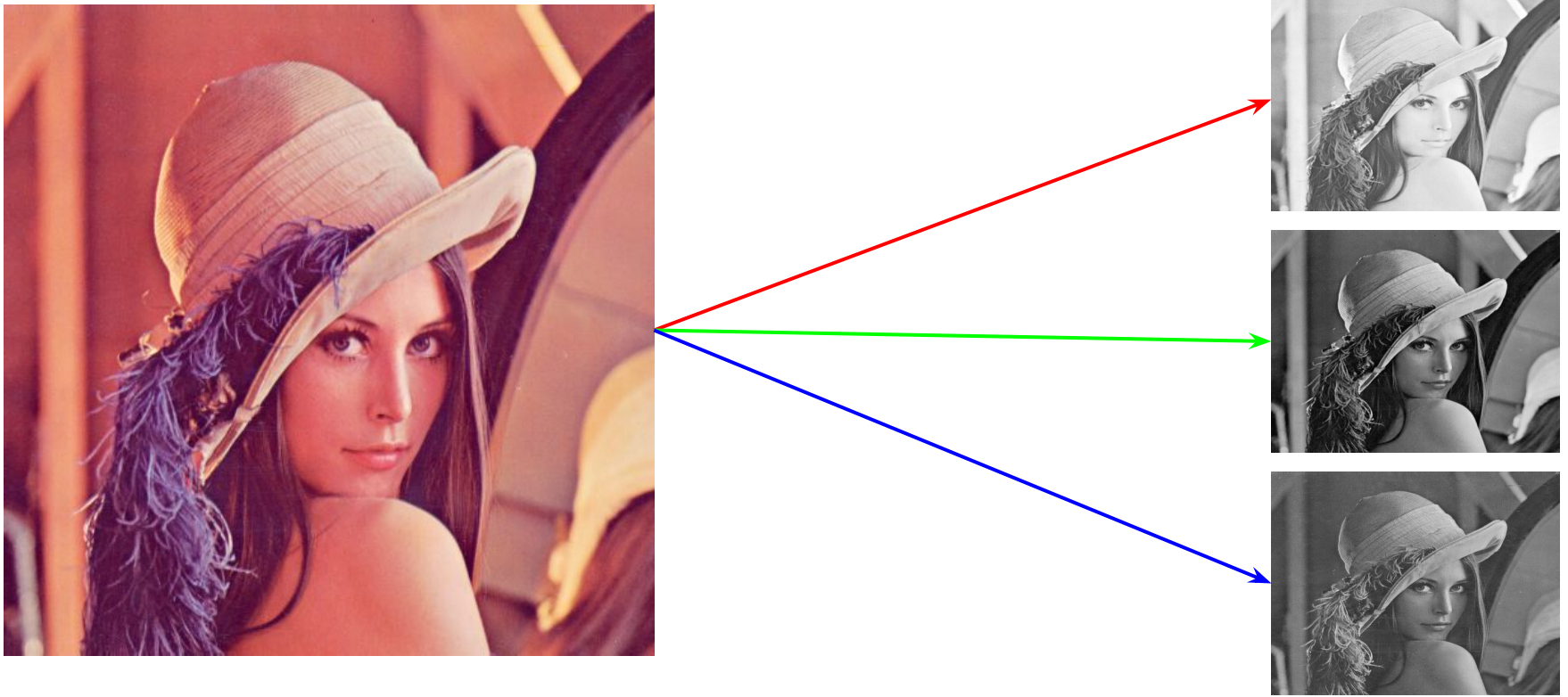
Lossless Compression Methods [Original Image can be rebuilt]	**Lossy Compression Methods** [High Compression ratio]
Huffman Coding	Chroma Subsampling
Run-length Encoding	JPEG Standard Compression
	Quad-tree based compression

Taking advantage of Human Eye

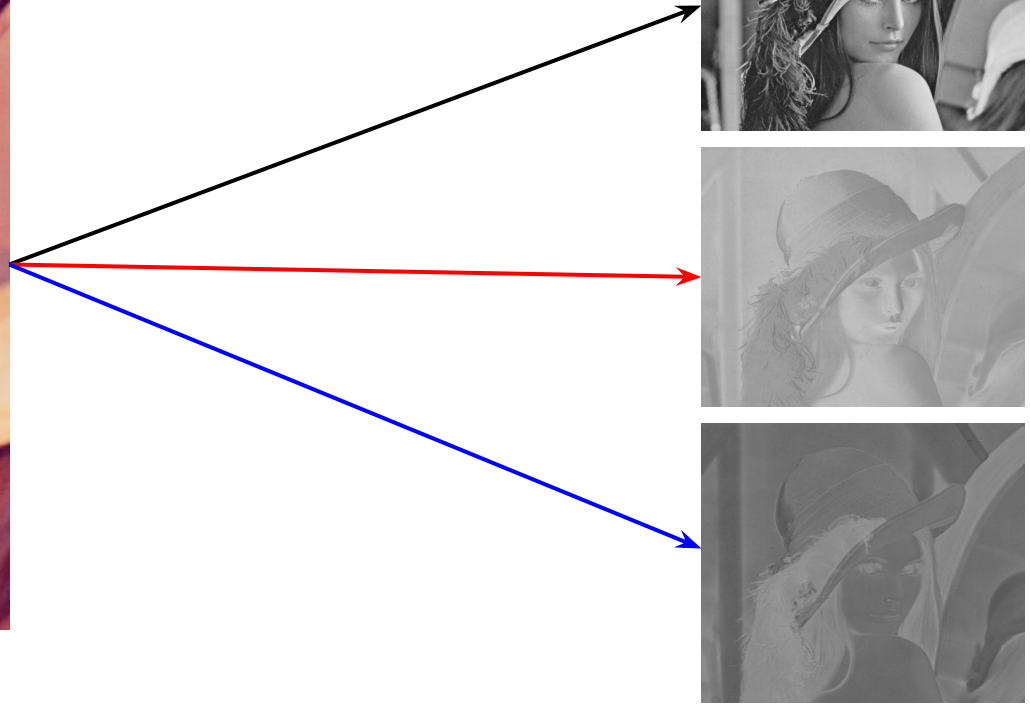
Human eye is more sensitive towards luminance then chrominance !

Average Human Eye contains ~20 times more rods than cones.

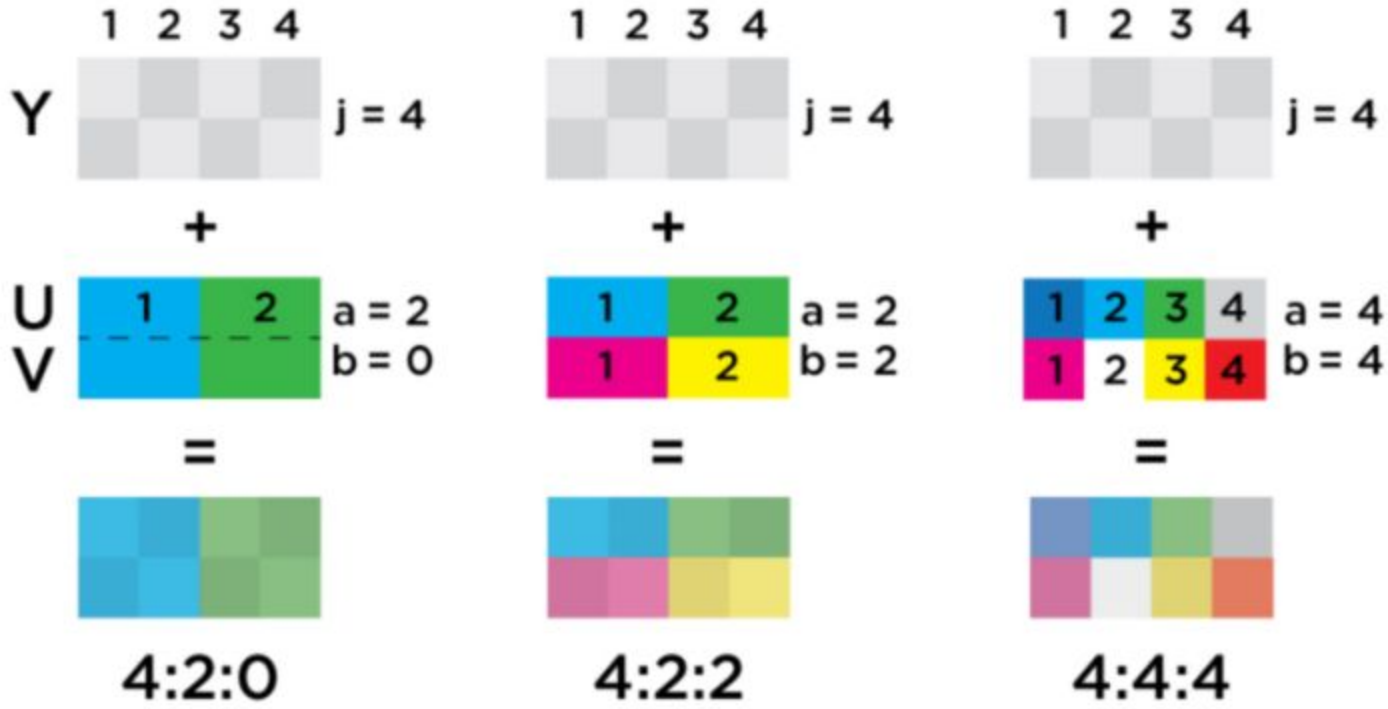
Color Components in Image



Y Cr Cb



[1] Mark S. Abeln(29th June 2010) Luminance is More Important than Color
(http://therefractedlight.blogspot.com/2010/06/luminance-is-more-important-than-color.html)

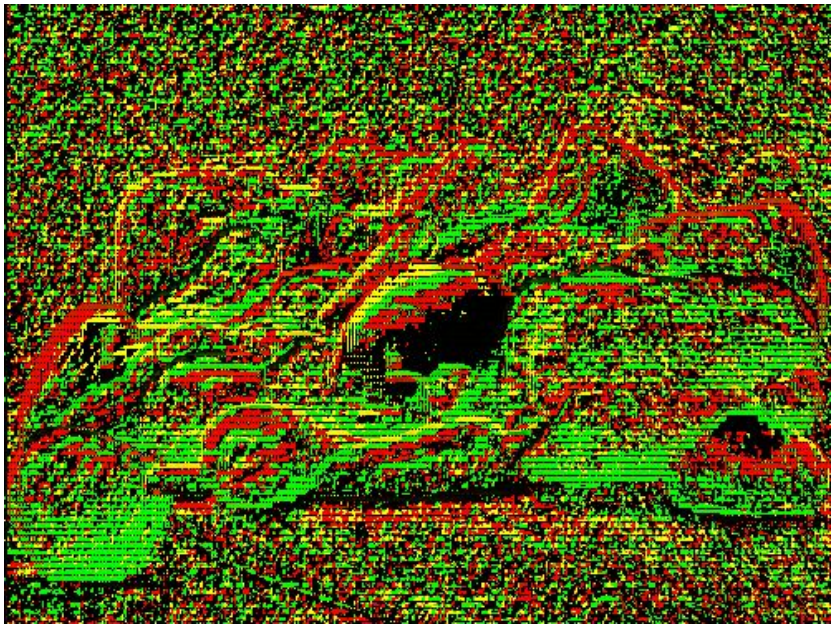

1 2 3 4
Y
j = 4
+
U
V
1
2
a = 2
b = 0
=
4:2:0
1 2 3 4
j = 4
+
1
2
1
2
a = 2
b = 2
=
4:2:2
1 2 3 4
j = 4
+
1 2 3 4
1 2 3 4
a = 4
b = 4
=
4:4:4

Chroma Subsampling

Sampling rates

- (4,1,0)
- (4,1,1)
- (4,2,0)
- (4,2,2)
- (4,4,0)
- (4,4,4)

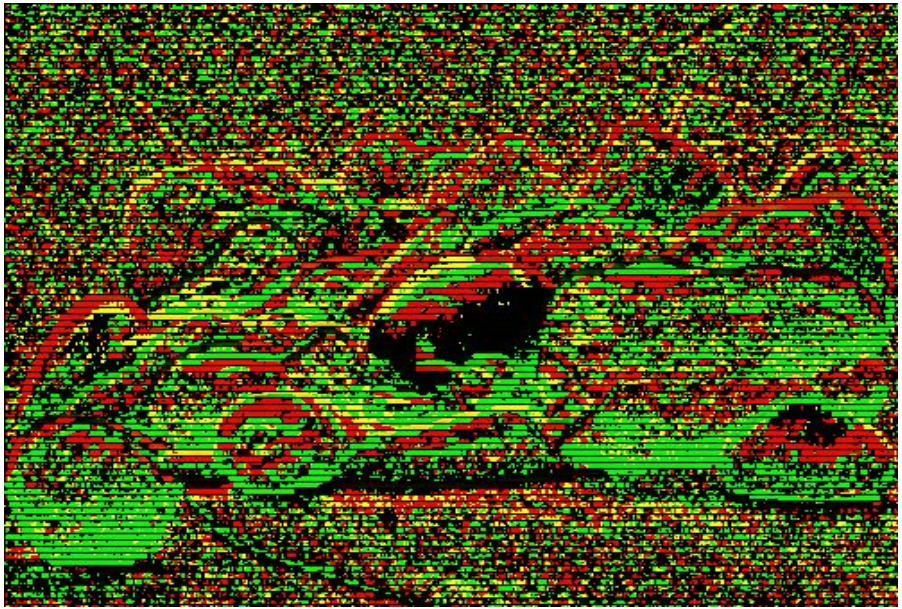
Sampling Rate: (4,1,0)



Compression Ratio	2.3964
RMSE	4.7752
PSNR(db)	34.5509

Compression Ratio = Original Size / Compressed Size

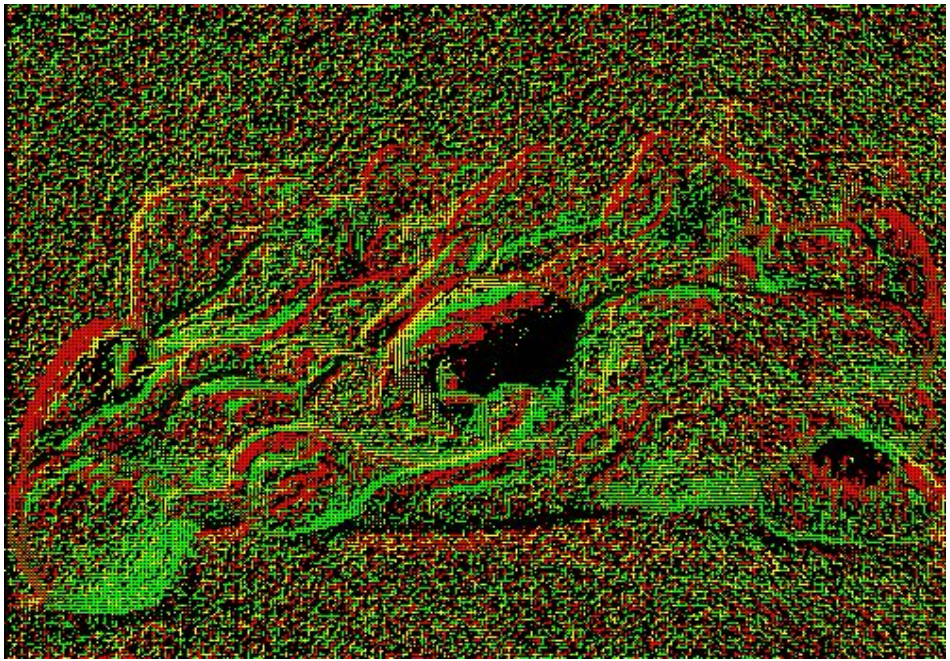
Sampling Rate: (4,1,1)



Compression Ratio	1.9964
RMSE	4.5008
PSNR(db)	35.0649

Compression Ratio = Original Size / Compressed Size

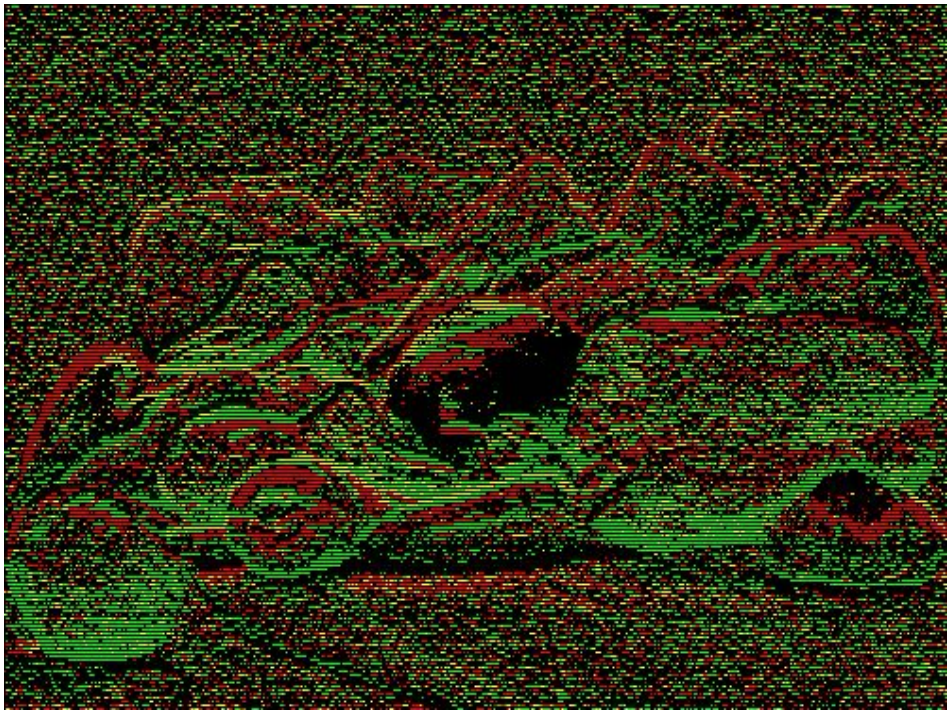
Sampling Rate: (4,2,0)



Compression Ratio	1.9986
RMSE	2.6064
PSNR(db)	39.8097

Compression Ratio = Original Size / Compressed Size

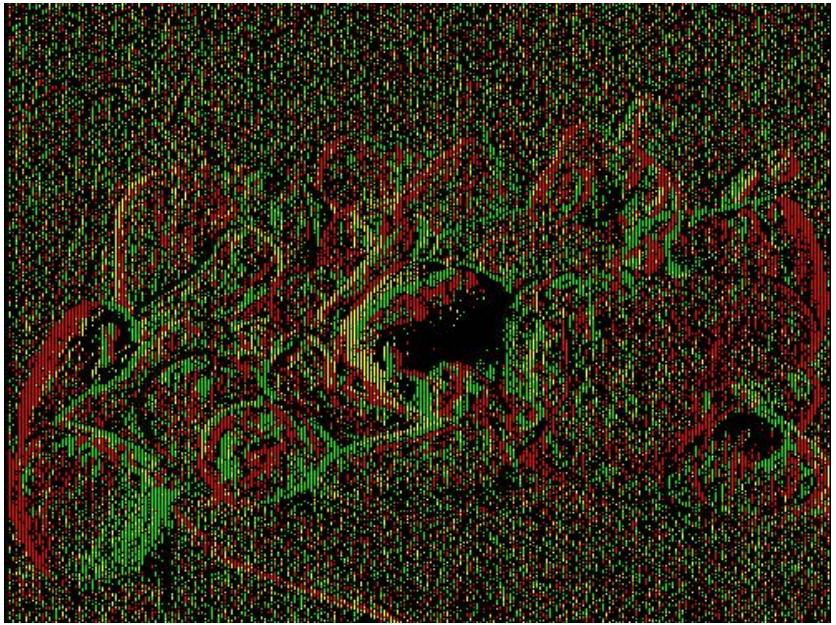
Sampling Rate: (4,2,2)



Compression Ratio	1.5
RMSE	2.0104
PSNR(db)	42.0650

Compression Ratio = Original Size / Compressed Size

Sampling Rate: (4,4,0)



Compression Ratio	1.4984
RMSE	1.6645
PSNR(db)	43.7046

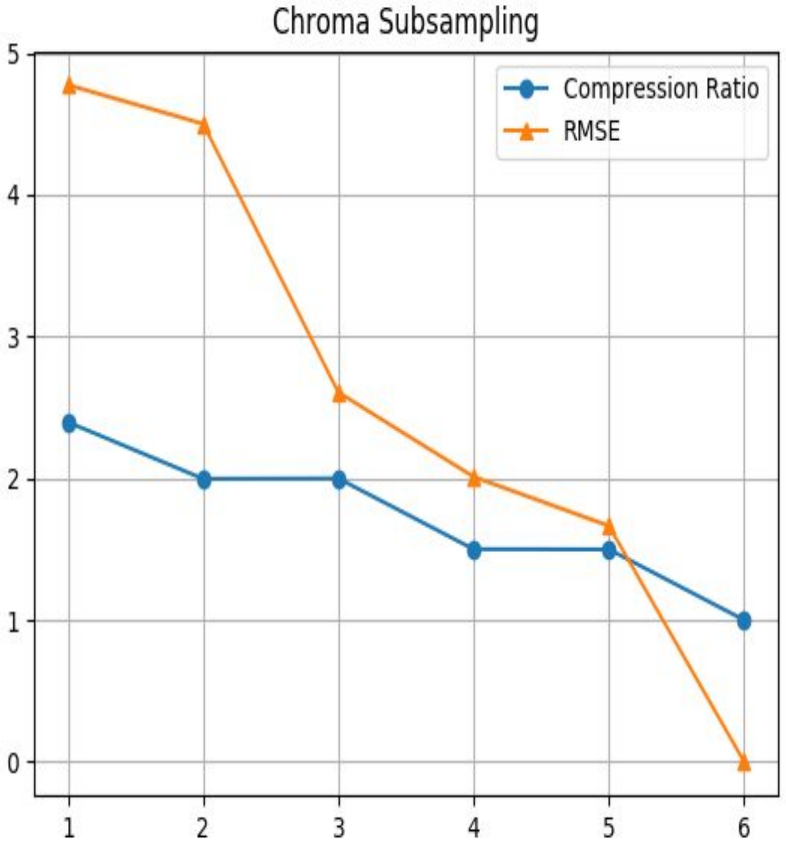
Compression Ratio = Original Size / Compressed Size

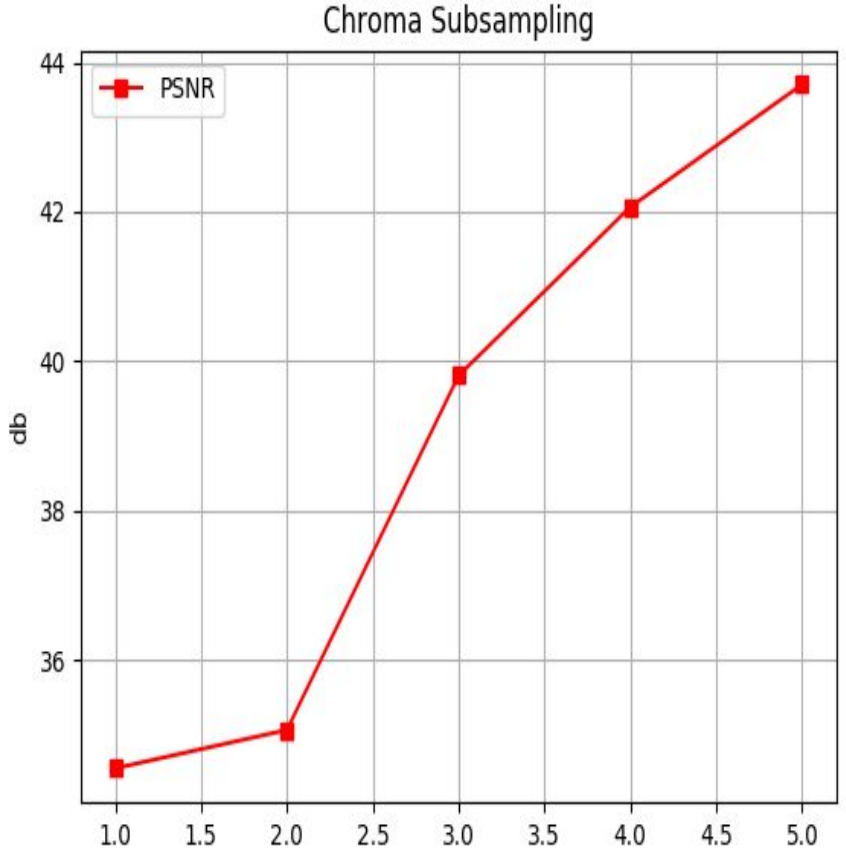
Sampling Rate: (4,4,4)

Compression Ratio	1
RMSE	0
PSNR(db)	Inf

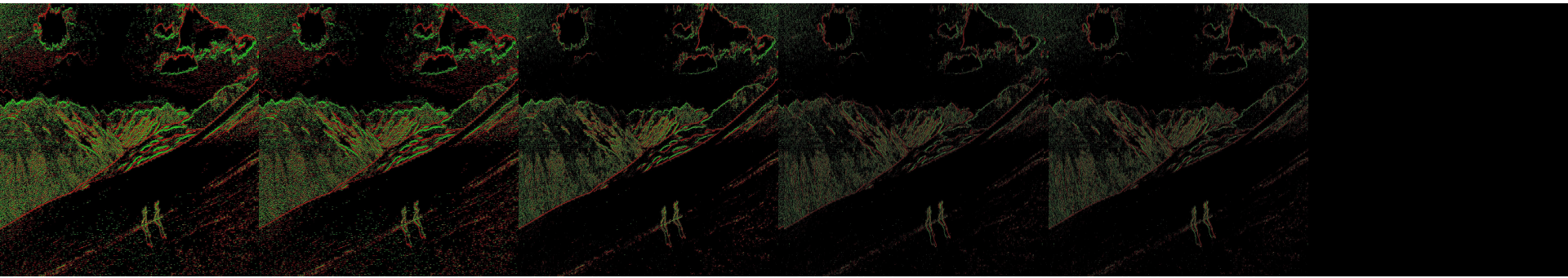
Compression Ratio = Original Size / Compressed Size

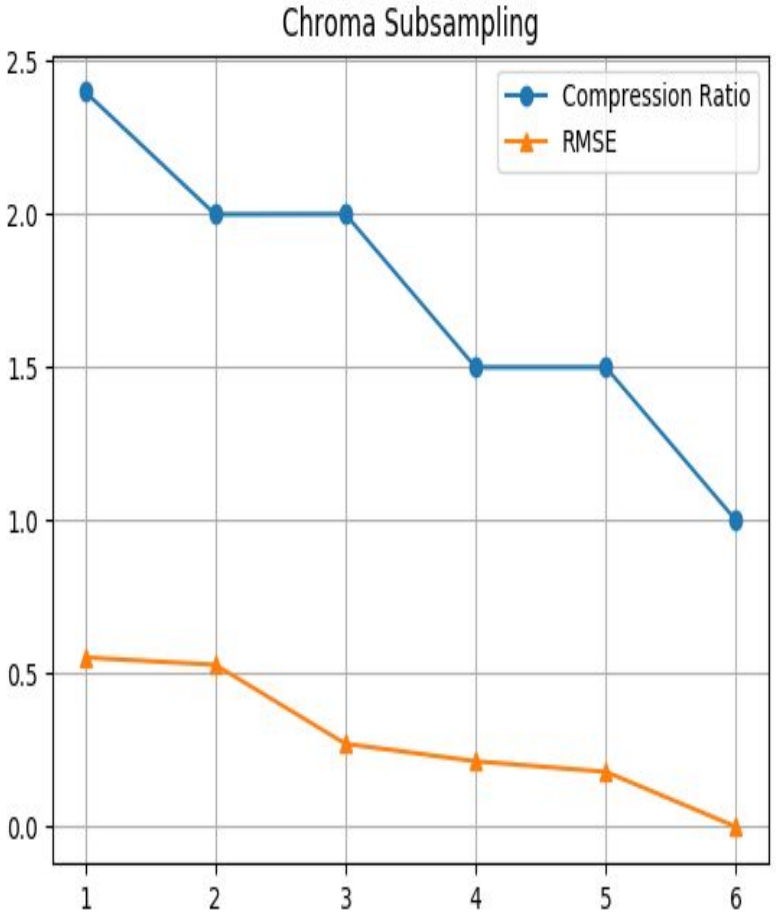
Summary of chroma subsampling on different sampling ratios

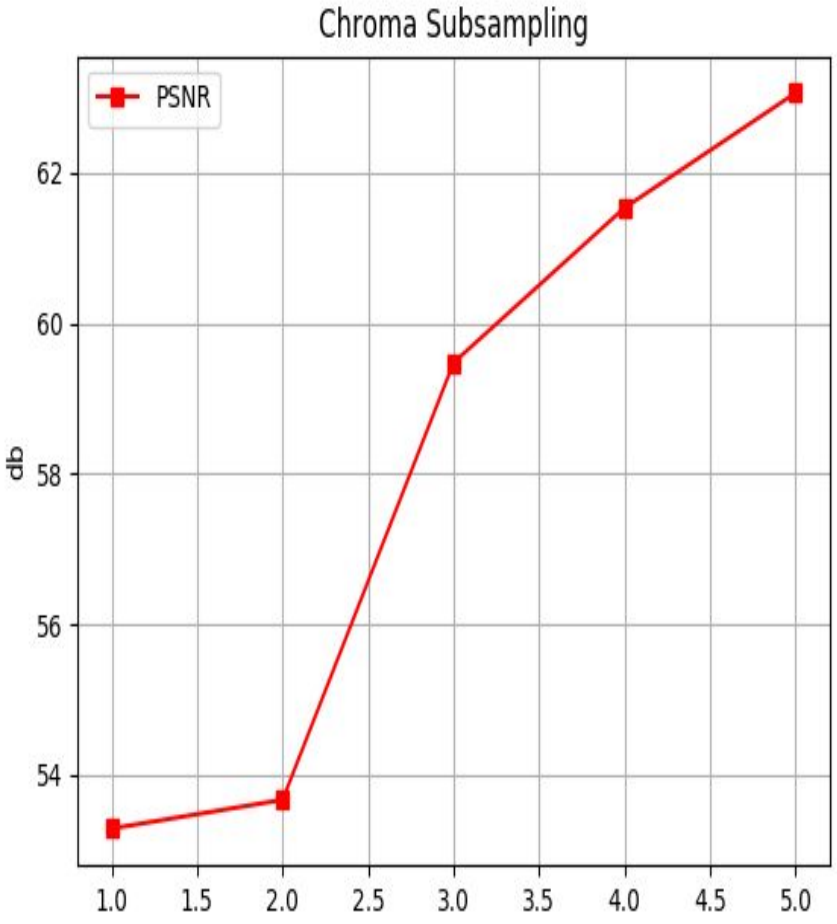


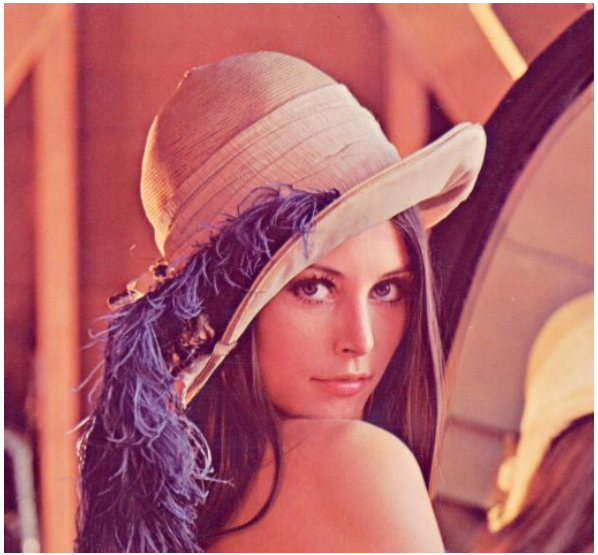


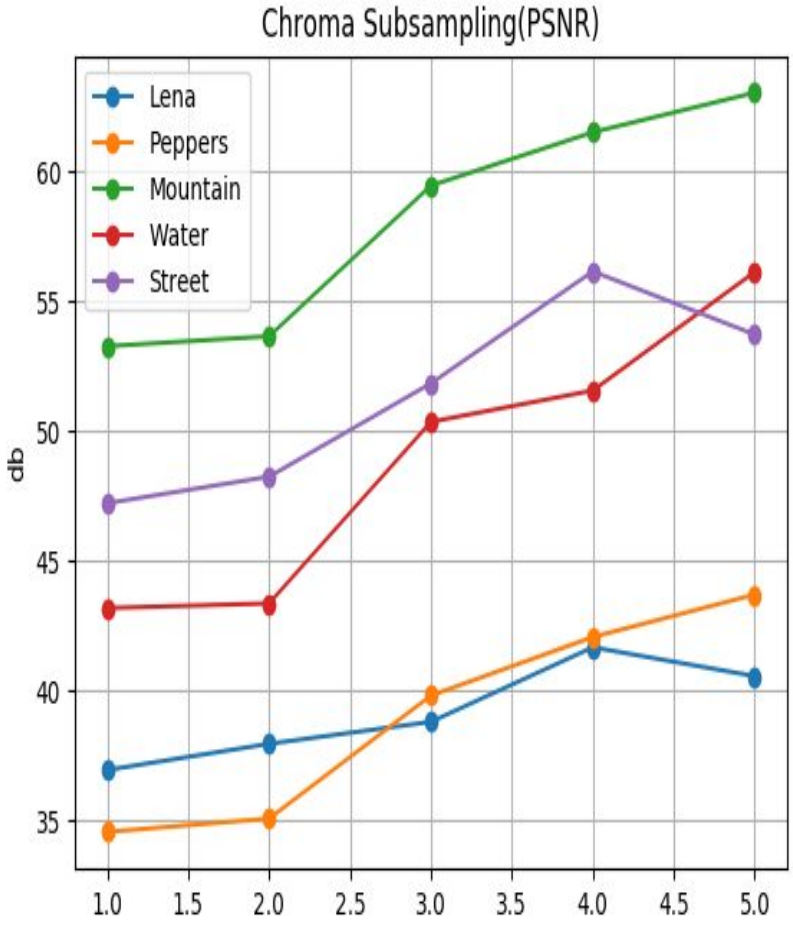
(4,1,0) (4,1,1) (4,2,0) (4,2,2) (4,4,0) (4,4,4)

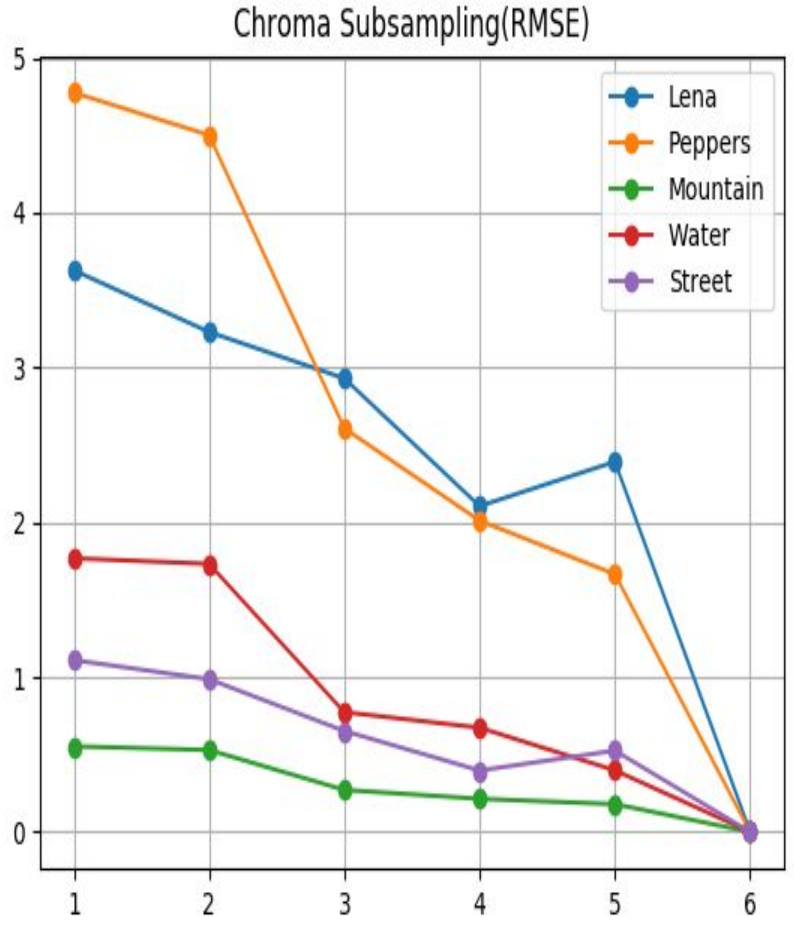



Chroma Subsampling
Compression Ratio
RMSE
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6


Chroma Subsampling
PSNR
db
62
60
58
56
54
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

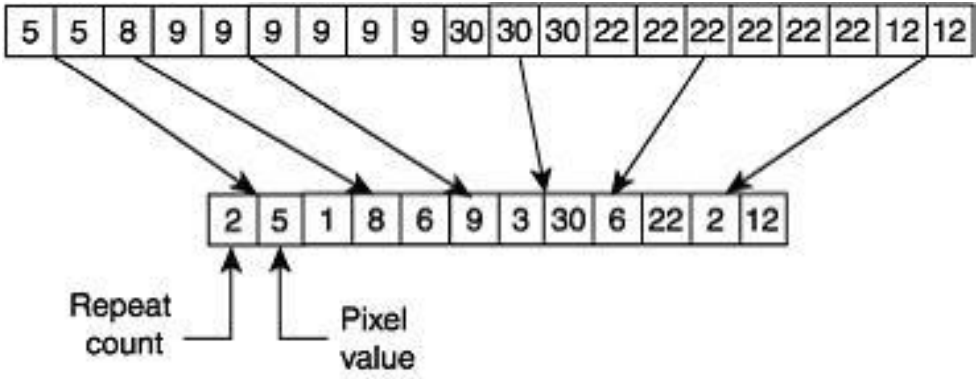



Chroma Subsampling(PSNR)
Lena
Peppers
Mountain
Water
Street
db
60
55
50
45
40
35
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

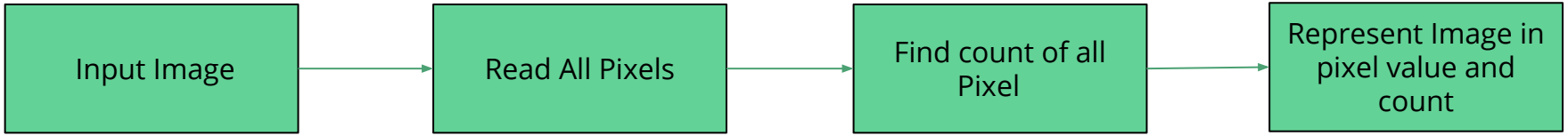

Chroma Subsampling(RMSE)
Lena
Peppers
Mountain
Water
Street
5
4
3
2
1
0
1
2
3
4
5
6

Run Length Encoding

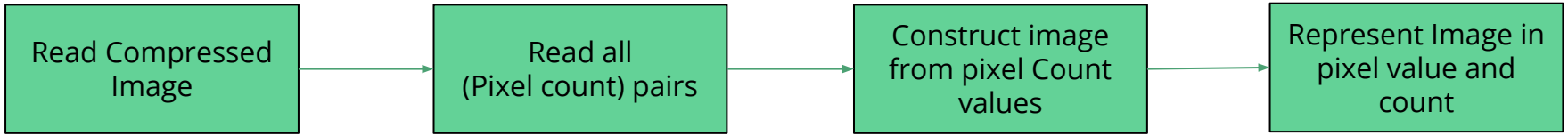
1. Lossless data compression
2. Single data value and count, rather than as the original run.
3. Useful on data that contains many such runs. Example - simple graphic images
4. Not useful with files that don't have many runs. Large file size.

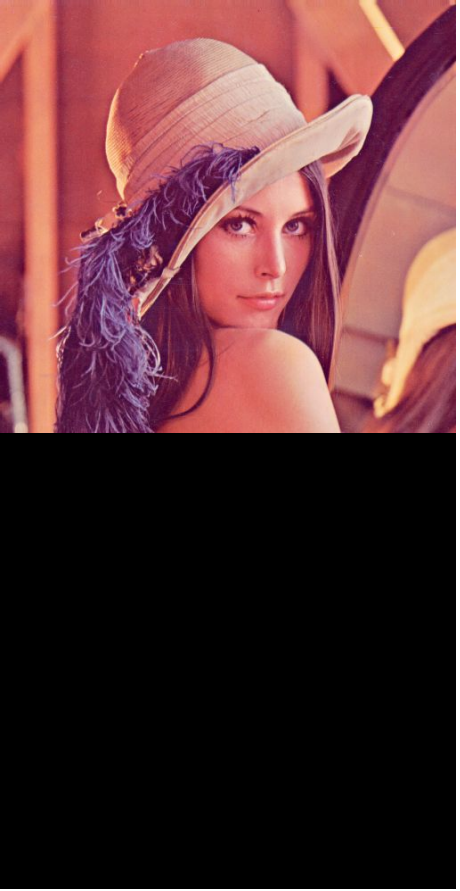
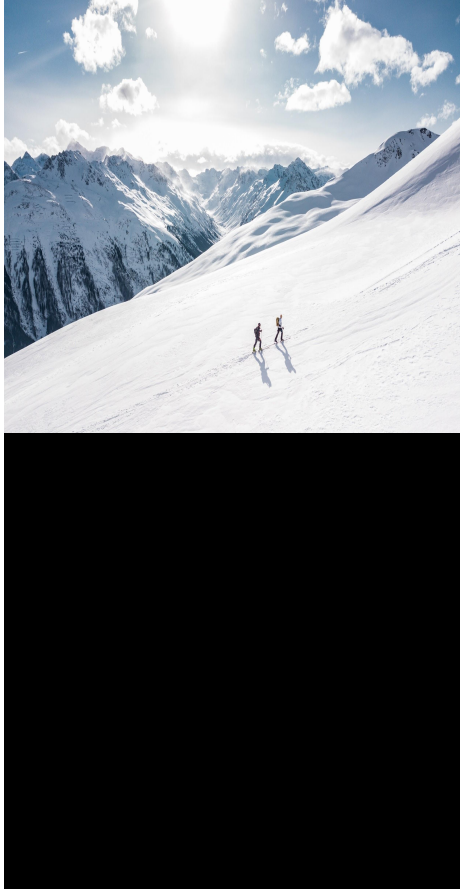
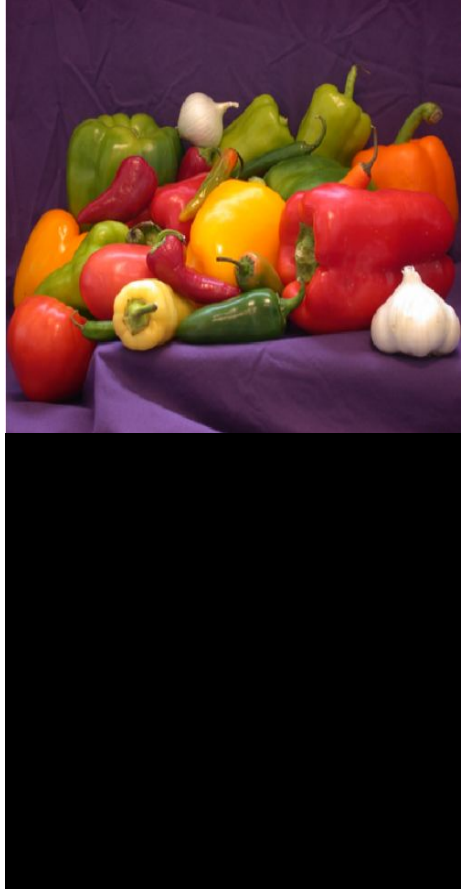
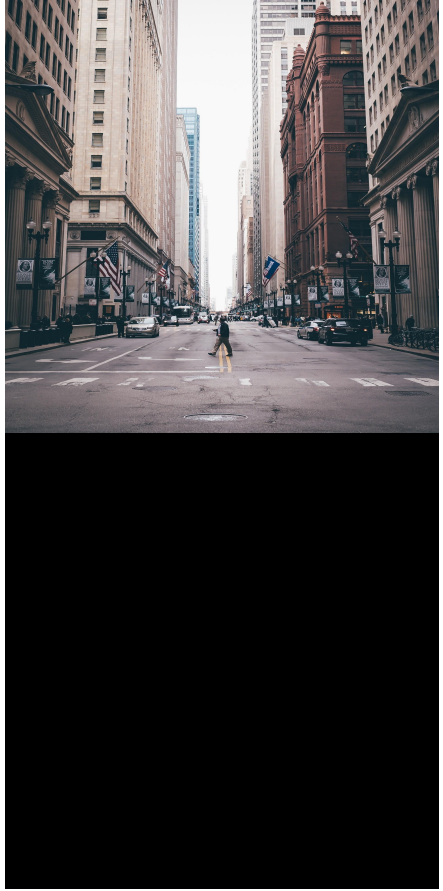


RLE Image Encoding

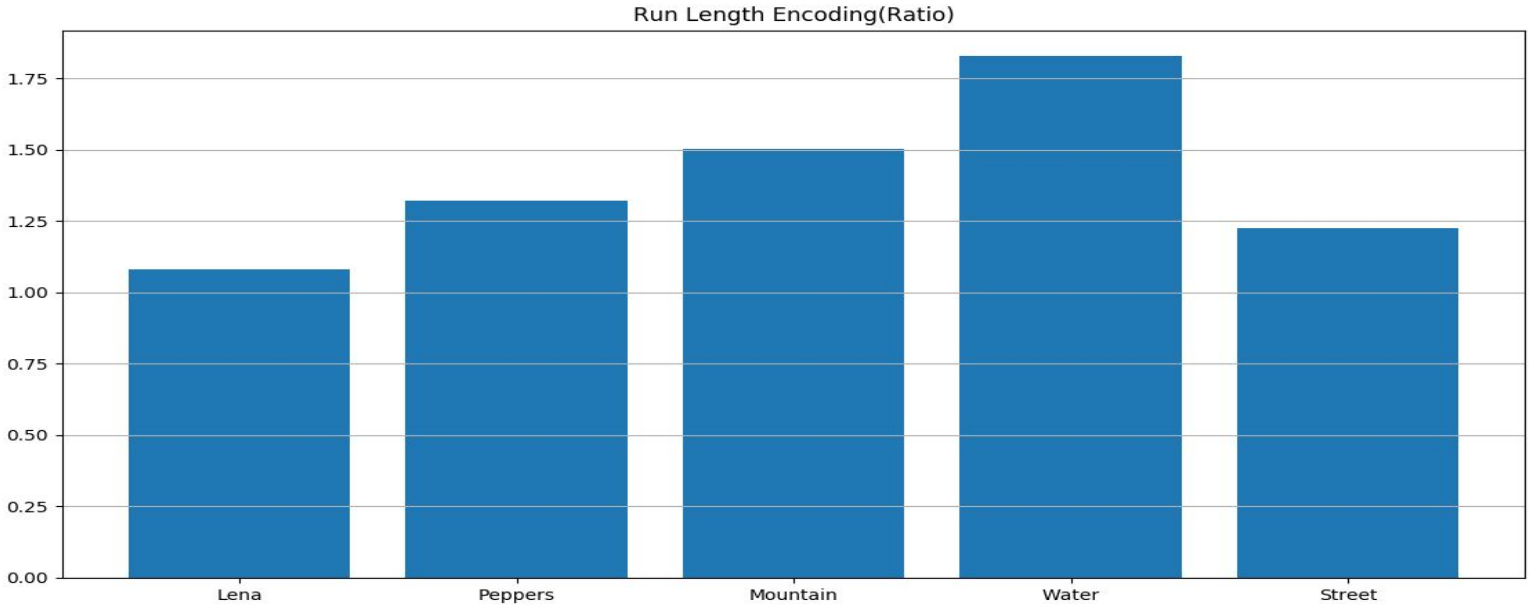


RLE Image Decoding



Lossless Compression, above picture shows pixel difference from original image

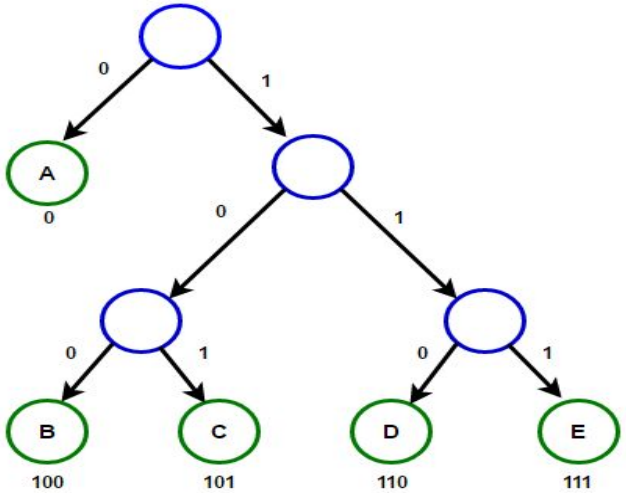


1.0815	1.3227	1.5024	1.8265	1.2249

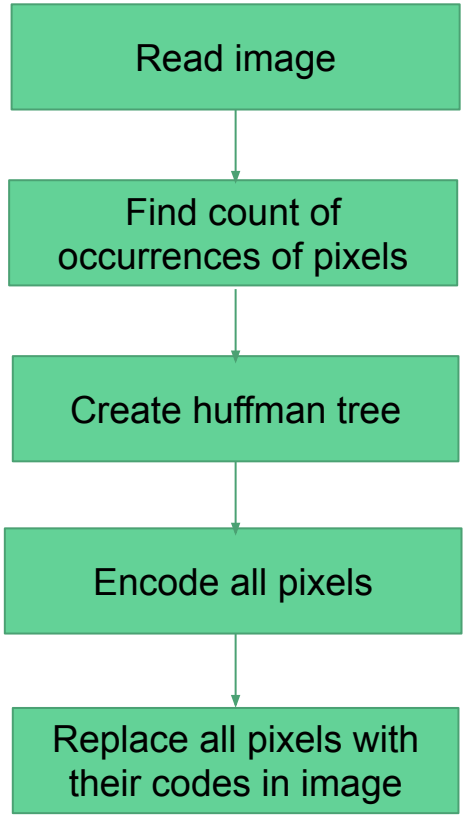
Compression Ratio = Original Size / Compressed Size

Huffman Encoding

1. Lossless data compression
2. Output variable-length code table for encoding a symbols
3. Assigns small Code to PIxels Appears More and Assigns big code to Symbol occurs Less
4. Overall Less bits per Pixel then Original image



Huffman Encoding

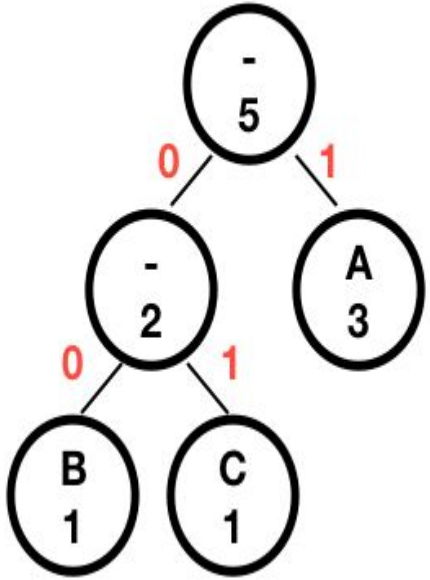


Time Complexity
O(NlogN)

$$H(X) = \sum_i p_i log_2(1/p_i)$$

$$L(X) = \sum_i p_i l_i$$

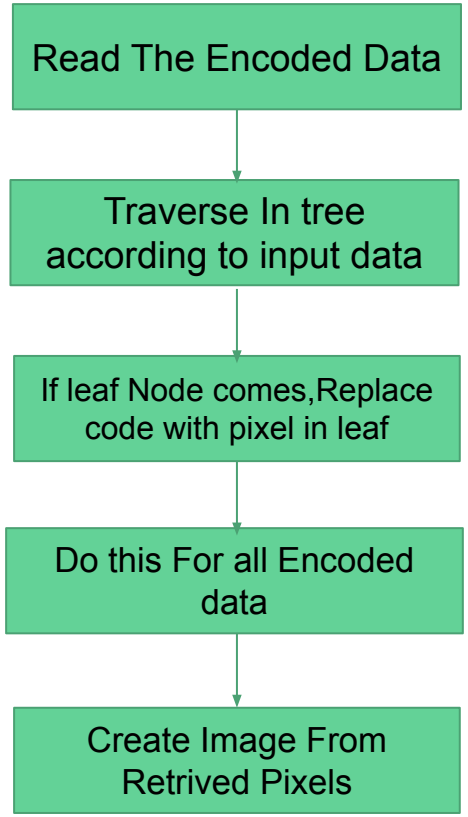
String to be encoded: ABACA

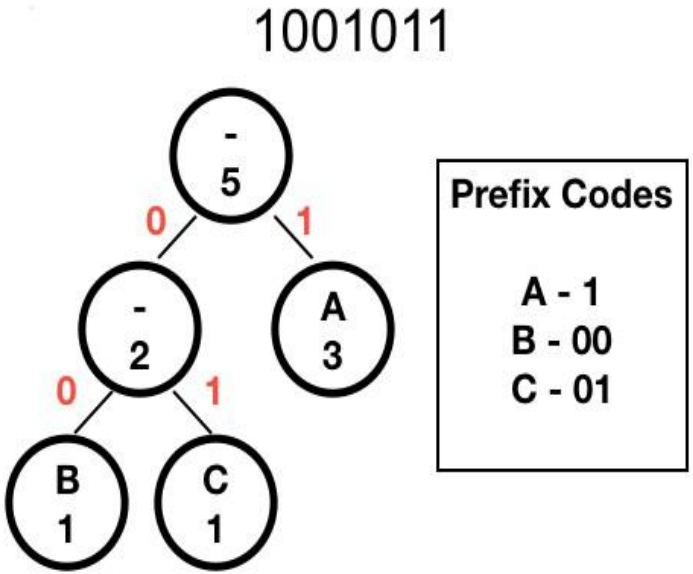


Prefix Codes

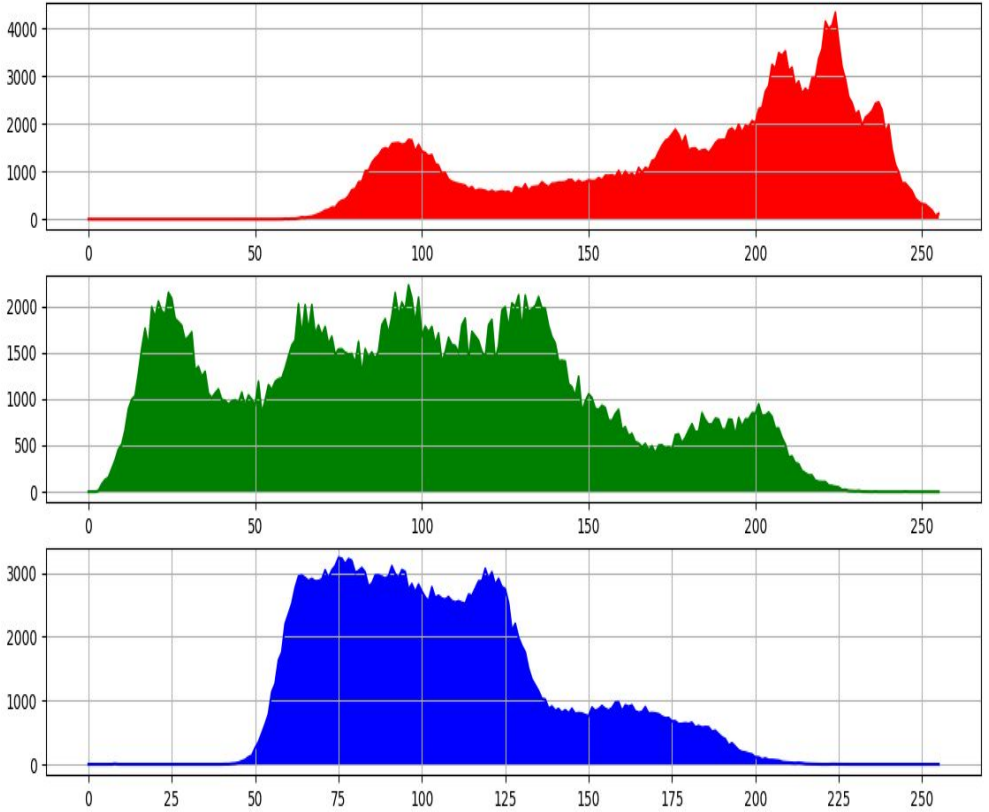
A - 1
B - 00
C - 01

Huffman Decoding





Pixel frequency distribution across 3 channels



Huffman Coding Example

```
[[196 204 215 171]
 [192 179 191 171]
 [162 145 180 194]
 [139 137 211 180]]
```

Symbol	Frequency	Probability	Huffman Code
171	2	0.125	010
180	2	0.125	100
137	1	0.0625	0000
139	1	0.0625	0001
145	1	0.0625	0010
162	1	0.0625	0011
179	1	0.0625	0110
191	1	0.0625	0111
192	1	0.0625	1010
194	1	0.0625	1011
196	1	0.0625	1100
204	1	0.0625	1101
211	1	0.0625	1110
215	1	0.0625	1111

Expected Length: 3.75

Entropy: 3.75

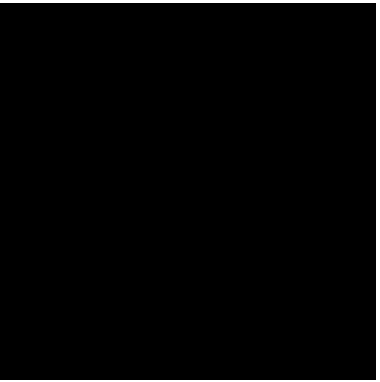
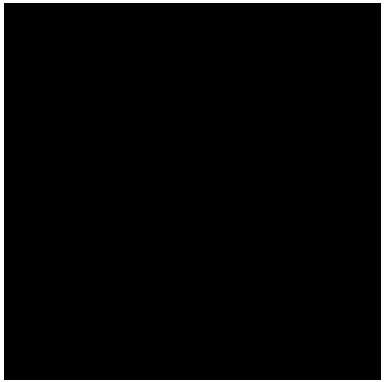
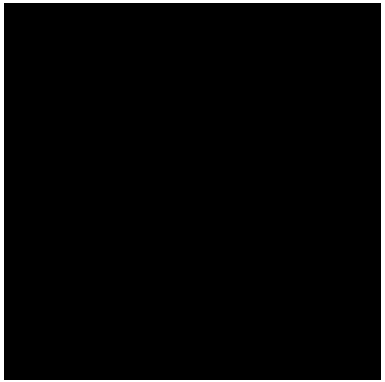
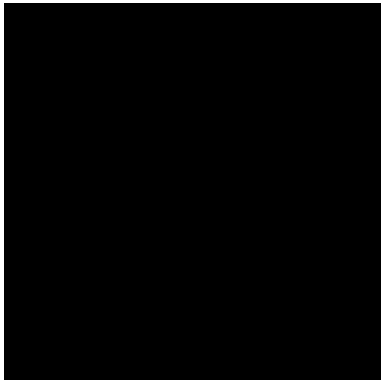
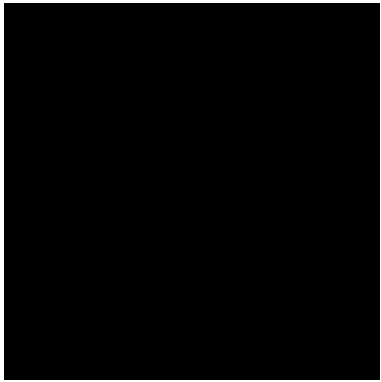
Huffman Coding Example

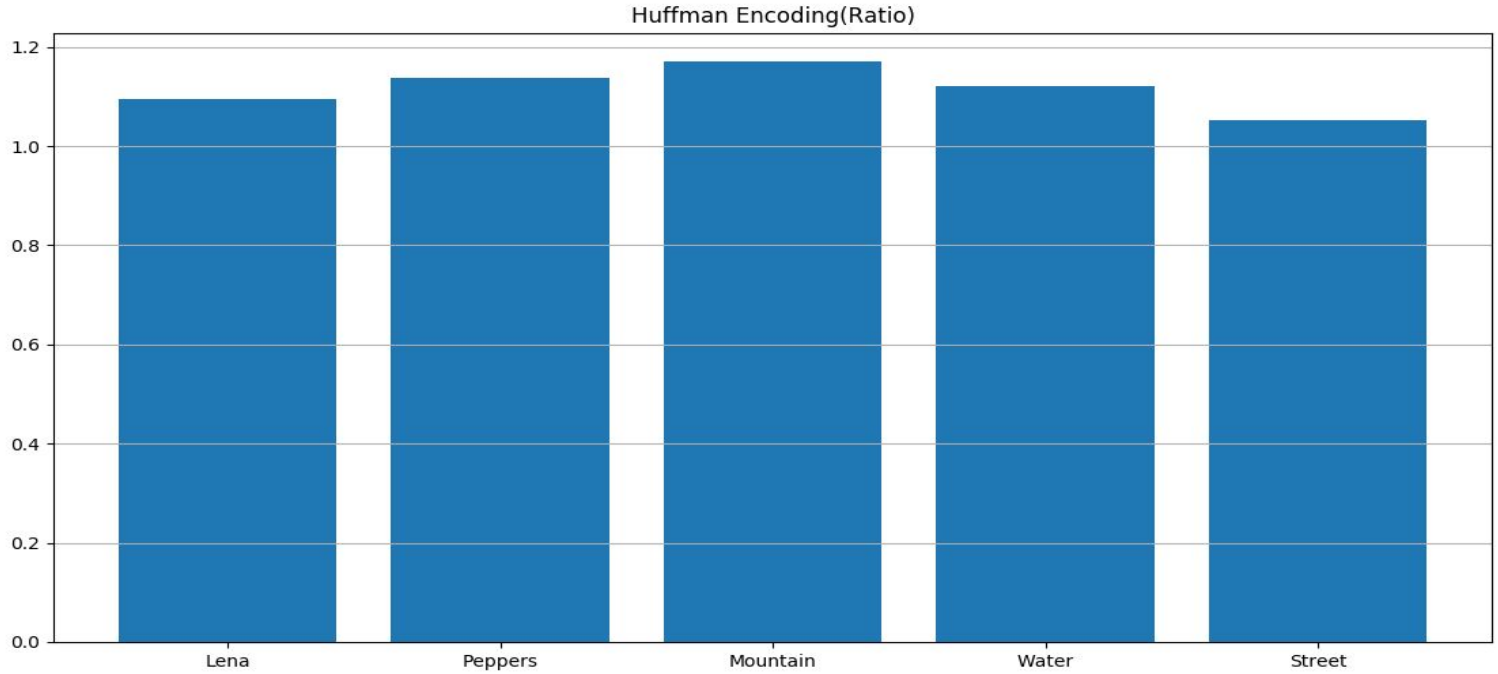
```
[[ 3  2  2  3  5  8 10 12]
 [ 2  2  3  4  5 12 12 11]
 [ 3  3  3  5  0  0  0  0]
 [ 3  3  4  6  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]]
```

Symbol	Frequency	Huffman Code	Probability
0	40	1	0.625
2	9	010	0.140625
3	8	001	0.125
5	4	0001	0.0625
12	4	0110	0.0625
4	3	01111	0.046875
21	1	00000	0.015625
6	1	000010	0.015625
8	1	000011	0.015625
10	1	011100	0.015625
11	1	011101	0.015625







1.0943	1.1385	1.1696	1.1202	1.0525

Compression Ratio = Original Size / Compressed Size

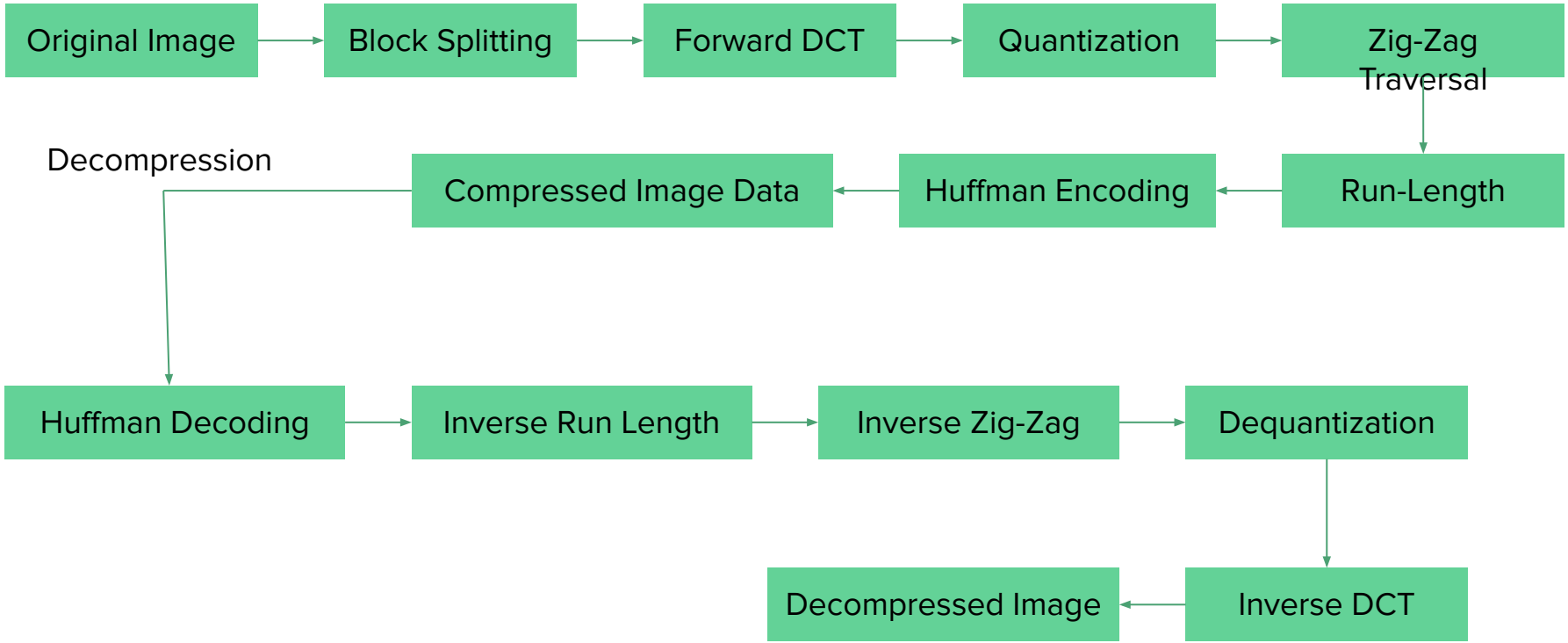
JPEG: Lossy Compression

JPEG stands for Joint Photographic Experts Group, the name of the committee that created the JPEG in 1992. One of the most popular lossy compression technique used for “visually lossless compression”.

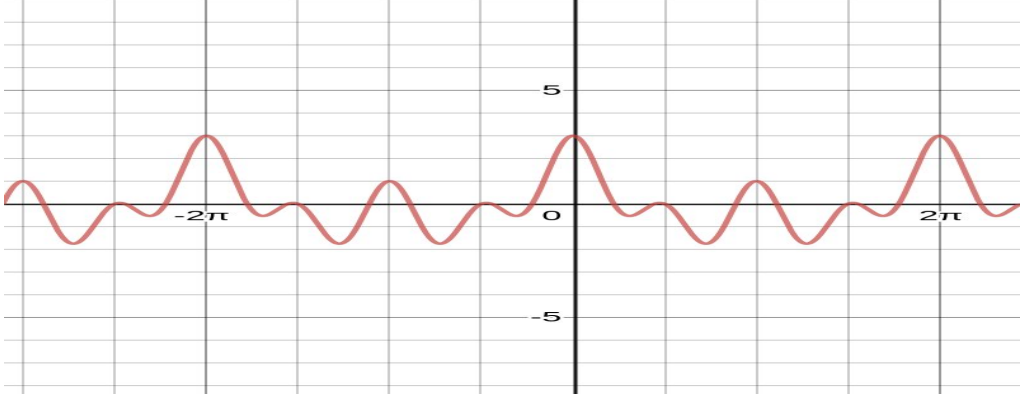
Whole algorithm can majorly divided into following steps:

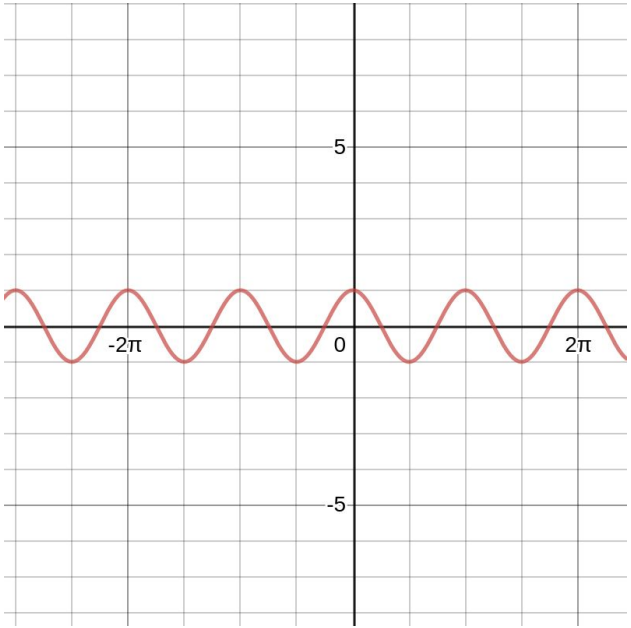
- Splitting
- Transform
- Quantization
- Encoding

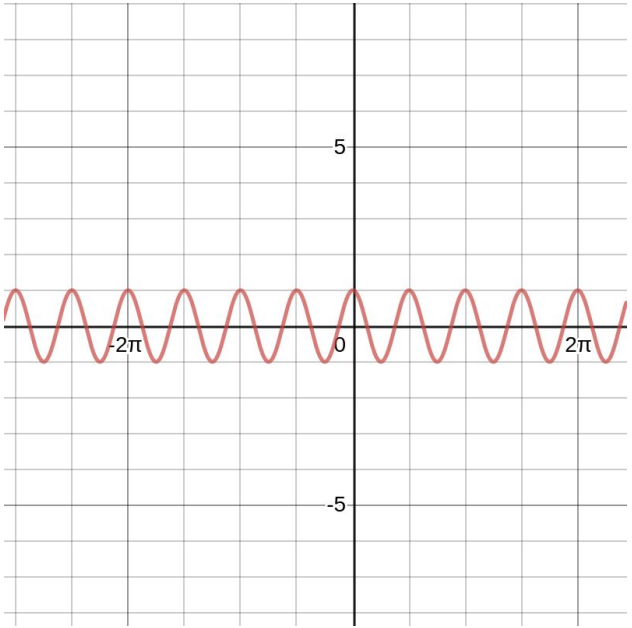
JPEG: Lossy Compression

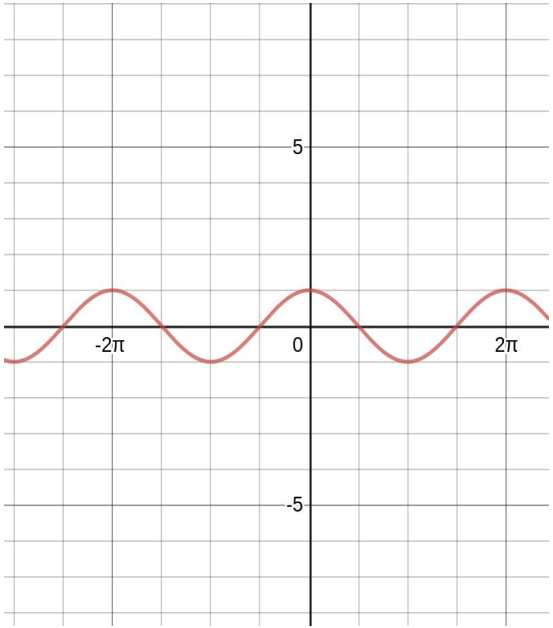


Block Splitting


5
-2π
0
2π
-5


5
-2π
0
2π
-5


5
-2π
0
2π
-5


5
-2π
0
2π
-5

DCT (Discrete cosine transform)

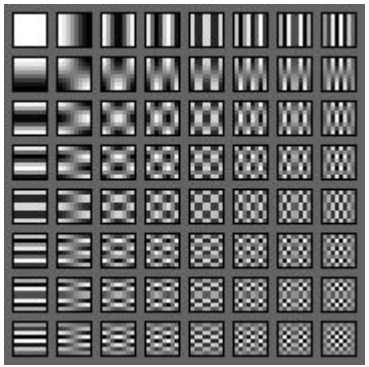
Discrete cosine transform expresses any finite data point as combinations of cosine functions

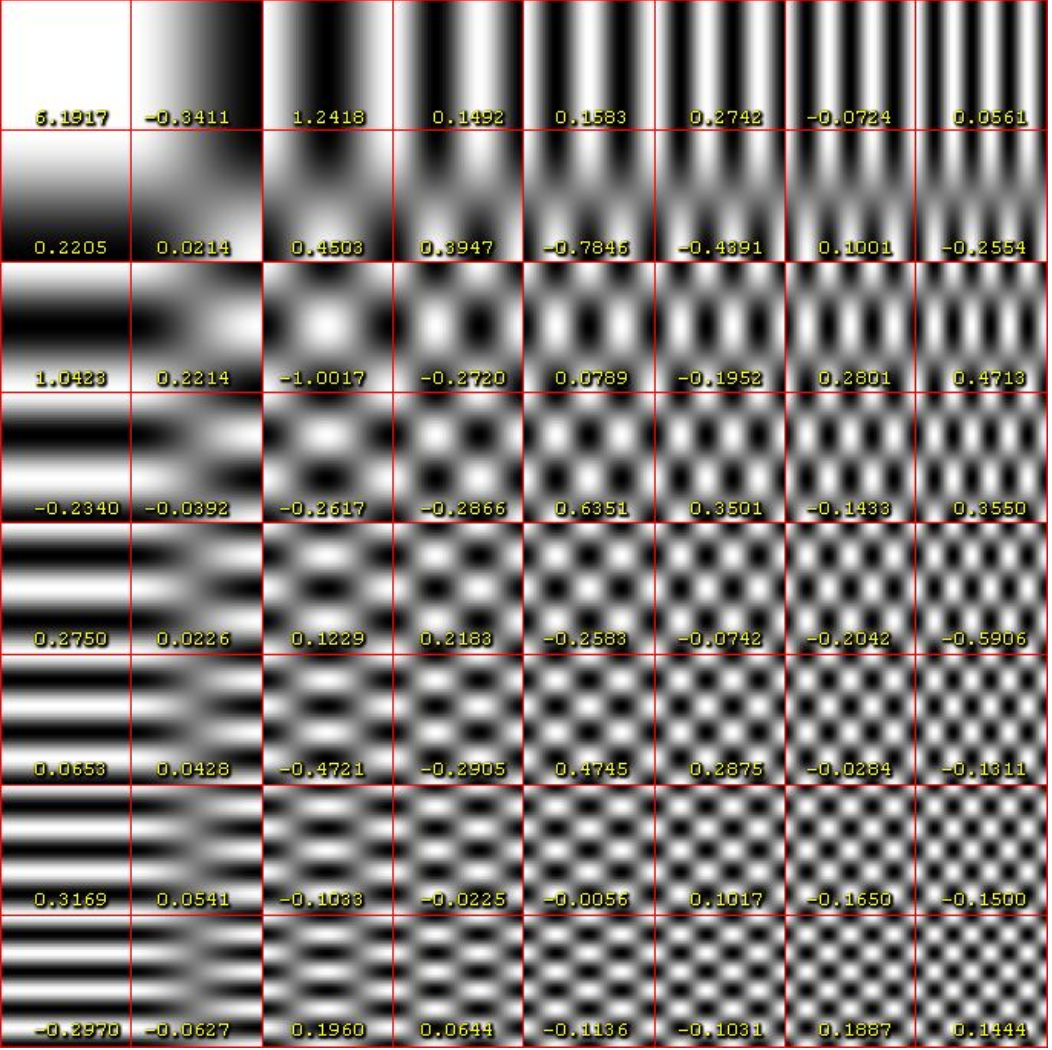
$$G_{u,v} = \frac{1}{4}\alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left[\frac{(2x+1)u\pi}{16}\right]\cos\left[\frac{(2y+1)v\pi}{16}\right]$$

$$\alpha(v) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } v = 0 \\ 1, & \text{otherwise} \end{cases} \qquad \alpha(u) = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } u = 0 \\ 1, & \text{otherwise} \end{cases}$$

DCT (Discrete cosine transform)

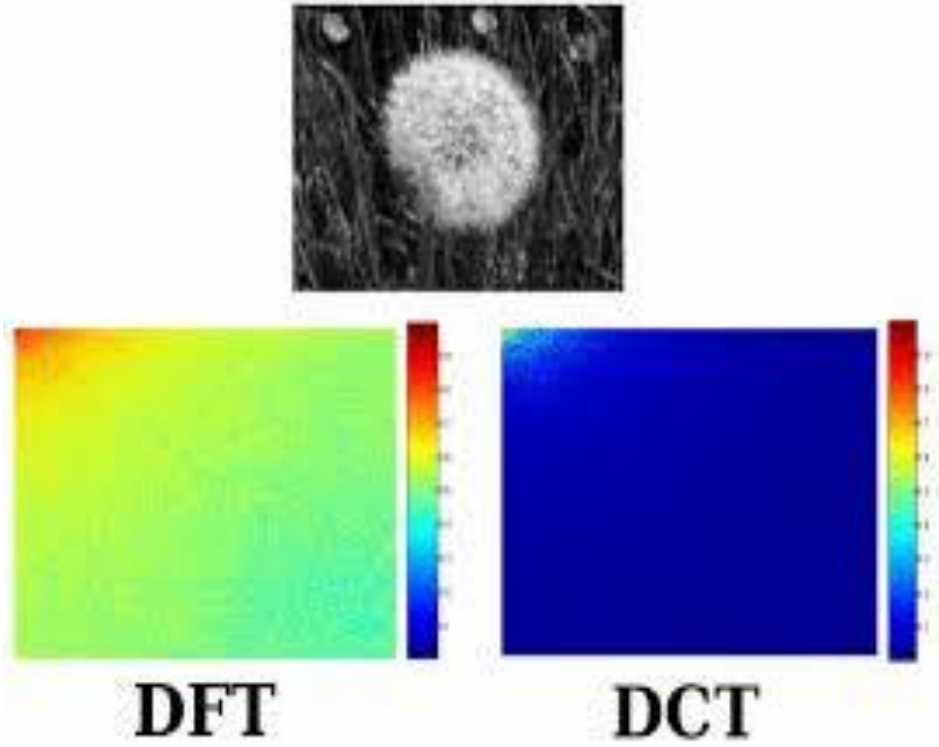
$$S(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} s(n_1, n_2) g(n_1, n_2, k_1, k_2)$$






```
6.1917  -0.3411 1.2418  0.1492  0.1583  0.2742  -0.0724 0.0561
0.2205  0.0214  0.4503  0.3947  -0.7846 -0.4391 0.1001  0.2554
1.0423  0.2214  -1.0017 -0.272  0.0789  0.1952  0.2801  0.4713
-0.234  -0.0392 0.2617  -0.2866 0.6351  0.3501  -0.1433 0.355
0.275   0.0226  0.1229  0.2183  -0.2583 -0.0742 -0.2042 0.5906
0.0653  0.0428  -0.4721 -0.2905 0.4745  0.2857  -0.0284 -0.1311
0.3169  0.0541  -0.1033 -0.0255 -0.0056 0.1017  -0.165  -0.15
-0.297  -0.0627 0.196   0.0644  -0.1136 0.1031  0.1887  0.1444
```

[2] Wikipedia contributors. (2019, November 11). Discrete cosine transform. In Wikipedia, The Free Encyclopedia. Retrieved 02:22, November 14, 2019, from https://en.wikipedia.org/w/index.php?title=Discrete_cosine_transform&oldid=925713320



[3] Michael H. Perrott(6.082 Fall 2006) Introduction to EECS 2

Reason for Quantization

- The human eye is good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. This allows one to greatly reduce the amount of information in the high frequency components
- Most important step & major lossy step in the whole algorithm. We make higher frequency elements zero.

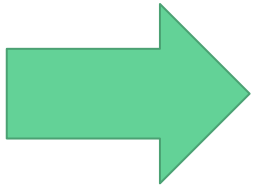
[Lossy Operation]
Dividing the Frequency Matrix with Quantization Matrix

Sample Quantization Matrix

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	28	31

```
[[1020  113  -20 -181 -273  -37  -82 -167]
 [ -33 -178   83  137  -47 -201  -81  103]
 [ -69 -174   67 -144  105 -145 -113  130]
 [  27  204  -17   71  195 -112  166  -42]
 [  77  -29  -31  -14   87  -14  -18  -23]
 [-160   29 -101  -18   29  -58 -118   39]
 [ -53   -2   93   86  103  -18  -95    8]
 [-165   89   21   39  111  -51  -92    9]]
```

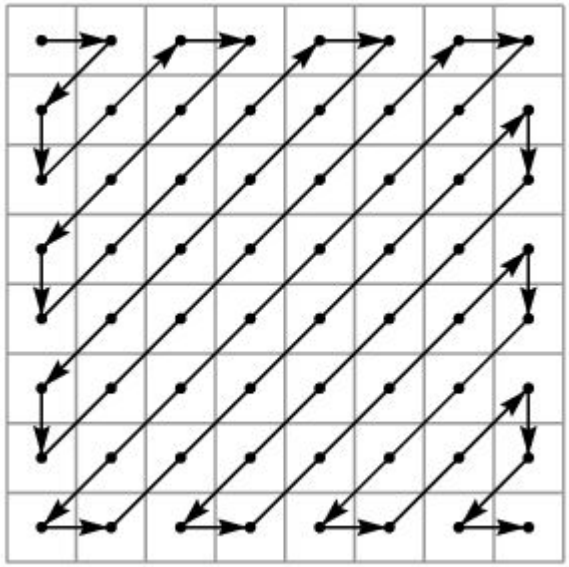
Element-wise division by
Quantization matrix



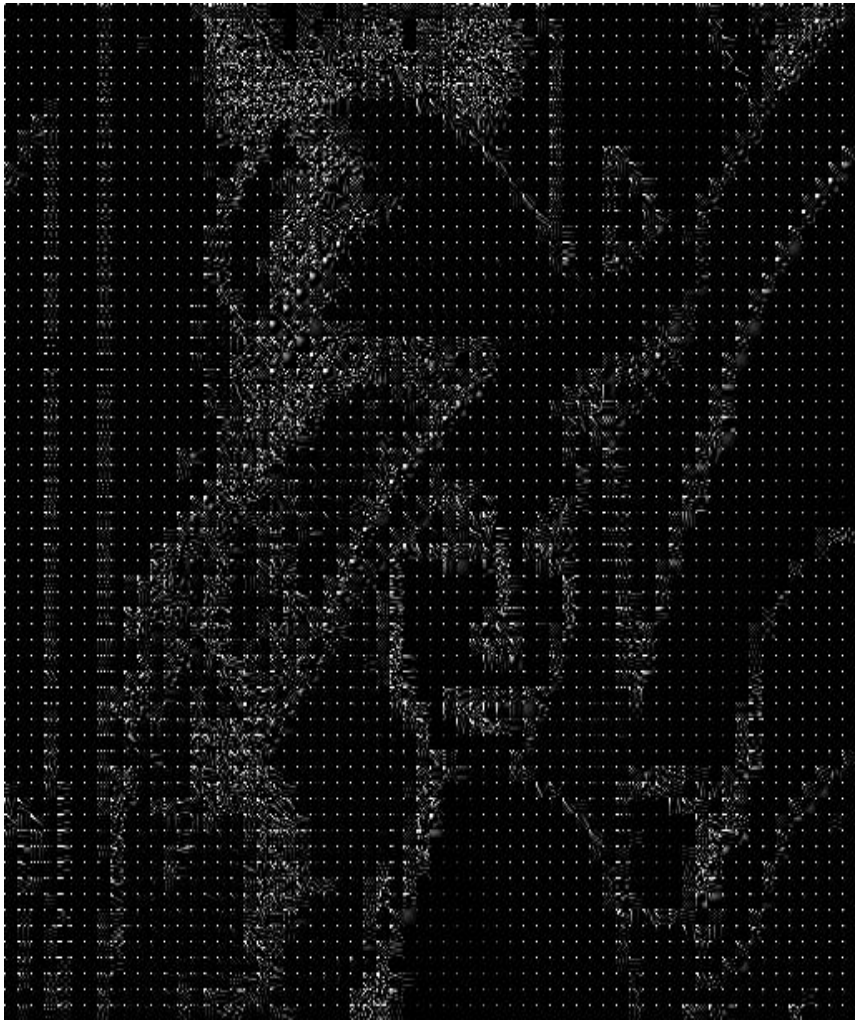
```
[[68   4   0  -4  -4   0  -1  -1]
 [-1  -5   1   2   0  -2   0   1]
 [-1  -3   1  -2   1  -1  -1   1]
 [ 0   3   0   0   2  -1   1   0]
 [ 1   0   0   0   0   0   0   0]
 [-2   0  -1   0   0   0   0   0]
 [ 0   0   0   0   0   0   0   0]
 [-1   0   0   0   0   0   0   0]]
```

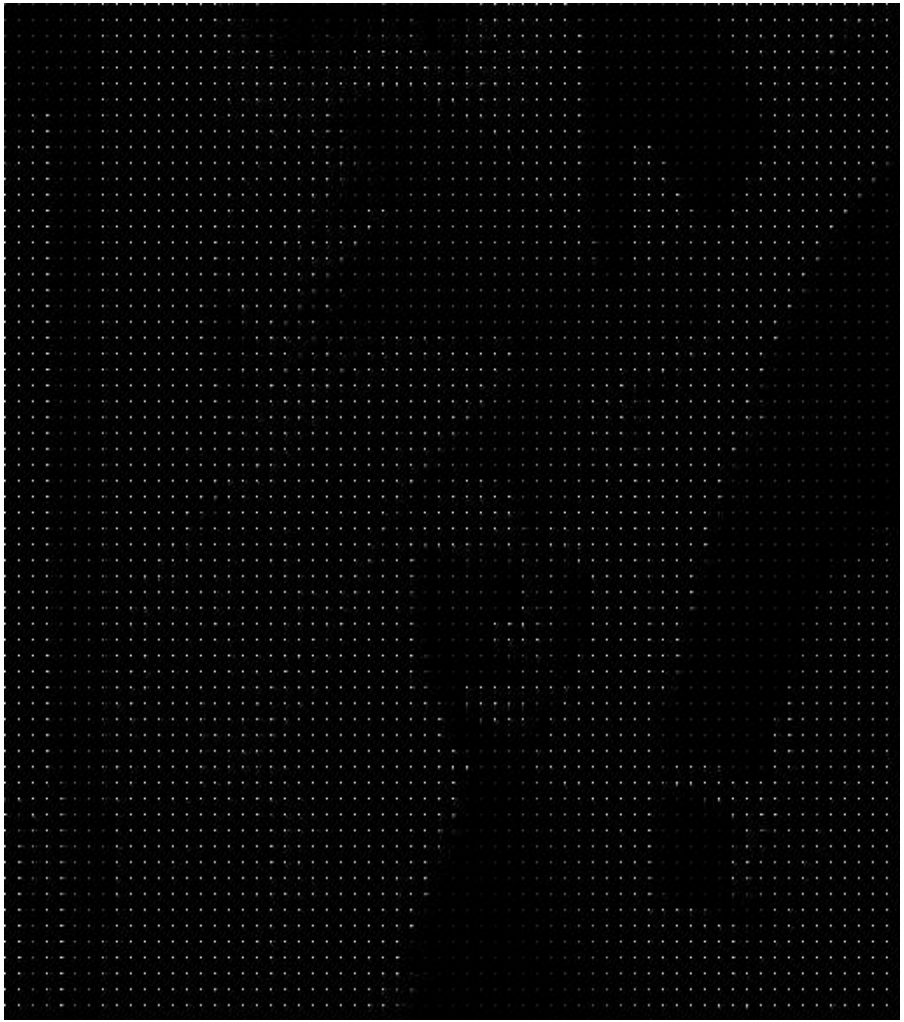
Zig-Zag Traversal

- By observation we can see most of the zeros are concentrated around bottom right corner of the matrix. So, to make run-length more efficient we traverse in such a way to maximize runs of consecutive zeros.



```
[68.  4. -1. -1. -5.  0. -4.  1. -3.  0.  1.  3.  1.  2. -4.  0.  0. -2.
  0.  0. -2.  0.  0.  0.  0.  1. -2. -1. -1.  0. -1.  2.  0. -1.  0. -1.
  0.  0.  0.  0. -1. -1.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

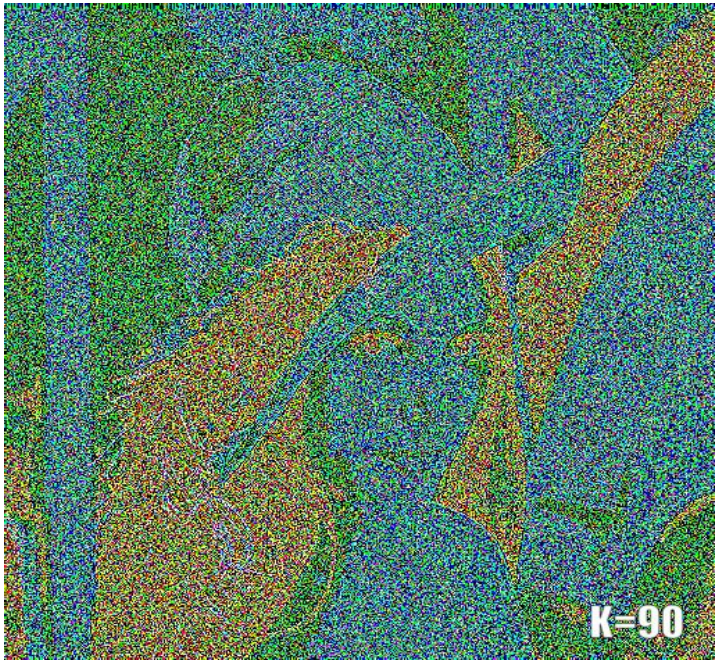




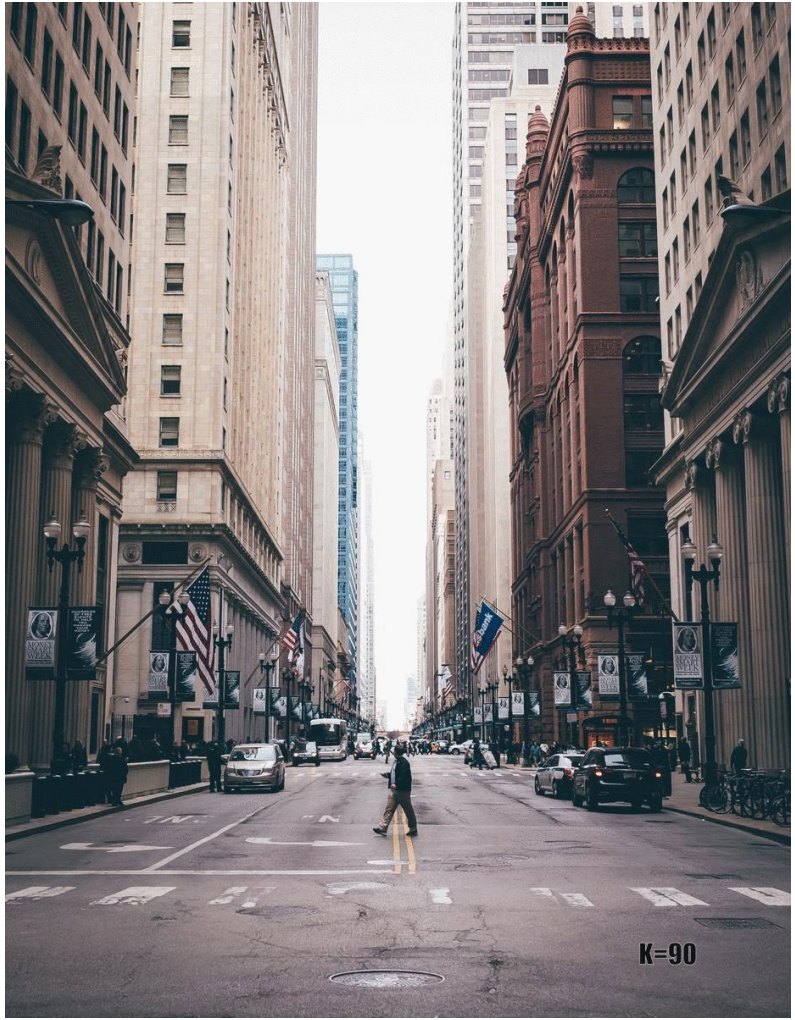
When alpha is increased (large values of alpha correspond to smaller values of the quality parameter q), more information is lost, and the file size decreases.

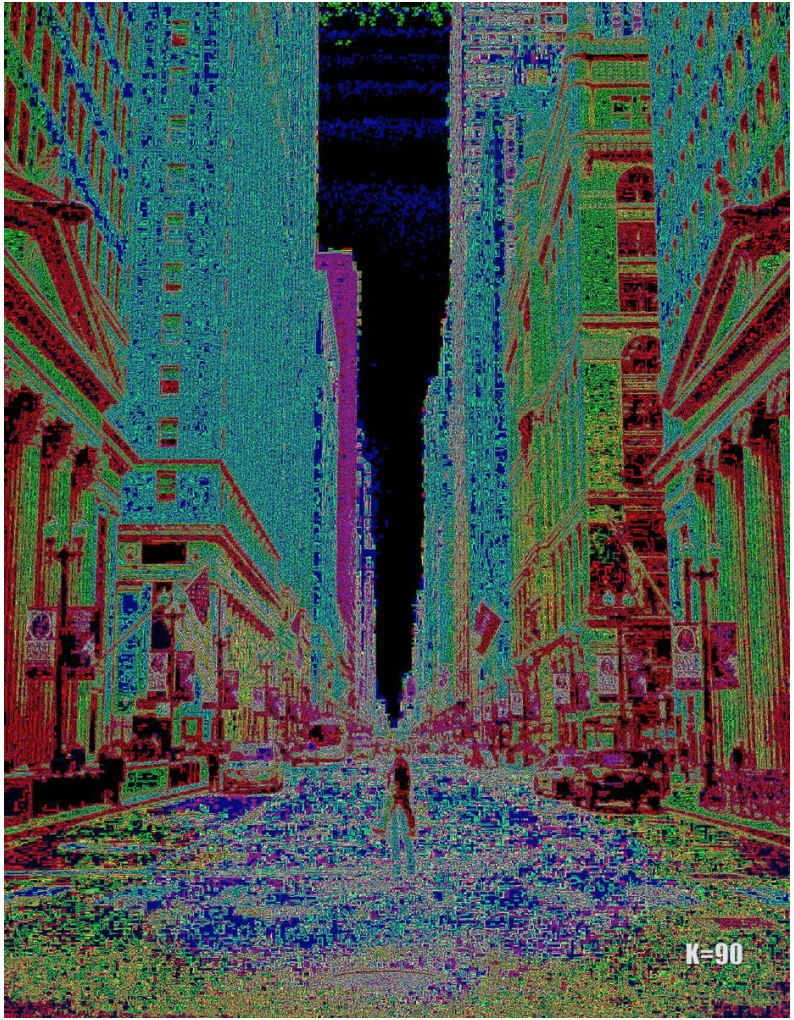
$$\alpha = \begin{cases} \frac{50}{q} & \text{if } 1 \leq q \leq 50 \\ 2 - \frac{q}{50} & \text{if } 50 \leq q \leq 100 \end{cases}$$

Here q = K



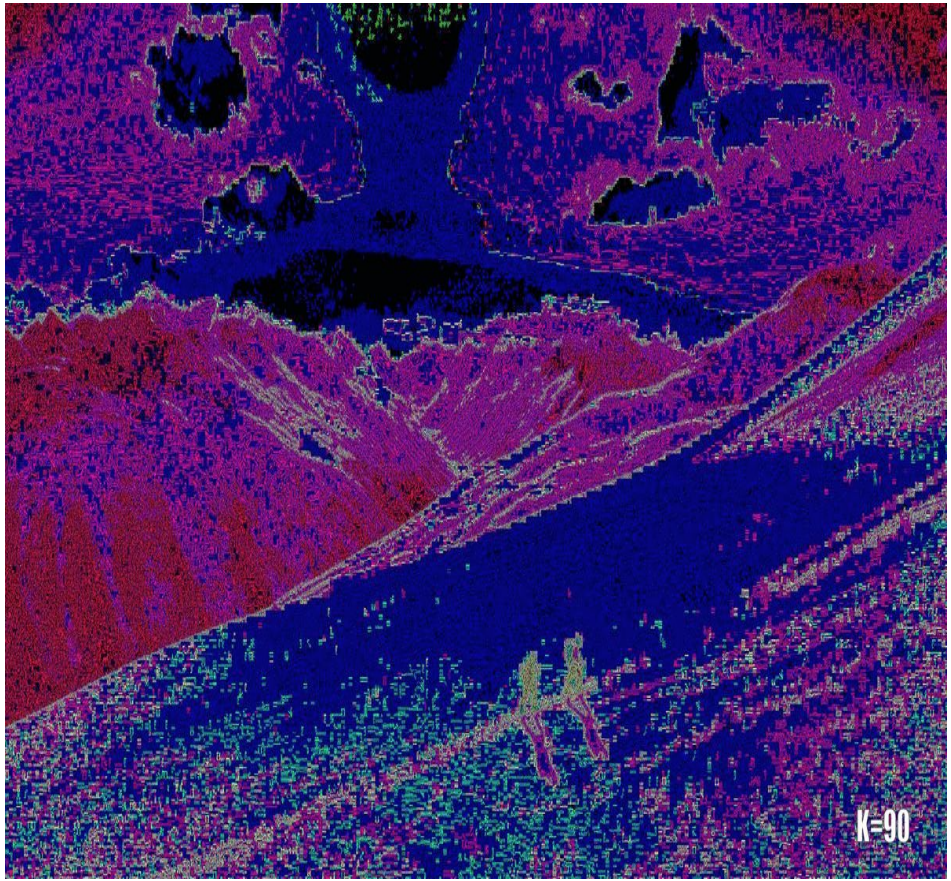
K	90	75	50	25	10
n(runlength)	1.5981	2.6231	3.3799	4.2855	6.9888
n(huffman)	2.39	3.9178	5.4812	7.5972	12.8735
RMSE	1.7894	3.1403	5.0993	8.7734	15.7249
PSNR	43.0762	38.1912	33.9804	29.2673	24.1990


K=90


K=90

K	90	75	50	25	10
n(runlength)	2.3062	2.7009	3.1106	3.8548	6.1826
n(huffman)	2.9046	3.8784	4.9917	6.6736	10.9128
RMSE	1.3316	2.7594	5.1355	9.2462	16.3912
PSNR	45.6431	39.3144	33.9191	28.8114	23.8385


K=90

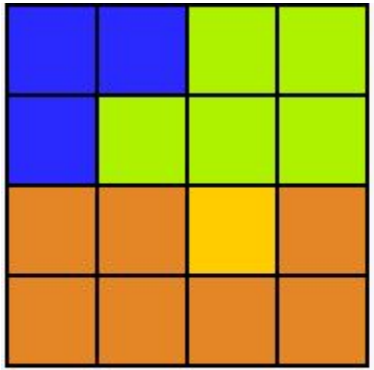

K=90

K	90	75	50	25	10
n(runlength)	4.0612	5.1997	6.7302	8.4133	12.7442
n(huffman)	5.4984	7.6391	10.5097	14.2167	22.7549
RMSE	1.1524	2.0161	3.4733	5.9909	10.6638
PSNR	46.8986	42.0404	37.3159	32.5808	27.5725

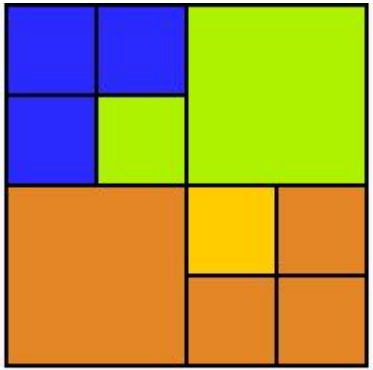
Artifacts

QuadTree Image Compression

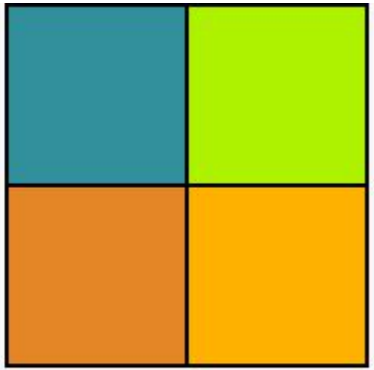
Quadtrees take advantage of the fact that cells in a grid are often the same as adjacent cells – for example, a green pixel is very likely surrounded by other green pixels.



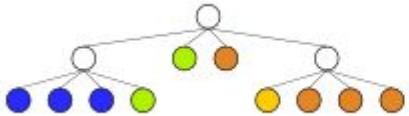
Original Image.

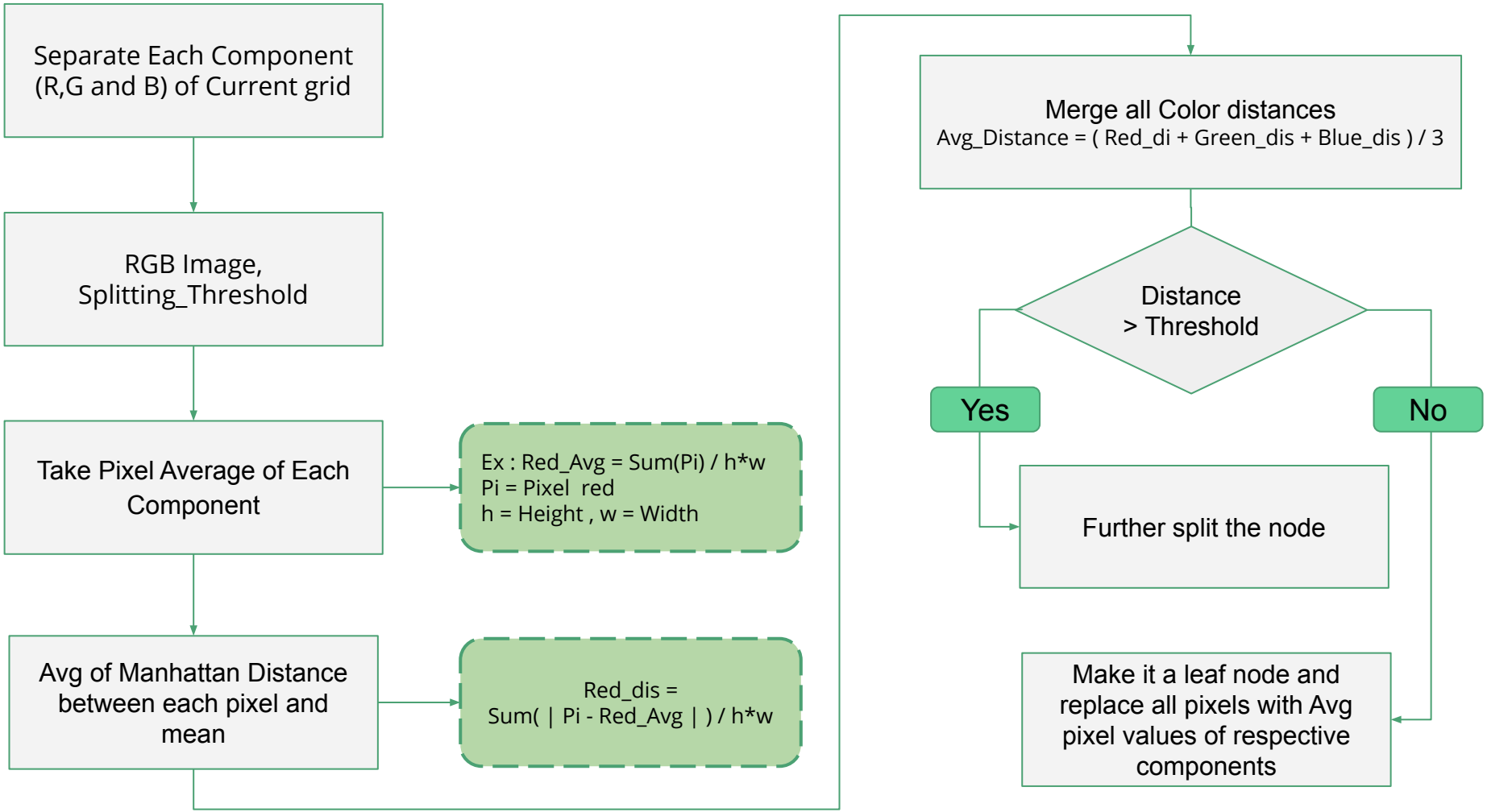


Quadtree representation of the image.



Quadtree representation with a higher tolerance value




Separate Each Component (R,G and B) of Current grid
RGB Image, Splitting_Threshold
Take Pixel Average of Each Component
Ex : Red_Avg = Sum(Pi) / h*w
Pi = Pixel red
h = Height , w = Width
Avg of Manhattan Distance between each pixel and mean
Red_dis = Sum(| Pi - Red_Avg |) / h*w
Merge all Color distances
Avg_Distance = (Red_di + Green_dis + Blue_dis) / 3
Distance > Threshold
Yes
No
Further split the node
Make it a leaf node and replace all pixels with Avg pixel values of respective components

Node Split Calculation

Original Matrix

0	10	18	19	95	95	99	98
8	15	22	24	98	101	95	95
35	30	48	46	99	100	102	101
36	37	45	50	95	100	102	100
150	180	230	96	73	72	75	71
255	145	90	180	73	70	71	70
255	176	200	205	75	74	73	71
136	91	210	228	75	75	75	70

Threshold = 30

Avg (Complete Matrix) = 93

Avg Manhattan Distance = 43

0	10	18	19
8	15	22	24
35	30	48	46
36	37	45	50

95	95	99	98
98	101	95	95
99	100	102	101
95	100	102	100

150	180	230	96
255	145	90	180
255	176	200	205
136	91	210	228

73	72	75	71
73	70	71	70
75	74	73	71
75	75	75	70

Calculate Avg Manhattan for all three component (RGB) and calculate their average. Resultant Average will be compared with threshold, if more then given threshold value so matrix will be further splitted (8x8 -> 4x4)

0	10	18	19
8	15	22	24
35	30	48	46
36	37	45	50

Threshold = 30

Avg = 27
Dis = 13

27

95	95	99	98
98	101	95	95
99	100	102	101
95	100	102	100

Threshold = 30

Avg = 98
Dis = 2

98

150	180	230	96
255	145	90	180
255	176	200	205
136	91	210	228

Threshold = 30

Avg = 162
Dis = 48

150	180	230	96
255	145	90	180

255	176	200	205
136	91	210	228

73	72	75	71
73	70	71	70
75	74	73	71
75	75	75	70

Threshold = 30

Avg = 74
Dis = 2

74

Resultant Matrix

Original Matrix

0	10	18	19	95	95	99	98
8	15	22	24	98	101	95	95
35	30	48	46	99	100	102	101
36	37	45	50	95	100	102	100
150	180	230	96	73	72	75	71
255	145	90	180	73	70	71	70
255	176	200	205	75	74	73	71
136	91	210	228	75	75	75	70

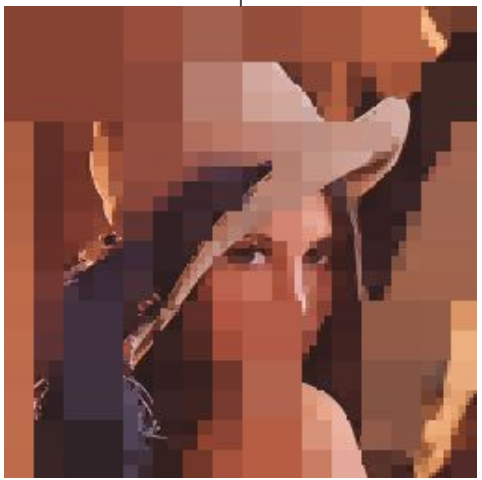
Result

<table>
<tr><td colspan="4">27</td><td>98</td></tr>
<tr><td>150</td><td>180</td><td>230</td><td>96</td><td rowspan="4">74</td></tr>
<tr><td>255</td><td>145</td><td>90</td><td>180</td></tr>
<tr><td>255</td><td>176</td><td colspan="2" rowspan="2">210</td></tr>
<tr><td>136</td><td>91</td></tr>
</table>

Threshold Value
100

Threshold Value: 300

Threshold Value
700



Compression Ratio after Run length over Quad image

Threshold Value 10	3.36
Threshold Value 50	4.84
Threshold Value 100	5.29
Threshold Value 300	7.21
Threshold Value 500	8.37

Compression Ratio

Threshold Value 10

Threshold Value 50

Threshold Value 70

Run Length Encoding

Compression Ratio

3.49

7.36

8.61