Homework 3

DPCM

2020311381 Sanghyeok Choi

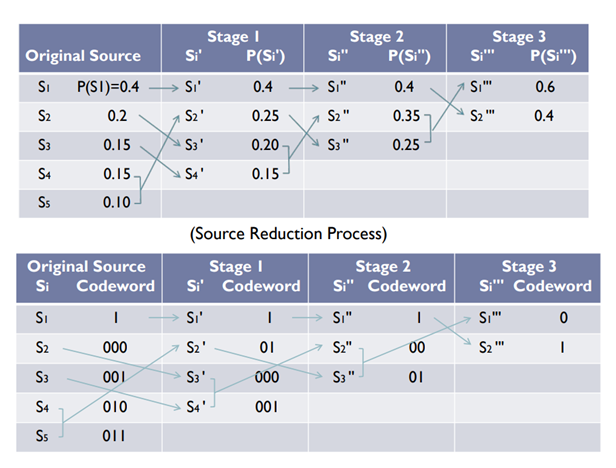
**Goal of Project** : Understanding Linear Transformations

Practicing Image Coding with Huffman coding, DPCM, Non-uniform Quantization (Lioyd-Max Method).

**Tools :** Huffman Code, DPCM, Lioyd-Max Non-uniform Quantization

1. Huffman Code

Huffman Code is one of encoding method. It depends on probability. When the higher probability the shorter code length, entire code would be short so compression effect happen. However you have to consider decoding, so every code should be decodable. For example, there are 4 symbol; a0,a1,a2,a3 that each code are 0, 10, 101, 0101. Then code 0101 can be decode 0 with 101=> a­0a2 and 0101=>a3. So it cannot decode accurate.



This is example of Huffman coding. Sort in descend by probability in each iteration. And add codeword in last two probability and combine them.

Let’s see more detail about example. There are 5 symbols; S1, S2, S3, S4, S5 and each probability are 0.4, 0.2, 0.15, 0.15, 0.1. They are already sort descend, if they don’t sort descend by probability. S5 has lowest probability and S4 has next probability. There are method that one is set all Huffman distribution and make code by each condition, the other method is make code in each step. There are no difference in result but I think second method is more reasonable. So Now S1, S2, S3 are not change but S4 add “0” and S5 add “1”.

So S1 = “” , S2 = “”, S3 = “”, S4 = “0”, S5 = “1”. Now combine last two probability S4 and S5 to {S4, S5} and probability become 0.15 + 0.1 = 0.25.

Now sort symbol S1, S2, S3, {S4, S5} descend by probability that each probability are 0.4, 0.2, 0.15, 0.25. => S1, {S4, S5}, S2, S3.

Add code in last two symbol that S2 add “0” and S3 add “1”.

So S1 = “” , S2 = “0”, S3 = “1”, S4 = “0”, S5 = “1”. Combine last two probability S2 and S3 to {S2, S3} and probability become 0.2 + 0.15 = 0.35.

Sort symbol S1, {S4, S5}, {S2, S3} descend by probability that each probability are 0.4, 0.25, 0.35,

=> S1, {S2, S3} , {S4, S5}. Add code in last two symbol that {S2, S3} add “0” and {S4, S5}, add “1”.

So S1 = “” , S2 = “00”, S3 = “01”, S4 = “10”, S5 = “11”. Combine last two probability {S2, S3} and {S4, S5} to { S2, S3, S4, S5} and probability become 0.25 + 0.35 = 0.6.

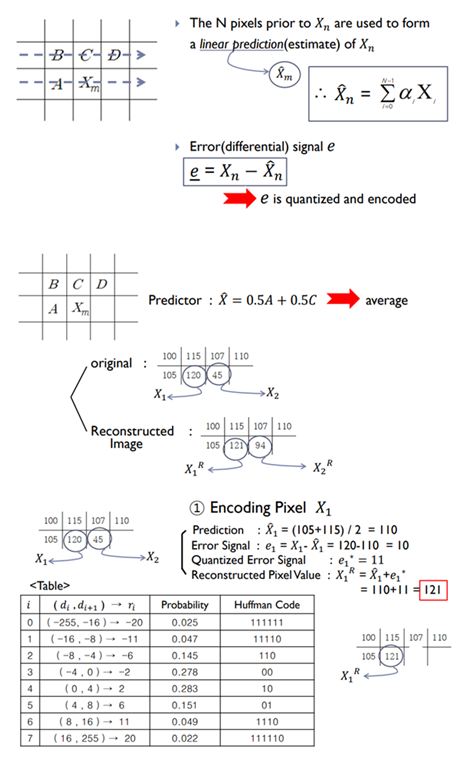
Sort symbol S1, { S2, S3, S4, S5} descend by probability that each probability are 0.4, 0.6,

=>{ S2, S3, S4, S5}, S1 Add code in last two symbol that { S2, S3, S4, S5} add “0” and S1 add “1”.

So S1 = “1” , S2 = “000”, S3 = “001”, S4 = “010”, S5 = “011”.

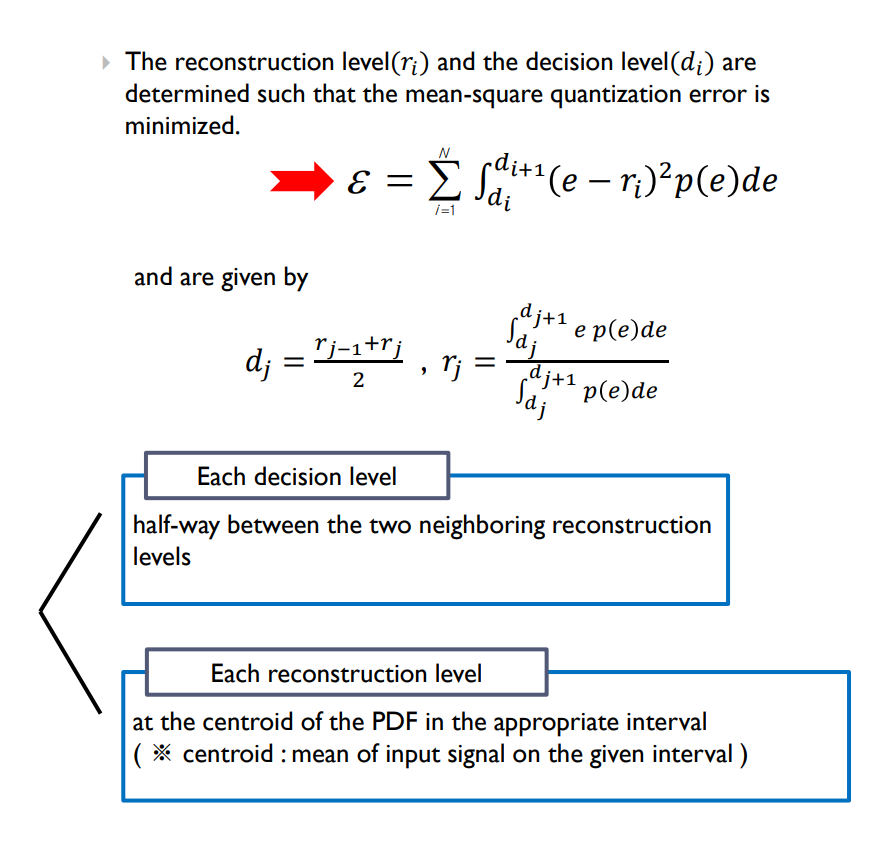
Finally you can encode S1 = “1” , S2 = “000”, S3 = “001”, S4 = “010”, S5 = “011”. Method could be little bit different but basic method is same, so result is same and I adopt this method and make function.

2. DPCM ( Differential Pulse-Code Modulation )



First calculate average of neighbor pixel. And calculate difference of original value and calculate. Make quantization error and encoding quantization error by Huffman code. When you reconstruct image, calculate neighbor pixel and add decoded error.

3. Lioyd-Max Non-uniform Quantization

****

Usually, quantization use uniform distribution. Because it is simple method. Just divide entire range equal and quantization by middle value of each divided range. However it ignore probability.

If probability focus on some range, that range of quantization error become high. So divide range consider probability not equal. Entire mean square error . d is range value that ith range is d(i) to d(i+1). r is reconstruct value that ith range become r(i). “Lioyd-Max Non-uniform Quantization” suggest how about set d and r that minimum value for entire mean square error. When satisfy dj = and , entire MSE become minimum that mean quantization is better than uniform.

Set initial value for decision level. And solve optimum r for given set of d by upper formula. Also calculate optimum d by calculated r. Repeat until values are convergence. This is method for quantization that use in DPCM.

**Process :**

In this project, I use picture of Lenna. And I convert 8 bits(256 levels) grey level image which size is 256 x 256.



< Fig 1 >

First apply DPCM. Remain first row and column, so g\*(0,j) = g(0,j) and g\*(i,0) = g(i,0). Now calculate g\*(i,j) = [ g\*(i-1,j) + g\*(i-1,j-1) + g\*(i,j-1) + g\*(i+1,j-1) ]. Consider g\*(i+1,j-1), calculate column direction not row direction. In last row, there is no g\*(i+1,j-1), so calculate

g\*(255,j) = [ g\*(254,j) + g\*(254,j-1) + g\*(255,j-1) ]. Get value ‘e’ by e = g – g\*.

Now let’s quantization ‘e’ by 64 quantized error e\*. I use “Lioyd-Max Non-uniform Quantization”. Then encode 64 quantized error.

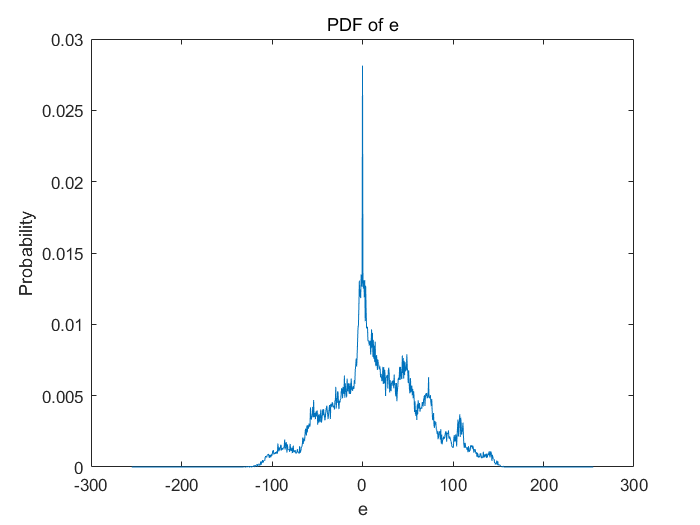
Reconstruct g, = g\* + e\*. (0,j) = g(0,j) and (i,0) = g(i,0) because I remain these values. Now calculate

(i,j) = [ g\*(i-1,j) + g\*(i-1,j-1) + g\*(i,j-1) + g\*(i+1,j-1) ] + e\*. Consider g\*(i+1,j-1), calculate column direction not row direction. In last row, there is no g\*(i+1,j-1), so calculate

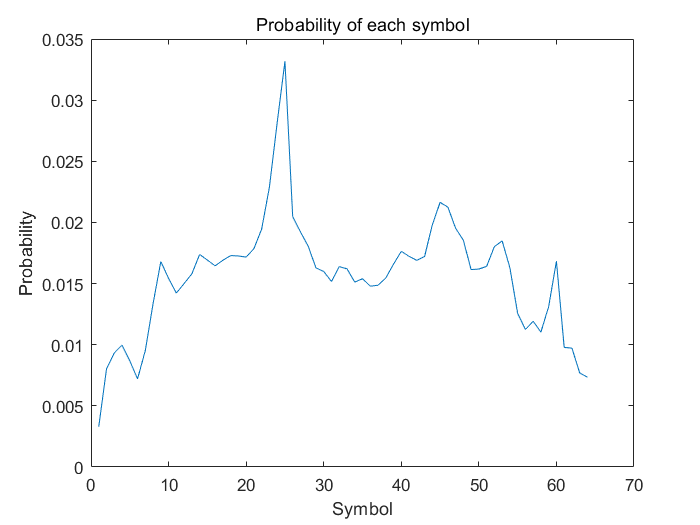
(255,j) = [ g\*(254,j) + g\*(254,j-1) + g\*(255,j-1) ] + e\*.

I apply this method by “Matlab” tool.

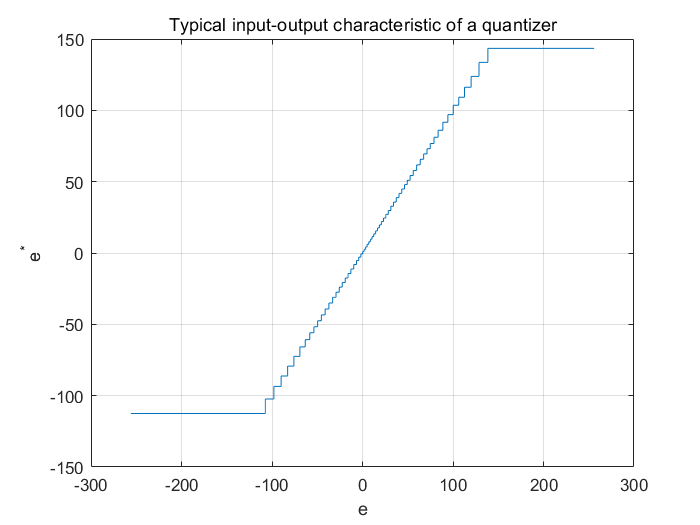
**Result :**



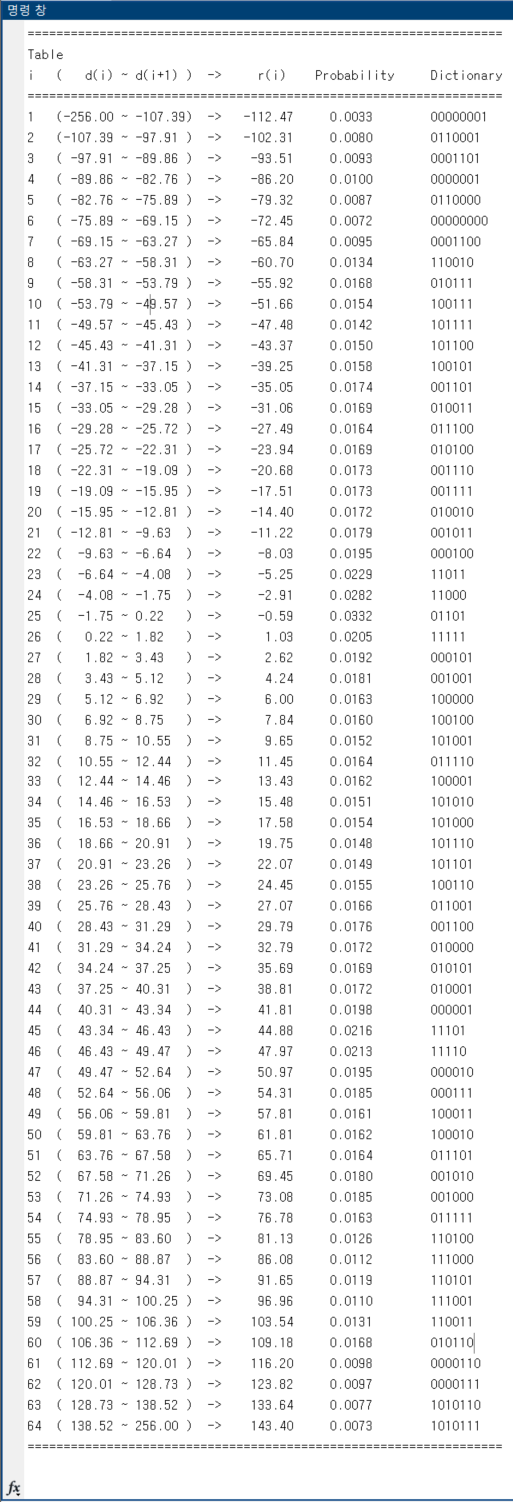
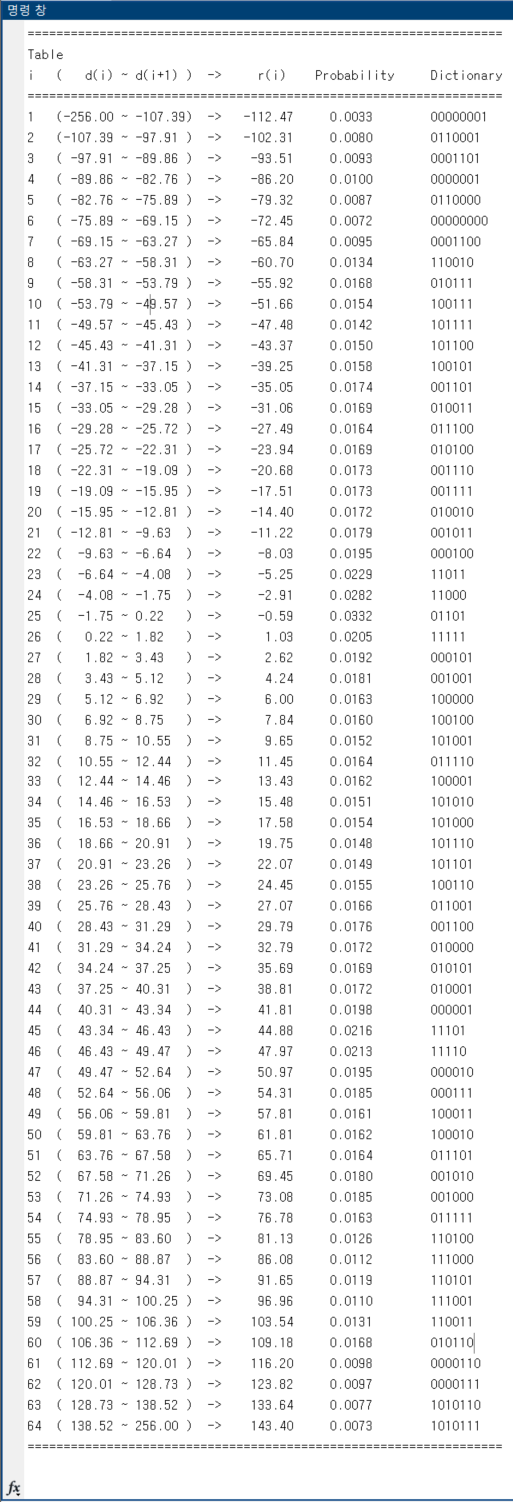
< Fig 2 >



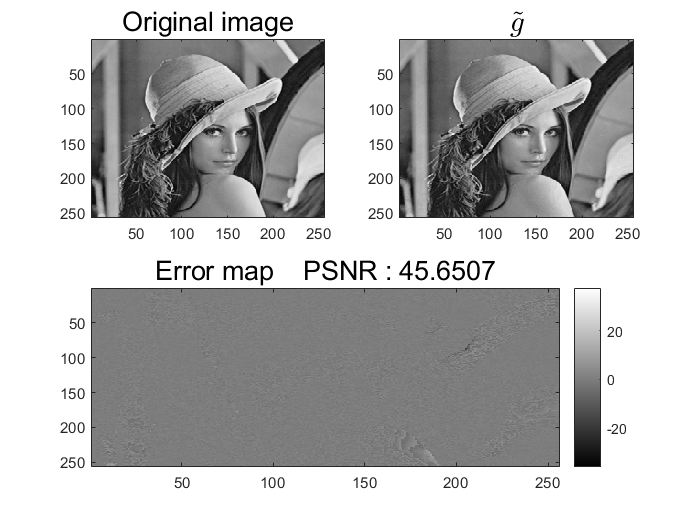
< Fig 3 >



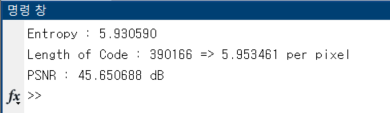
< Fig 4 >



< Fig 5 >



< Fig 6 >



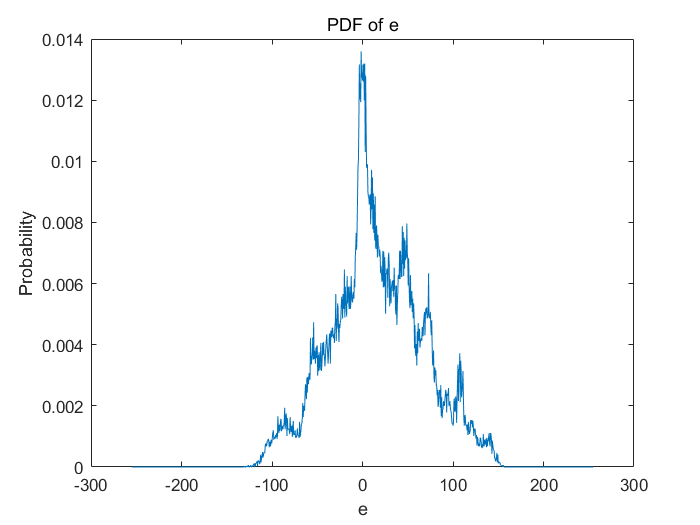
When you see < Fig 2 > and e = 0, you will feel discomfort. It because I remain first row and column, so there are a lot of e = 0. So it looks strange, but it looks like gaussian distribution. I use Lloyd-Max quantizer that non uniform quantizer. As you can see < Fig 4 >, the higher probability range, the more narrow quantization. So you can reduce error compare with uniform quantization. Code length of e was 390166. Entire pixel is 65536, so each pixel use only 5.95346069336 bits. It is under 6 bits, When you consider original image is 8bits in each pixel, you can save 2bits in each pixel. Original image is 8 bits(256 levels) grey level image which size is 256 x 256. That mean totally it needs 256 x 256 x 8 = 524288 bits. But code length is 390166 and this mean you need 390166 bits for reconstruct image. So it compress by .

Entropy calculate by . In information theory, you cannot compress average code bit under entropy. Huffman code is efficient encoding, almost compress to entropy bits.

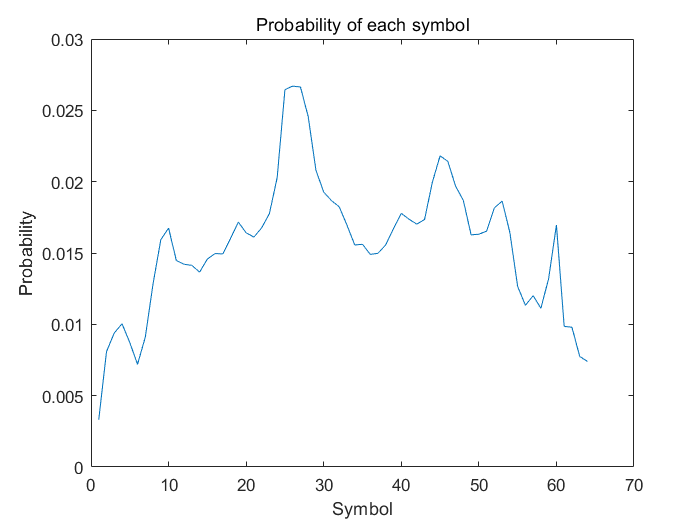
I calculate PSNR by 20log(Maximum value of original image) – 10log(MSE). PSNR is 45.6507, you can say it is fine to reconstruct image.

**Discussion :**

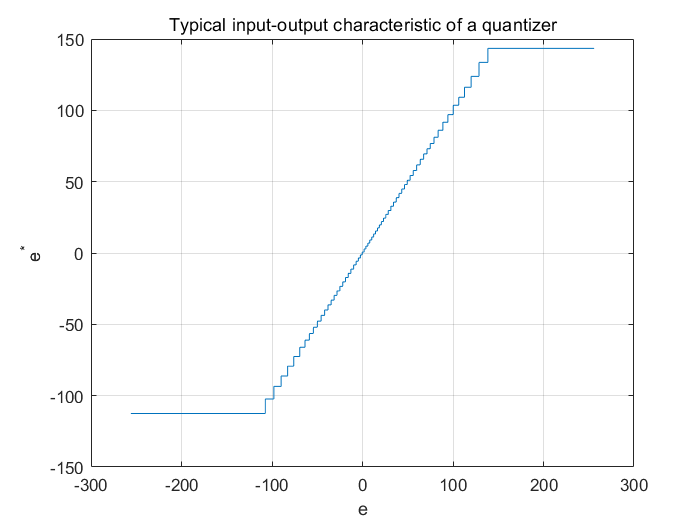
Let’s not consider first row and column of image when calculate e. It must be 0 because their value are not change.



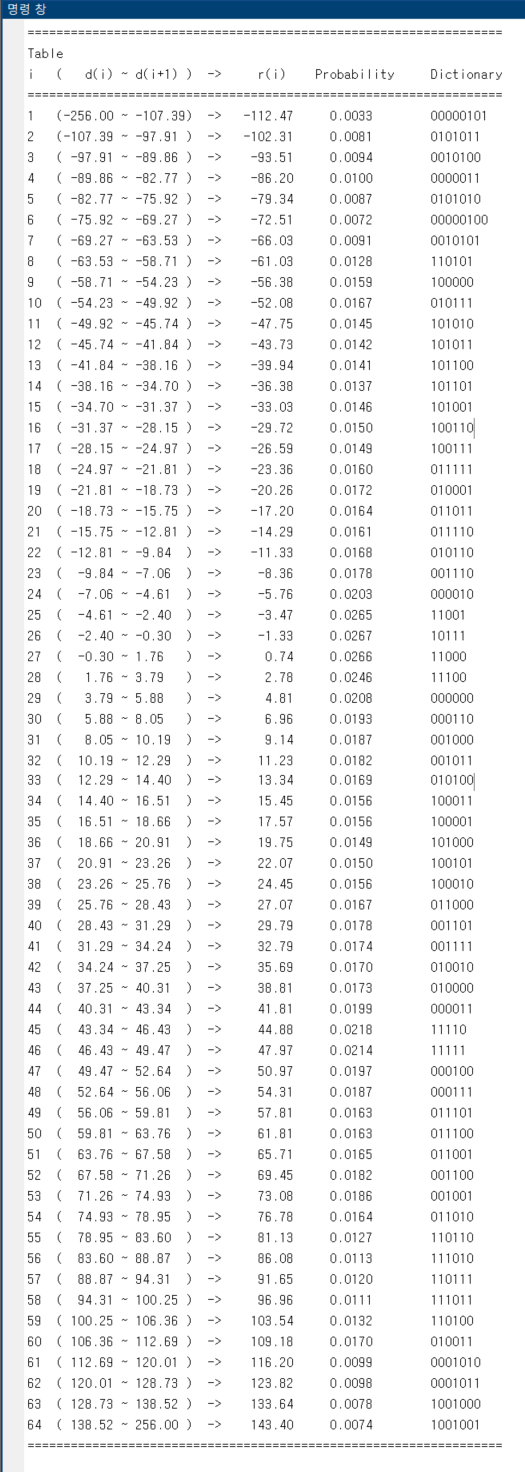
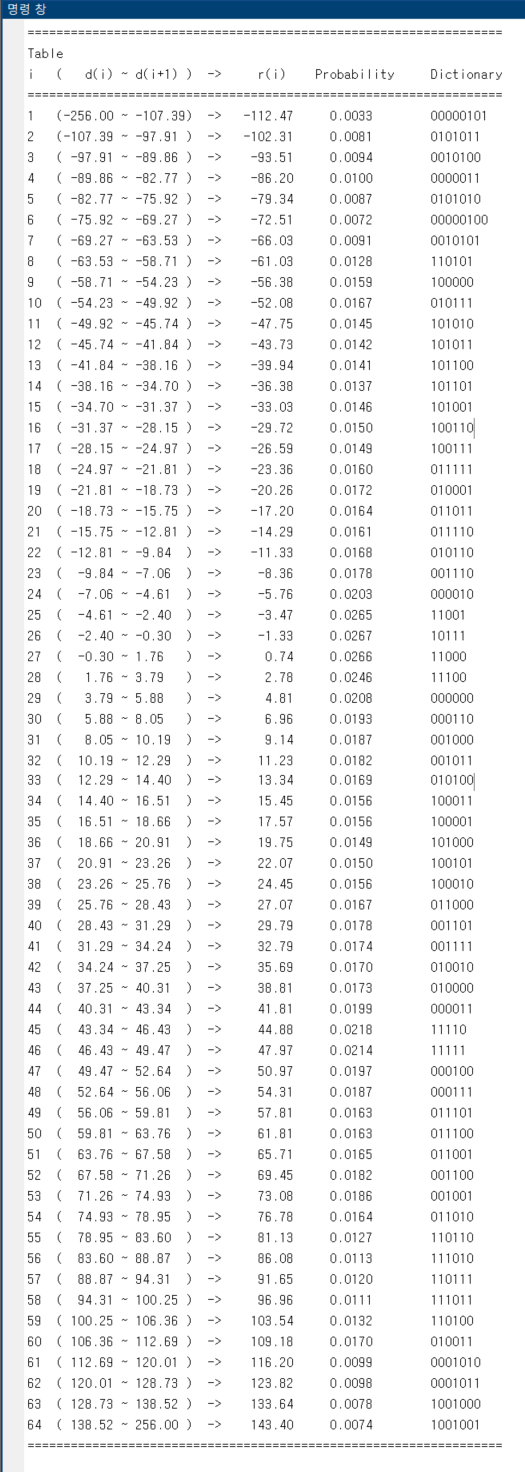
< Fig 7 >



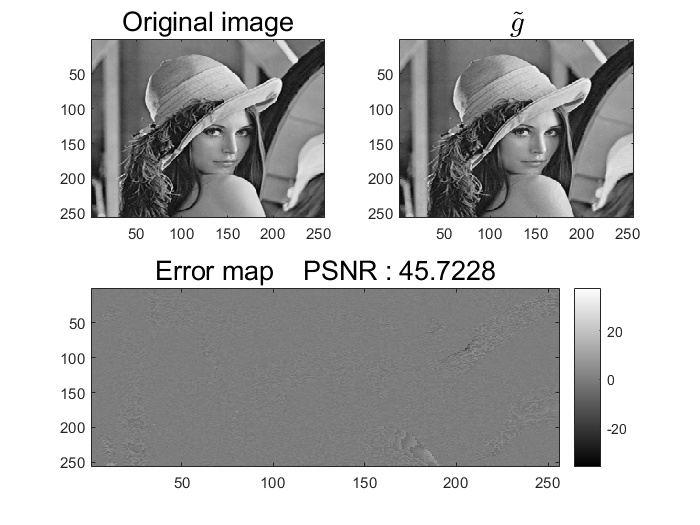
< Fig 8 >



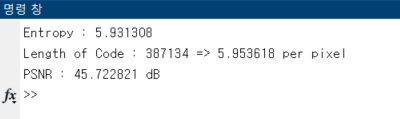
< Fig 9 >



< Fig 10 >

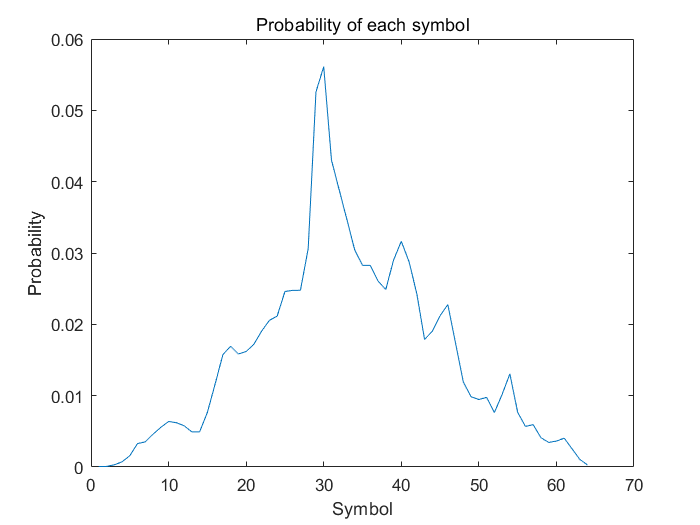


< Fig 11 >

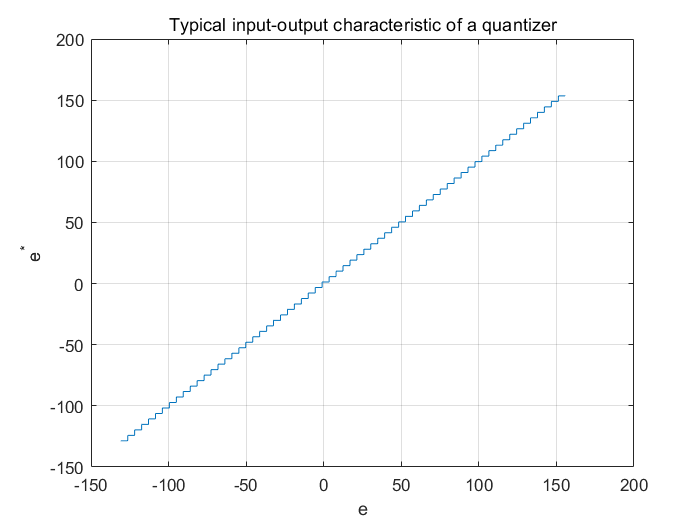


PDF is still strange, but it looks like gaussian distribution. Now code length is 387134. I don’t care about first row and column, pixels I use are 65025. Each pixel uses 5.95361783929, you can save 2bits per pixel. So you can image compress by DPCM. I calculate PSNR without first row and column. PSNR become better because I remove unnecessary data that first row and column. I think this is more reasonable method than previous one.

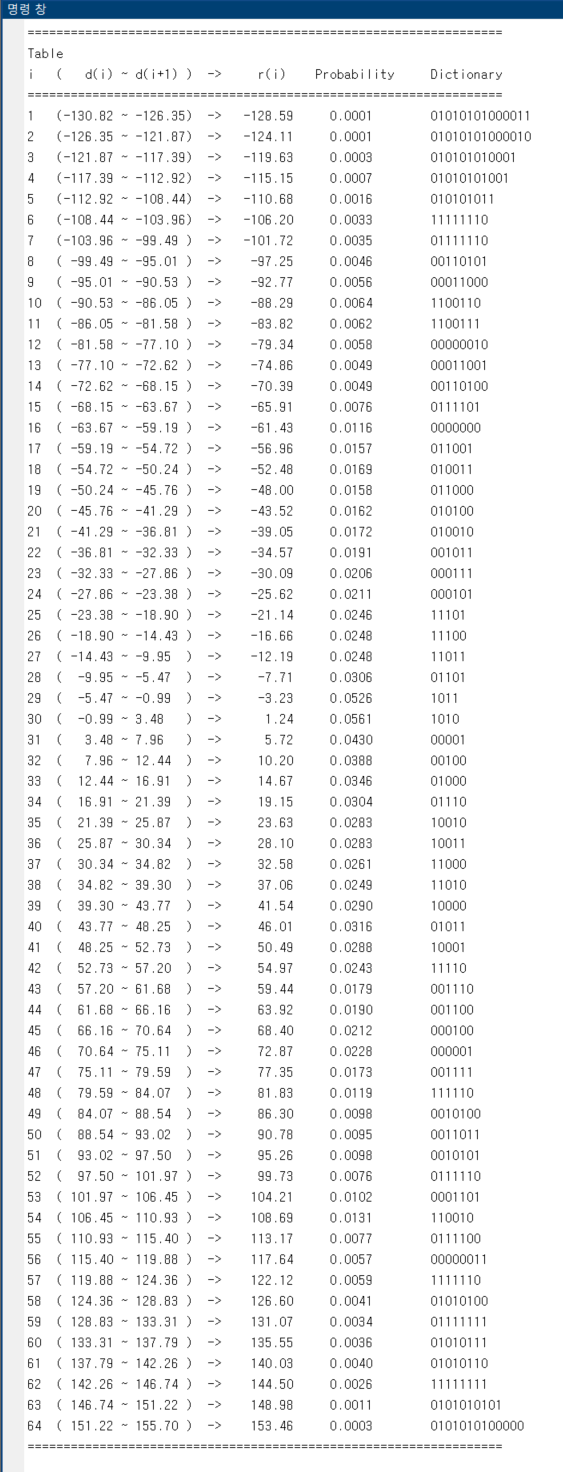
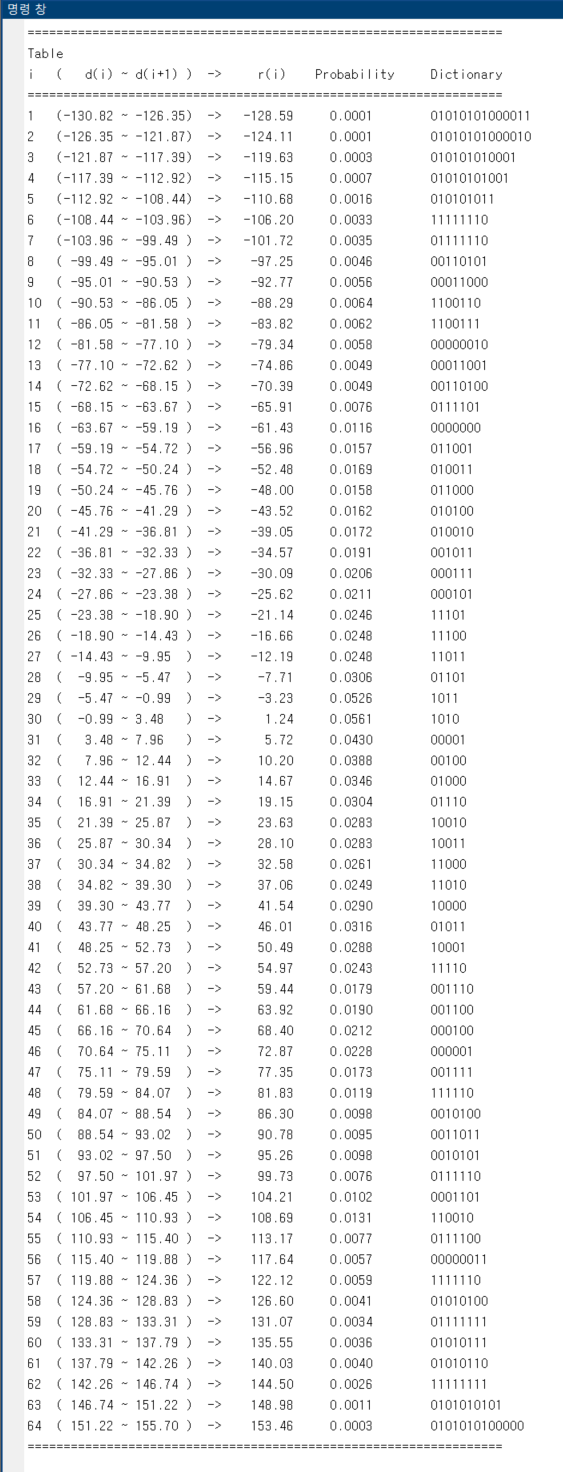
Now let’s see uniform quantization, not “Lioyd-Max Non-uniform Quantization”.



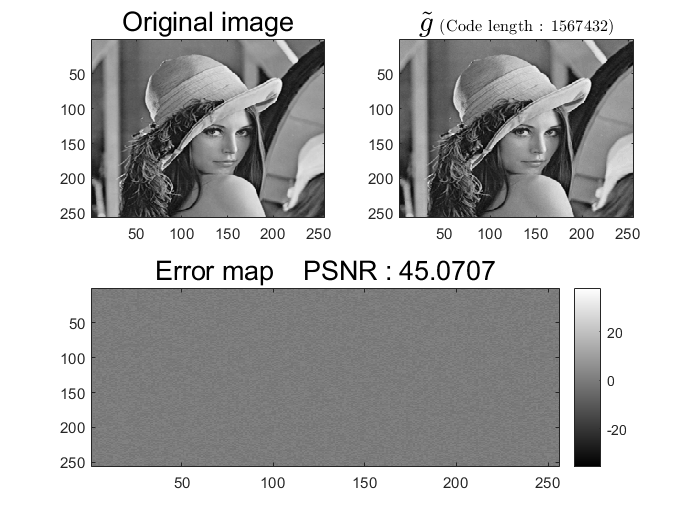
< Fig 12 >



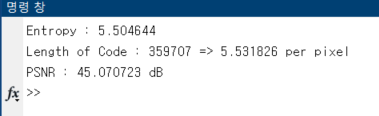
< Fig 13 >



< Fig 14 >



< Fig 15 >



I consider total range minimum and maximum value of image. I think -255 to 255 is too unreasonable because probability of near -255 or 255 value is surely 0. As you see < Fig 13 >, it is evenly quantized. Entropy and length of code become shorter. So it looks more efficient. But when you see PNSR, it decreases. As I mention, uniform quantization is not consider probability so some range could lots of error. However it has more focus probability, so it can compress more than “Lioyd-Max Non-uniform Quantization”. So if you want to compress more, use uniform quantization. But if you want to consider error too, use “Lioyd-Max Non-uniform Quantization”.

In ideal case, when you apply Lloyd-Max quantizer, you can get almost same probability in each range. But I cannot get PDF because it is not infinite number of values, and PDF function is not continuous. Also it has convergence problem such as saddle points or local minimum. But anyway it can be comparable with uniform quantization.

**Reference :**

Chapter4 of TextbookFile

<https://kr.mathworks.com/help/matlab/ref/hist.html>

https://ko.wikipedia.org/wiki/최대\_신호\_대\_잡음비

https://ko.wikipedia.org/wiki/레나\_ (이미지)