MATLAB을 이용한

디지털 영상처리의 기초

14.1 무손실 및 손실 압축

우리는 영상의 파일들이 매우 크게 된다는 사실을 알고 있다. 그래서 파일을 저장하거나 전송하기 위해 가능하면 파일의 사이즈를 작게하는 것이 중요하다. 제 1장에서 압축의 경향에 대한 간략한 설명을 하였고, 이 절에서 몇 가지의 표준 압축방법을 알아보기로 한다. 2가지의 압축방법의 차이를 구별할 필요가 있는데, 무손실 압축은 모든 정보가 유지되는 방식이며, 손실 압축은 약간의 정보가 제거되는 방식이다. 200–2

손실 압축: VQ, JPEG
200 198 199 200 -> 200 –2 1 1 로 coding
무손실 압축: Huffman, DPCM(Differential Pulse coded Modulation)
RLE (Run Length Encoding)

14.2 허프만 부호화

허프만부호화의 개념은 간단하다. 영상에서 그레이 값들을 표현하는데 고정길이코드(8비트)를 사용하지 않고 가변길이코드를 사용하는데, 영상 내에서 그레이 값들이 확률적으로 자주 나오는 값일수록 보다 짧은 코드를 사용하는 것이다.

예를 들면 4가지의 그레이 값들, 0, 1, 2 및 3을 가지는 2비트 그레이스케일 영상을 가정하고, 그 값들의 출현 확률이 각각 0.2, 0.4, 0.3 및 0.1이라 한다. 즉 영상 내에서 화소의 20%는 그레이 값 50을 가지고, 40%는 그레이 값 100을 가지는 등이다. 아래의 표는 영상에서 고정길이와 가변길이 코드를 보였다.

Gray value	Probability	Fixed code	Variable code
0	0.2	00	000
1	0.4 빈도↑	01	1 코드길이↓
2	0.3	10	01
3	0.1	11	001

이 영상이 어떻게 압축되었는지 살펴보자. 각 그레이 값은 그 자신의 유일한 코드를 가진다. 화소 당 평균 비트수는 기대값으로 쉽게 아래와 같이 계산될 수 있다(확률적인 방법).

fixed code 로는 2비트 필요,

$$(0.2 \times 3) + \underline{(0.4 \times 1)} + (0.3 \times 2) + (0.1 \times 3) = 1.9.$$

1.9비트 필요 good

여기서 가장 긴 코드워드는 가장 낮은 확률을 가지는 것을 알 수 있다. 이 평균은 실제로 2보다 더 작다.

이것은 엔트로피(entropy)의 개념으로 더 정밀하게 만들 수 있는데, 이는 정보의 양을 측정하는 것이다. 특히 영상의 엔트로피 H는 정보의 손실이 없이 영상을 부호화하는데 요구되는 화소 당 이론적 최소의 비트수이다. 이것은 아래와 같이 정의된다.

$$H = -\sum_{i=0}^{L-1} p_i \log_2(p_i),$$

여기서 인덱스 i는 영상의 그레이스케일 이고, p_i는 영상에서 발생되는 그레이 레벨 i의 확률이다. 위의 예에서, 엔트로피는 아래와 같다.

$$H = -(0.2\log_2(0.2) + 0.4\log_2(0.4) + 0.3\log_2(0.3) + 0.1\log_2(0.1)) = 1.8464.$$

비록 이 값이 부호화에 사용된다 하더라도, 화소 당 1.8464비트 이하로는 결코 사용할 수 없다는 의미이다. 이러한 기초에서, 위의 허프만부호화의 구조는 화소 당 평균 비트 수가 이 이론적인 최소값에 근접하는 2가 매우 좋은 결과를 줄 수 있다.

주어진 영상에 대하여 허프만부호를 구하기 위해 아래와 같은 과정으로 진행한다.

1. 영상에서 각 그레이 값의 확률을 구한다.
2. 2진 트리로부터 가장 낮은 확률을 취하여 더한다.
3. 그 꼭지 점에서 트리의 각 가지에 임의로 0과 1을 할당한다.
4. 위에서 아래로 부호(코드)를 읽는다.

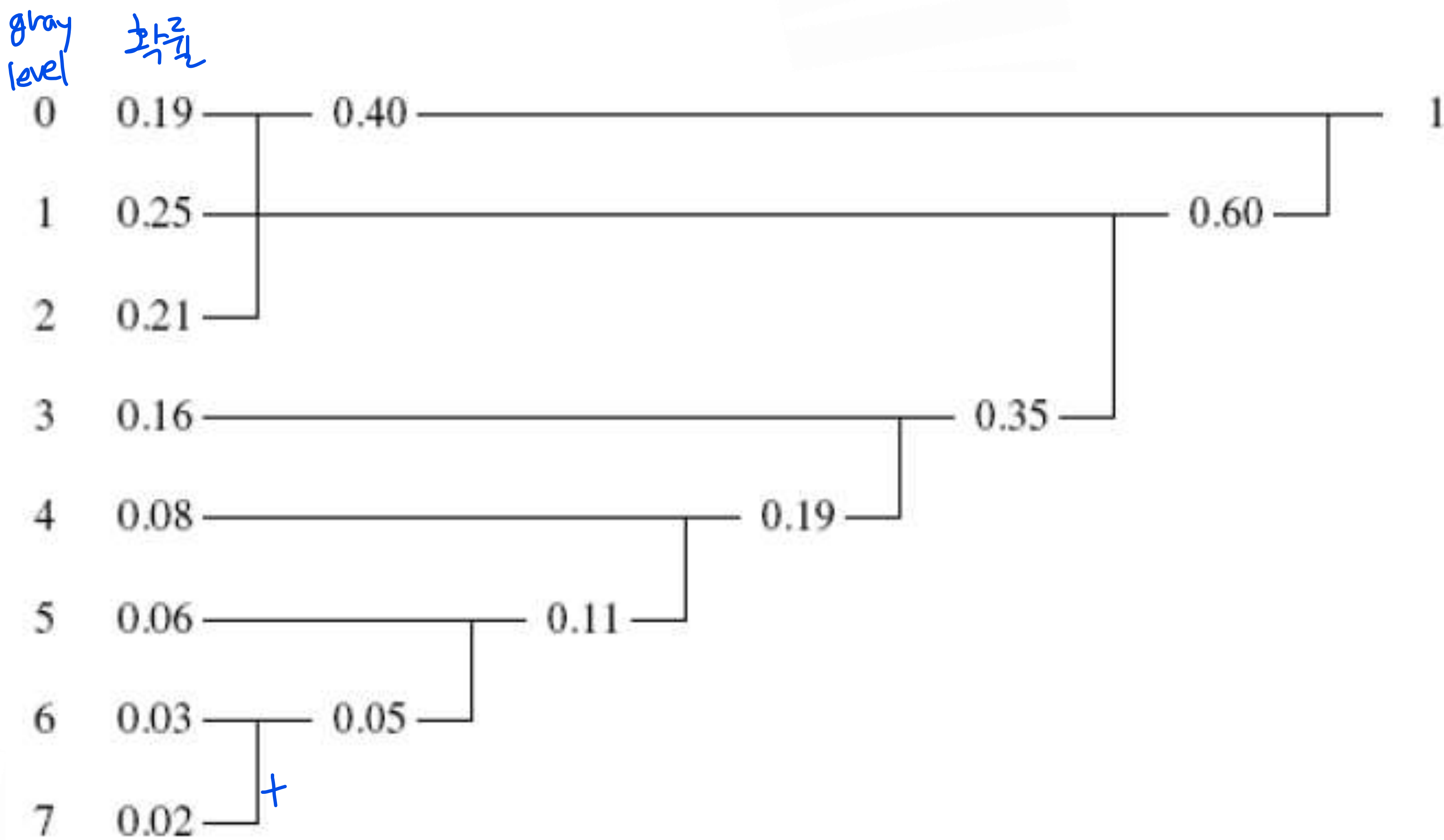


그림 14.1 허프만 부호의 형성

확률 작은 것 2개끼리 계속 더해 트리 생성

이 과정을 보기 위해 아래의 확률을 가지는 3비트(0-7의 그레이 값) 그레이스케일 영상을 생각하자

gray value	0	1	2	3	4	5	6	7
probability	0.19	0.25	0.21	0.16	0.08	0.06	0.03	0.02

이들 확률에 대하여 엔트로피는 2.6508로 계산될 수 있다. 그림 14.1과 같이 한번에 2개의 확률을 결합할 수 있다. 이때 우리는 임의로 확률을 선택할 수 있다. 2번째 단계는 방금 얻어진 트리의 각 가지에 0과 1을 임의로 할당한다. 그림 14.2에 이를 보였다.

$$H = (0.19log_2(0.19) + 0.25log_2(0.25) \ldots\ldots .. 0.02log_2(0.02)) = \underbrace{2.6508}_{\text{엔트로피}}$$

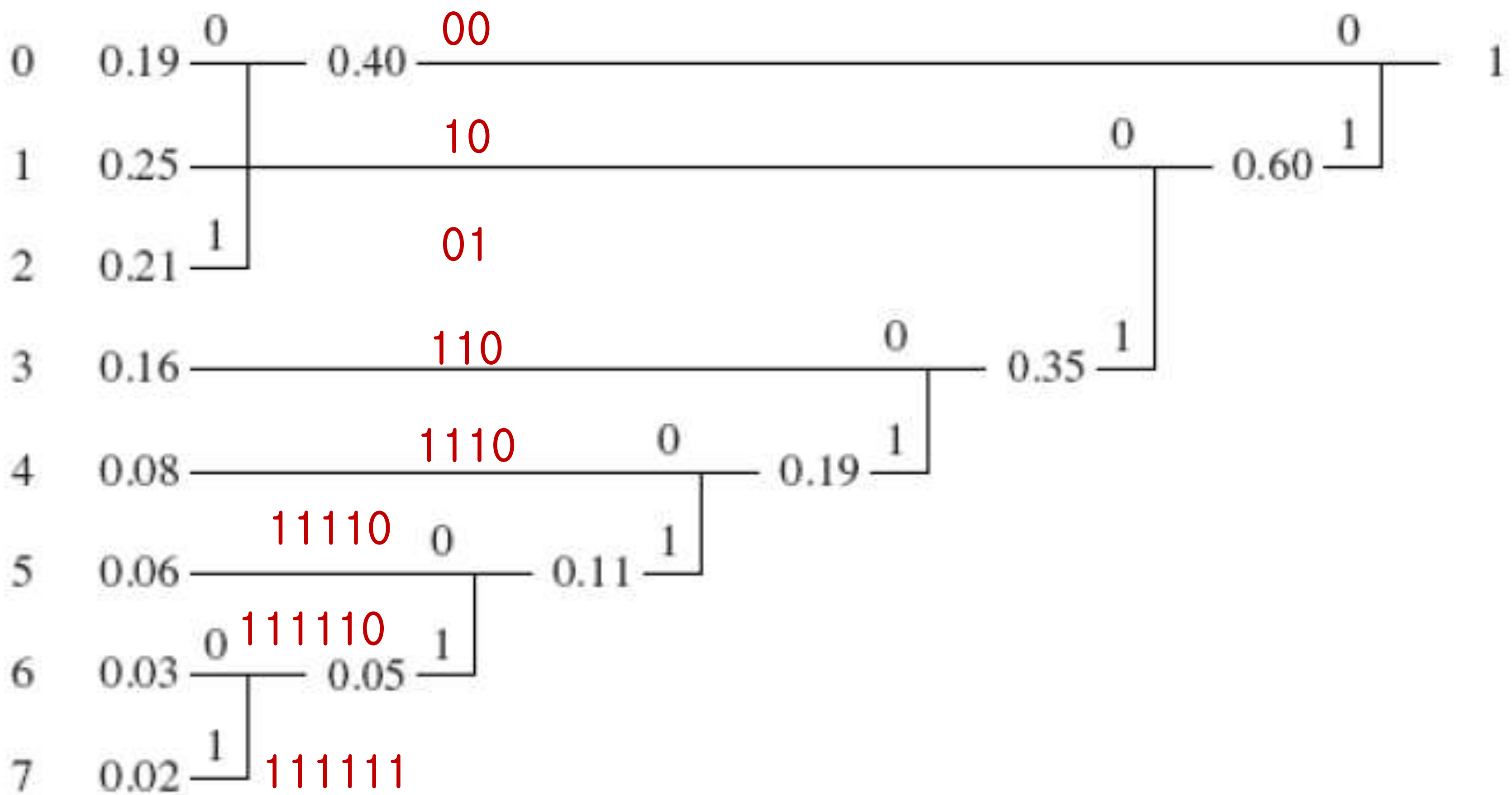


그림 14.2 각 가지의 0과 1의 할당

각 그레이 값에 대한 부호를 얻기 위해 오른쪽 위의 1에서 시작하여 왼쪽 아래로 되돌아오면서 지나는 0 혹은 1의 부호를 읽는다. 이 과정은 아래와 같다.

	Gray value	Huffman code
0.19	0	00
0.25	1	10
0.21	2	01
0.16	3	110
0.08	4	1110
0.06	5	11110
0.03	6	111110
0.02	7	111111

고정비트/ 압축비트: 3/2.7 = 1.11

압축비: 1.11 : 1

위와 같이, 기대값으로 화소 당 평균 비트수를 아래와 같이 계산할 수 있다.

$$(0.19 \times 2) + (0.25 \times 2) + (0.21 \times 2) + (0.16 \times 3)$$
$$+ (0.08 \times 4) + (0.06 \times 5) + (0.03 \times 6) + (0.02 \times 6) = 2.7,$$

이것은 화소 당 3비트 이상의 현저한 개선이 있고, 엔트로피에 의해서 2.6508의 이론적 최소값에 매우 근접한다.

허프만부호화는 유일하게 복호될 수 있고, 일방통행으로 문자열이 복호될 수 있다. 예를 들면 아래와 같은 수열을 고려해보자

1 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 인코딩

위에서 만들어진 허프만부호로 복호할 수 있다. 부호워드 1 혹은 11은 없고, 첫 3비트는 그레이 값이 3으로서 110이다. 이 수열로 시작하는 부호워드가 없음을 주의하라. 그 다음 몇 개의 비트에 대하여 1110은 하나의 부호워드이며 이 수열로 시작하는 다른 것은 없고 더 작은 수열의 부호워드도 없다. 따라서 이 수열을 그레이레벨 4로서 복호할 수 있다. 이 방법을 계속하여 아래와 같이 복호할 수 있다.

디코딩

$$\underbrace{1\ 1\ 0}_{3}\ \underbrace{1\ 1\ 1\ 0}_{4}\ \underbrace{0\ 0}_{0}\ \underbrace{0\ 0}_{0}\ \underbrace{1\ 0}_{1}\ \underbrace{0\ 1}_{2}\ \underbrace{1\ 1\ 1\ 1\ 0}_{5}$$

어떤 코드도 다른 코드의 접두사가 될 수 없다.

14.3 줄길이 부호화(Run-length coding)

줄길이부호화(RLE)는 간단한 아이디어에 기초한다. 각 수열에서 반복되는 0과 1을 부호화하는 것이다. RLE는 팩스전송에서 표준으로 사용되고 있다. 2진 영상에 대하여 여러 가지 다른 RLE의 구현방법이 있다. 한 가지 방법은 0의 수로 시작하면서 각 라인을 분리하여 부호화한다. 아래의 영상을 부호화한다고 생각하자.

0	1	1	0	0	0
0	0	1	1	1	0
1	1	1	0	0	1
0	1	1	1	1	0
0	0	0	1	1	1
1	0	0	0	1	1

이는 아래와 같이 부호화된다. 1로 시작하면 0 추가

(123)(231)(0321)(141)(33)(0132)

또 다른 방법은 수의 묶음을 각 행으로 부호화하는데, 각 묶음에서 첫째의 수는 1의 위치이고 2번째 수는 그 1의 길이이다. 따라서 위의 2진 영상은 아래와 같이 부호화된다.

1의 2번째부터 2번나옴

(22)(33)(1361)(24)(43)(1152)

그레이스케일 영상은 제 3장에서 설명한 비트평면으로 분리하여 부호화할 수 있다. 간단한 예를 들면 다음의 4비트 영상을 고려하고, 2진 표현을 하면 아래와 같다.

10	7	8	9		1010	0111	1000	1001
11	8	7	6	→	1011	1000	0111	0110
9	7	5	4		1001	0111	0101	0100
10	11	2	1		1010	1011	0010	0001

이를 비트평면으로 분리하면 아래와 같다.

0	1	0	1	1	1	0	0	0	1	0	0	1	0	1	1
1	0	1	0	1	0	1	1	0	0	1	1	1	1	0	0
1	1	1	0	0	1	0	0	0	1	1	1	1	0	0	0
0	1	0	1	1	1	1	0	0	0	0	0	1	1	0	0
0th plane				1st plane				2nd plane				3rd plane			

이들 각 평면은 선택한 RLE의 구현방법으로 분리하여 부호화할 수 있다.

그러나 비트평면에는 문제점이 있는데, 이는 그레이 값의 작은 변화가 비트에서 큰 변화를 일으킬 수 있다. 예를 들면 7에서 8로의 변화는 2진 수열 0111에서 1000로 변화를 의미하므로 4비트 전체의 변화의 원인이 된다. 당연히 이 문제는 8비트 영상을 악화시킨다. RLE를 효과적으로 하기 위하여 매우 비슷한 그레이 값들의 길이가 길면 압축률이 높은 부호화를 기대할 수 있다. 그러나 이런 경우만 있는 것이 아니다. 불규칙하게 7과 8의 값으로 구성되는 4비트 영상 비트평면에 상관이 적고, 효과적인 압축이 되지 못한다.

이 난점을 극복하기 위하여 2진 **그레이부호**(Gray codes)로 그레이 값들을 부호화할 수 있다. 그레이부호는 하나의 수열과 다음 수열 사이에 단지 하나의 비트만 변화하도록 주어진 길이의 모든 2진 수열을 순서화하는 것이다. 따라서 그레이부호는 아래와 같다.

15	1	0	0	0
14	1	0	0	1
13	1	0	1	1
12	1	0	1	0
11	1	1	1	0
10	1	1	1	1
9	1	1	1	0
8	1	1	0	0
7	0	1	0	0
6	0	1	0	1
5	0	1	1	1
4	0	1	1	0
3	0	0	1	0
2	0	0	1	1
1	0	0	0	1
0	0	0	0	0

이것의 장점을 보기 위해 다음과 같이 4비트 영상의 2진부호와 그레이부호를 생각하자.

$$
\begin{array}{cccc} 8 & 8 & 7 & 8 \\ 8 & 7 & 8 & 7 \\ 7 & 7 & 8 & 7 \\ 7 & 8 & 7 & 7 \end{array}
\longrightarrow
\begin{array}{cccc} 1000 & 1000 & 0111 & 1000 \\ 1000 & 0111 & 1000 & 0111 \\ 0111 & 0111 & 1000 & 0111 \\ 0111 & 1000 & 0111 & 0111 \end{array}
\quad
\begin{array}{cccc} 1100 & 1100 & 0100 & 1100 \\ 1100 & 0100 & 1100 & 0100 \\ 0100 & 0100 & 1100 & 0100 \\ 0100 & 1100 & 0100 & 0100 \end{array}
$$

여기서 첫 2진 배열은 표준 2진부호화란 것이고, 2번째 열은 그레이부호로 부호화한 것이다. 이의 2진 비트평면은 아래와 같다.

$$
\underset{\text{0th plane}}{\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{array}}
\quad
\underset{\text{1st plane}}{\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{array}}
\quad
\underset{\text{2nd plane}}{\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{array}}
\quad
\underset{\text{3rd plane}}{\begin{array}{cccc} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{array}}
$$

그레이부호에 대응하는 부호는 아래와 같다.

0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1
0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	0
0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0
0th plane				1st plane				2nd plane				3rd plane			

그레이부호평면은 하나의 비트평면을 제외하고는 높은 상관을 가진다. 여기서 모든 2진 비트평면은 무상관을 가진다.

14.4 JPEG 알고리즘

손실압축은 압축률을 높이기 위해 데이터 일부의 손실을 허용한다. 가능한 많은 압축 방법 중에서 Joint Photographic Expert Group(JPEG)에서 개발한 알고리즘이 가장 널리 사용되고 있다. 이것은 영상의 화소 그 자체가 부호화되지 않고 **변환부호화** (transform coding)를 이용한다.

이 알고리즘의 핵심은 Discrete Cosine Transform(DCT) 이다. 어떤 사이즈의 배열도 적용할 수 있지만, JPEG 알고리즘은 오로지 8X8 블록을 적용한다. 만일 f(i,j)가 하나의 블록이면, 순방향 2차원 DCT의 정의는 아래와 같다.

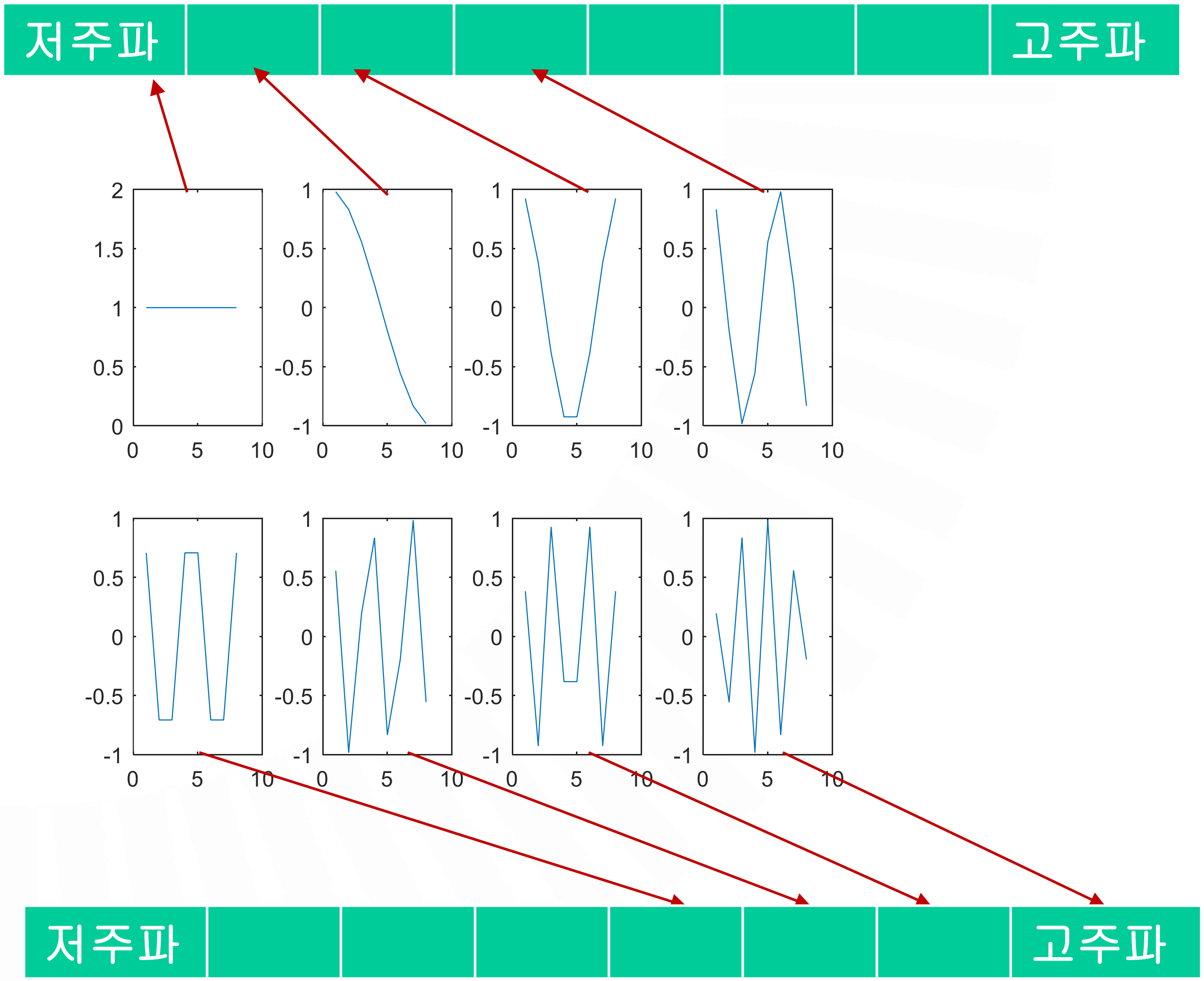
$$F(u,v)=\frac{C(u)C(v)}{4}\sum_{j=0}^{7}\sum_{k=0}^{7}f(j,k)\cos\left(\frac{(2j+1)u\pi}{16}\right)\cos\left(\frac{(2k+1)v\pi}{16}\right)$$

이의 역방향 변환 IDCT의 정의는 아래와 같이 표현된다.

$$f(i,j)=\sum_{u=0}^{7}\sum_{v=0}^{7}f(u,v)C(u)c(v)\cos\left(\frac{(2j+1)u\pi}{16}\right)\cos\left(\frac{(2k+1)v\pi}{16}\right)$$

- for u=0:7
-     for v=0:7
-         a(v+1)=v+1;
-         b(v+1)=cos(((2*v+1)*u*pi)/16);
-     end
-     subplot(2,4,u+1),plot(a,b);
- end

DCT Domain



저주파							
							고주파

여기서 C(w)는 아래와 같이 정의된다.

$$C(w) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } w = 0 \\ 0 & \text{otherwise.} \end{cases}$$

DCT는 특히 압축에 적합한 여러 가지 성질을 가지는데 이는 아래와 같다.

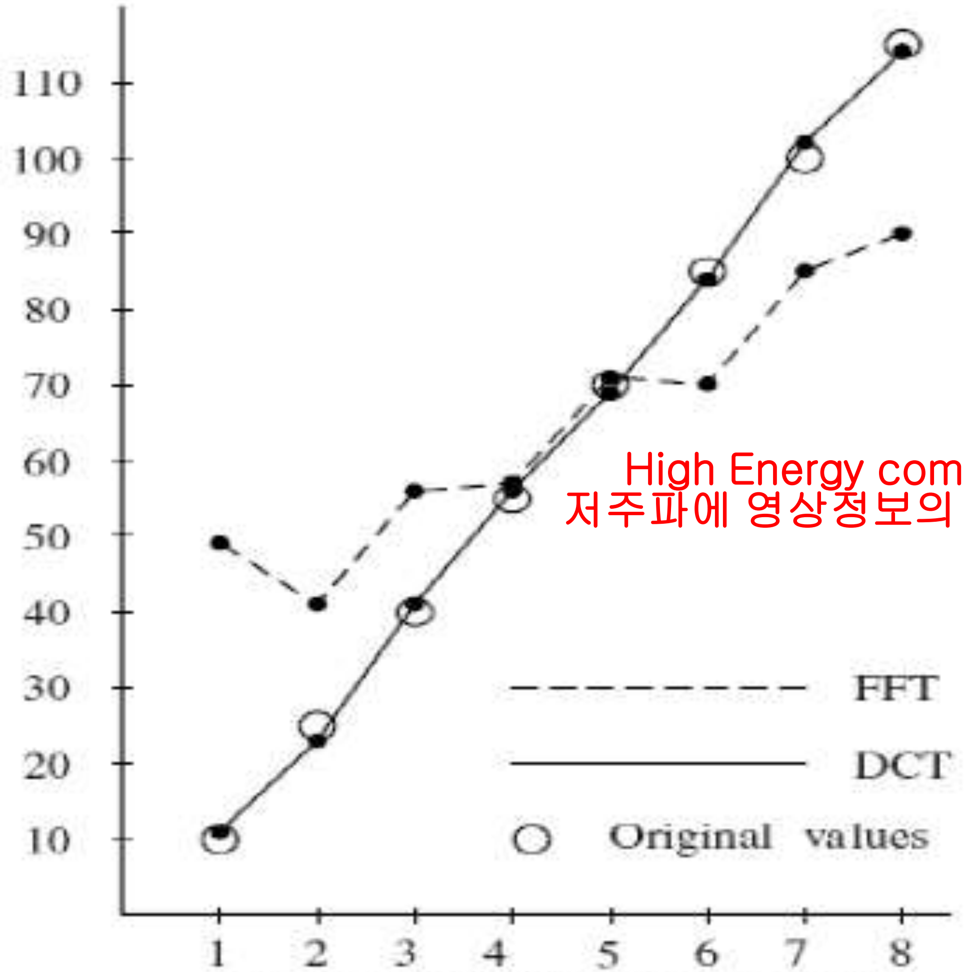
1. 복잡한 복소수를 필요로 하지 않고 실수의 값을 가진다.
2. 이는 작은 수의 계수로 많은 정보량을 묶을 수 있기 때문에 높은 정보의 묶음을 가진다. 저주파의 인식 ↑
3. 하드웨어 구현에 매우 효과적이다.
4. FFT와 같이 최대의 효율을 가지는 변환이 가능하다.
5. 기저값(basis values)은 데이터에 무관하다.

2차원 DCT는 분리 가능하고, 1차원 DCT의 수열로 계산할 수 있다. 먼저, 가로방향으로 1차원 DCT를 적용한 후에 그 결과를 세로방향으로 변환할 수 있다.

정보의 묶음에 대한 능력을 보기 위해 예를 들어 보면, 아래와 같이 간단한 선형 수열을 고려한다.

```
>> a=[10:15:115]

a =

    10    25    40    55    70    85   100   115
```

```
>> da=dct(a);
>> da(5:8)=0;
>> round(idct(da))

ans =

    11    23    41    56    69    84   102   114
```

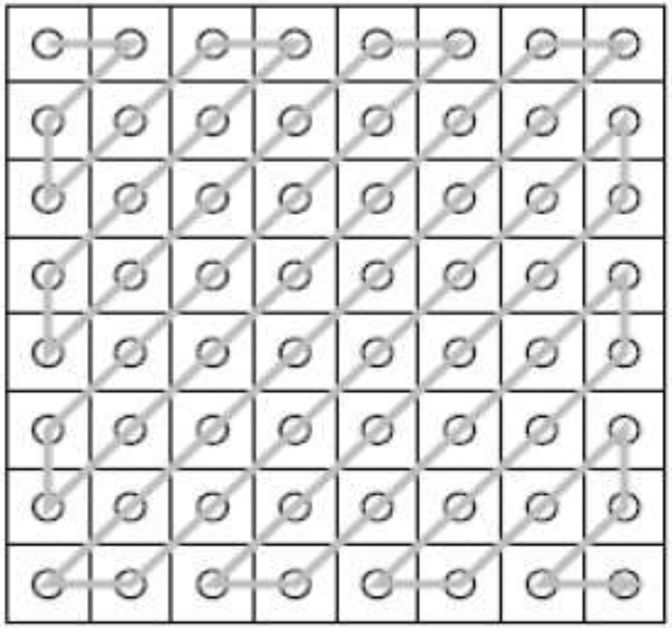


High Energy compaction property
저주파에 영상정보의 대부분이 모여 있다.

그림 14.3 FFT와 DCT의 비교

JPEG 의 기본적 압축구조는 다음과 같이 적용된다.

1. 영상을 8×8 블록으로 나누고 각 블록을 변환하며 압축은 블록단위로 이루어진다.
2. 주어진 블록에 대하여, 해당 값들은 각 값에서 128을 빼서 시프트된다.
3. DCT는 이 시프트된 블록에서 적용된다.
4. DCT 값들을 정규화 매트릭스 Q로 나누어서 정규화된다. 이 정규화 값들은 해당 블록 내 대부분의 요소들이 0이 되어서 압축이 된다.
5. 이 매트릭스는 아래의 그림과 같이 지그재그 순서로 왼쪽 위에서 모두 0이 아닌 값들을 벡터로 간주한다.



6. 각 벡터의 첫째 계수는 각 벡터에서 가장 큰 값이고, 이는 DC계수이며, 각 값과 이전 블록에서의 값의 차이를 리스트에 의해 부호화된다. 이것은 모든 값들(첫 번째 블록 제외)은 작은 값을 유지한다.
7. 이들 값들은 RLE를 이용하여 다시 압축된다.
8. 모든 나머지 값들(AC 계수들)은 허프만부호화를 이용하여 압축된다.

손실압축의 정보량은 위 단계 4의 정규화 매트릭스 Q의 스케일링에 따라 변화될 수 있다. 압축을 풀기 위해 위의 단계들을 역으로 적용하는데, 허프만부호화와 RLE는 정보의 손실이 없이 복원될 수 있다. 복원하는 과정에서 다음과 같은 처리가 실행된다.

1. 벡터는 8X8 매트릭스로 돌아가서 읽혀진다.
2. 이 매트릭스는 정규화 매트릭스와 곱해진다.
3. 이 결과에 역 DCT가 적용된다.
4. 이 결과는 원래의 영상을 얻기 위해 128까지 되돌려서 시프트된다.

JPEG 그룹에서 사용되어온 정규화 매트릭스는 아래와 같다.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \times \text{압축비}$$

우리는 MATLAB의 함수를 이용하여 DCT와 양자화의 실험을 할 수 있다. 2차원 DCT와 역 DCT는 각각 함수 dct2와 idc2로 구현될 수 있다.

```
>> c=imread('caribou.tif');
>> x=151;y=90;
>> block=c(x:x+7,y:y+7)

block =

    87    95    92    73    59    57    57    55
    74    71    68    59    54    54    51    57
    64    58    57    55    58    65    66    65
    57    63    68    66    74    89    98   104
    95   109   117   114   119   134   145   140
   128   139   146   139   140   148   151   143
   137   135   125   118   137   156   154   132
   122   119   113   110   128   144   140   142
```

여기서 이 블록의 각 값에서 128을 아래와 같이 뺀다.

```
>> b=double(block)-128

b =

   -41   -33   -36   -55   -69   -71   -71   -73
   -54   -57   -60   -69   -74   -74   -77   -71
   -64   -70   -71   -73   -70   -63   -62   -63
   -71   -65   -60   -62   -54   -39   -30   -24
   -33   -19   -11   -14    -9     6    17    12
     0    11    18    11    12    20    23    15
     9     7    -3   -10     9    28    26     4
    -6    -9   -15   -18     0    16    12    14
```

이를 아래와 같이 DCT 처리한다.

2차원

```
>> bd=dct2(b)

bd =

 -225.3750  -30.7580   17.3864    5.6543  -22.3750   -1.8591    3.7575    1.7196
 -241.5333   52.0722    0.8745  -21.2434    8.1434    1.8639    0.9420   -1.3369
   -2.5427   50.9316    5.0847    9.1571    1.5820   -3.8454    1.5706   -0.6043
  102.5557   23.3927  -11.5151  -12.7655  -10.6629    2.8179   -3.6743    1.2462
   -2.3750  -20.7081    3.5090  -10.3182   -1.3750   -2.4723    0.3054   -0.7308
  -12.7510    1.5740    2.7664    8.1034   -5.2779    1.0922   -1.6694    1.0561
    6.6005    7.8668   -4.9294   -7.0092    2.1860    0.8872    0.6653   -0.1783
   10.6630    0.4486   -0.1019    7.9728   -4.0241    2.4364   -2.3823    0.6011
```

여기서 정규화 매트릭스 Q를 읽어서 DCT의 결과를 Q로 아래와 같이 나누기 처리한다.

이 단계에서 블록의 내부에는 대부분의 값들이 0으로 바뀐다. 이 블록에 벡터를 출력하면 아래와 같이 처리된다.

-14 -3 -20 0 4 2 0 0 4 7 0 1 0 -1 -1 0 0 0 -1 0 -1 EOB

여기서 EOB는 블록의 끝을 나타낸다. 이 단계까지가 8X8 블록 단위의 처리로서 작은 값들을 포함하는 길이 21의 벡터로 축소된다.

압축을 풀기 위해 이 벡터를 위의 매트릭스 bq 형태로 각 요소의 값들을 재현한다. 그 후에 정규화 매트릭스 Q를 아래와 같이 곱하기 처리한다.

여기서 아래와 같이 역 DCT 처리한다.

```
>> q = [16 11 10 16 24 40 51 61;...
12 12 14 19 26 58 60 55;...
14 13 16 24 40 57 69 56;...
14 17 22 29 51 87 80 62;...
18 22 37 56 68 109 103 77;...
24 35 55 64 81 104 113 92;...
49 64 78 87 103 121 120   101;...
72 92 95 98 112 100 103 99];
>> bq=round(bd./q)

bq =

   -14    -3     2     0    -1     0     0     0
   -20     4     0    -1     0     0     0     0
     0     4     0     0     0     0     0     0
     7     1    -1     0     0     0     0     0
     0    -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     0
```

값을 읽는 순서는 24P

```
>> bq2=bq.*q

bq2 =

  -224   -33    20     0   -24     0     0     0
  -240    48     0   -19     0     0     0     0
     0    52     0     0     0     0     0     0
    98    17   -22     0     0     0     0     0
     0   -22     0     0     0     0     0     0
   -24     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

```
>> bd2=idct2(bq2)

bd2 =

  -48.1431  -39.4257  -39.8246  -53.5852  -65.6253  -68.5089  -70.4960  -74.9017
  -52.5762  -46.8345  -50.6187  -65.1825  -74.3228  -71.8983  -68.6282  -69.8981
  -70.6699  -66.1335  -69.6750  -80.0362  -81.2435  -69.9392  -59.8115  -57.6095
  -68.4457  -61.8134  -60.1283  -62.2478  -54.7898  -37.7751  -26.0007  -24.2342
  -29.6526  -21.6215  -16.7263  -14.7648   -5.2632    9.2177   14.8476   11.3981
    1.6297    7.3329    9.2805    8.8036   15.4106   24.9217   24.0240   15.7152
    3.0533    4.5797    1.2799   -2.1680    4.6076   15.8252   16.2949    8.4867
   -2.8366   -4.0640  -10.3394  -14.0550   -3.8001   13.2610   19.3977   14.9470
```

마지막으로 이 결과에 128을 더하여 아래와 같이 정수화 처리한다.

```
>> b2=round(bd2+128)

b2 =
```

복원된 데이터

```
    80    89    88    74    62    59    58    53
    75    81    77    63    54    56    59    58
    57    62    58    48    47    58    68    70
    60    66    68    66    73    90   102   104
    98   106   111   113   123   137   143   139
   130   135   137   137   143   153   152   144
   131   133   129   126   133   144   144   136
   125   124   118   114   124   141   147   143
```

이들 값들은 원래 블록의 값들과 매우 비슷함을 볼 수 있다. 원 데이터와 복원 데이터의 값들의 차이는 아래와 같다.

```
>> double(block)-double(b2)

ans =

     7     6     4    -1    -3    -2    -1     2
    -1   -10    -9    -4     0    -2    -8    -1
     7    -4    -1     7    11     7    -2    -5
    -3    -3     0     0     1    -1    -4     0
    -3     3     6     1    -4    -3     2     1
    -2     4     9     2    -3    -5    -1    -1
     6     2    -4    -8     4    12    10    -4
    -3    -5    -5    -4     4     3    -7    -1
```

이 알고리즘은 저주파영역에서 정밀하게 처리되며, 이 경우에 원래 블록이 매우 적은 오차범위 내에서 복원될 수 있다.

우리는 JPEG 압축을 blkpro 함수를 이용하여 실험할 수 있고, 이 함수는 영상에서 각 블록 단위로 함수를 적용하며, blkpro 함수는 파라메터로서 주어지는 블록사이즈로 처리된다. 우리는 2가지의 함수로 설계하는데, jpg_in은 각각 8X8 블록으로 압축하고 jpg_out은 복원하는 것으로서 압축의 역순으로 처리한다. 각 함수에 대하여 구체적인 파라메터 n을 포함하는데 이것은 정규화 매트릭스를 스케일하는데 사용된다. 이 함수들을 그림 14.4에 보였다. 여기서 caribou(사슴) 영상에 이를 적용해보기로 한다.

압축하기

```
function out=jpg_in(x,n)
q=[16 11 10 16 24 40 51 61;...
    12 12 14 19 26 58 60 55;...
    14 13 16 24 40 57 69 56;...
    14 17 22 29 51 87 80 62;...
    18 22 37 56 68 109 103 77;...
    24 35 55 64 81 104 113 92;...
    49 64 78 87 103 121 120 101;...
    72 92 95 98 112 100 103 99];
bd=dct2(double(x)-128);
out=round(bd./(q*n));
```

압축풀기

```
function out=jpg_out(x,n)
q=[16 11 10 16 24 40 51 61;...
    12 12 14 19 26 58 60 55;...
    14 13 16 24 40 57 69 56;...
    14 17 22 29 51 87 80 62;...
    18 22 37 56 68 109 103 77;...
    24 35 55 64 81 104 113 92;...
    49 64 78 87 103 121 120 101;...
    72 92 95 98 112 100 103 99];
out=round(idct2(x.*q*n)+128);
```

그림 14.4 JPEG 압축을 실험하기 위한 MATLAB 함수

```
>> cj1=blkproc(c,[8,8],'jpg_in',2);
>> length(find(cj1==0))

ans =

        51940
```

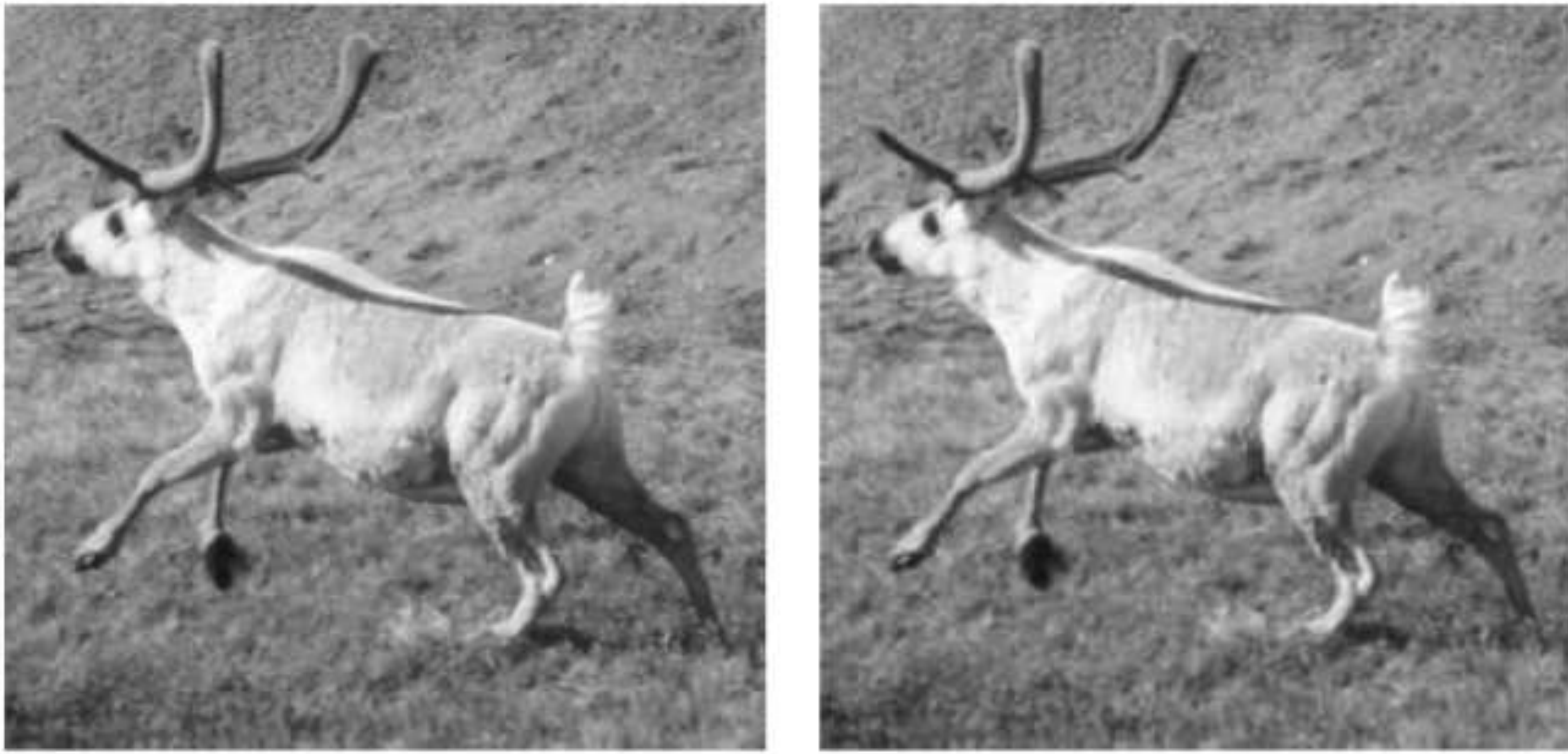
우리는 해당 영상의 각 8X8 블록으로 압축과정을 적용한다. 여기서 양자화 매트릭스로 나누고 정수화까지 처리한다. 2번째 명령의 초점은 이 단계에서 얼마나 많은 정보가 제거되지를 보는 것이다. 원 영상은 0과 128 사이에 65,536가지의 다른 정보(화소 값)를 포함하고 있다. 그러나 여기서는 단지 65,536−51,940=13,596가지의 정보만 남게 되고, 이들의 최대 및 최소값은 각각 아래와 같다.

```
>> max(cj1(:)),min(cj1(:))

ans =

    60


ans =

   -45
```

그러므로 정보는 수뿐만이 아니라 범위도 훨씬 적게 된다. 여기서 우리는 아래와 같이 그 역 과정을 처리해 보자.

```
>> C1=blkproc(cj1, [8,8], 'jpg_out', 2);
>> c1=uint8(c1);    imshow (c1/255);
```

원 영상과 그 결과 c2를 그림 14.5에 보였다. 원 영상과 복원 영상 사이에 보이는 차이는 느낄 수 없다. 그러나 이들 간에는 아래와 같이 서로 같지 않다.



Capt. Budd Christman/NOAA Corps.

(a) (b)

그림 14.5 JPEG 압축 (a) 원 영상 (b) 압축 및 복원 후 영상

```
>> max(double(c(:))-double(c1(:)))

ans =

    31

>> min(double(c(:))-double(c1(:)))

ans =

   -26
```

우리는 아래와 같이 그 차이를 그림 14.6에서 볼 수 있다.

```
>> imshow(mat2gray(double(c)-double(c1)))
```

우리는 양자화의 다른 레벨을 실험해 볼 수 있다. 여분의 파라메터 n을 2로 가정해보자. 이것은 정규화 매트릭스에서 각 값들을 2배하는 효과를 가진다. 그러므로 DCT 계수들이 아래와 같이 더 많이 0으로 된다.

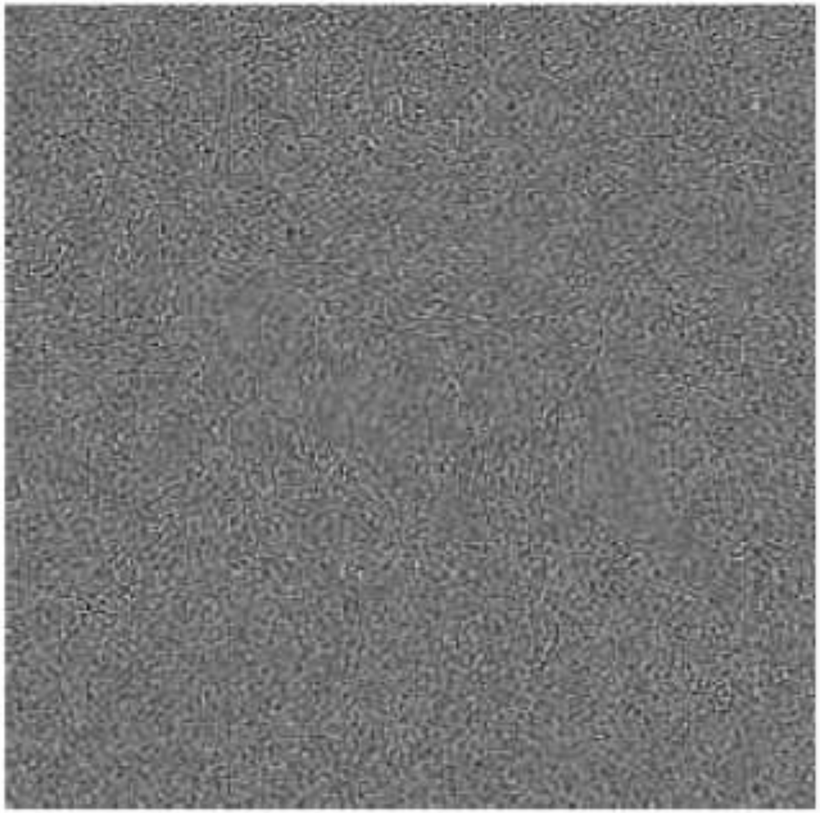
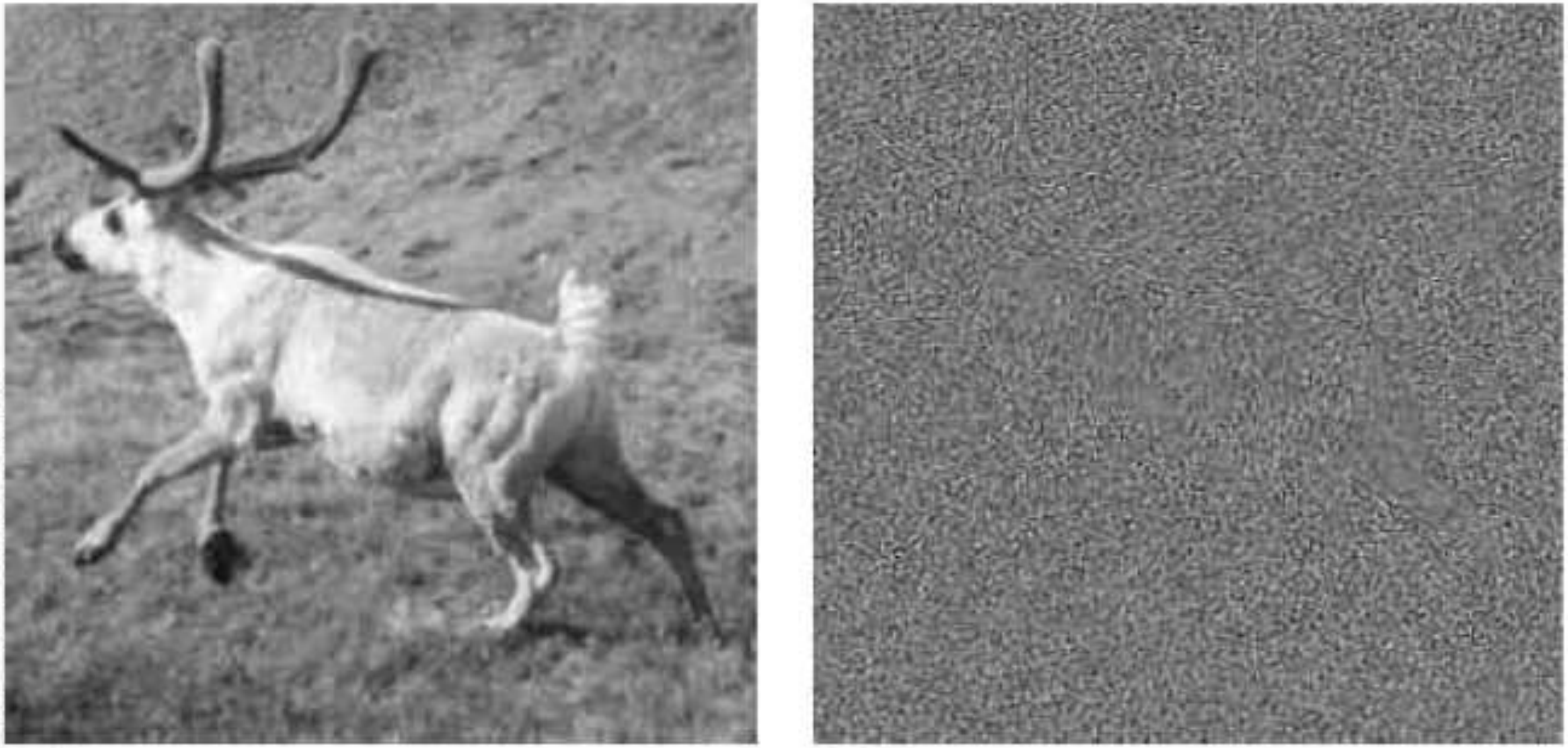


그림 14.6 원 영상과 복원 영상의 차분 영상

이것은 단지 8,807개의 값만이 0이 아니라는 것을 의미한다. 이때 cj의 최대값과 최소값이 각각 30과 −22를 구할 수 있다. 여기서 압축을 풀어서 그 결과를 디스플레이할 수 있고, 위와 같이 같은 명령에서 그 차이를 볼 수 있다.이 결과를 그림 14.7에 보였다.



Capt. Budd Christman/NOAA Corps.

(a) (b)

그림 14.7 스케일 2에 의한 복원 영상(a)와 원 영상과의 차분 영상(b)

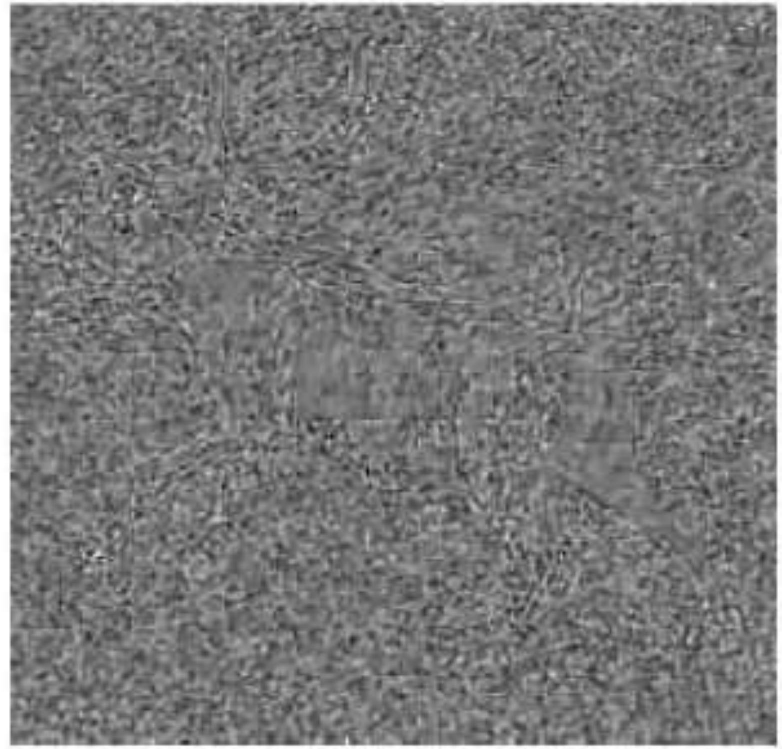
다시 스케일 값 n을 5로 아래와 같이 시도해 보자.

```
>> cj5=blkproc(c,[8,8],'jpg_in',5);
>> length(find(c5==0))

ans =

      61512
```

제 14장 영상부호화 및 압축

Capt. Budd Christman/NOAA Corps.

(a)



(b)

그림 14.8 스케일 5에 의한 복원 영상(a)와 차분 영상(b)

그 결과는 그림 14.8과 같다. 이 단계에서 얼마간의 섬세한 정보를 잃게 되지만, 영상은 여전히 화질이 좋은 편이다. 스케일 계수를 증가시키면 양자화 후 값의 범위는 감소한다. 위의 매트릭스 cj에 대하여 최대와 최소값은 각각 12와 −9이다.

마지막으로 스케일 계수를 10으로 시도하면 아래와 같다.

```
>> cj10=blkproc(c,[8,8],'jpg_in',10);
>> length(find(cj10==0))

ans =

      63684
```

최대 및 최소값은 각각 6과 −4이고, 정보는 단지 1,852개로 된다. 이 결과는 그림 14.9와 같다. 이 영상은 확실히 열화가 생김을 볼 수 있다. 그러나 동물은 여전이 깨끗하게 보인다.

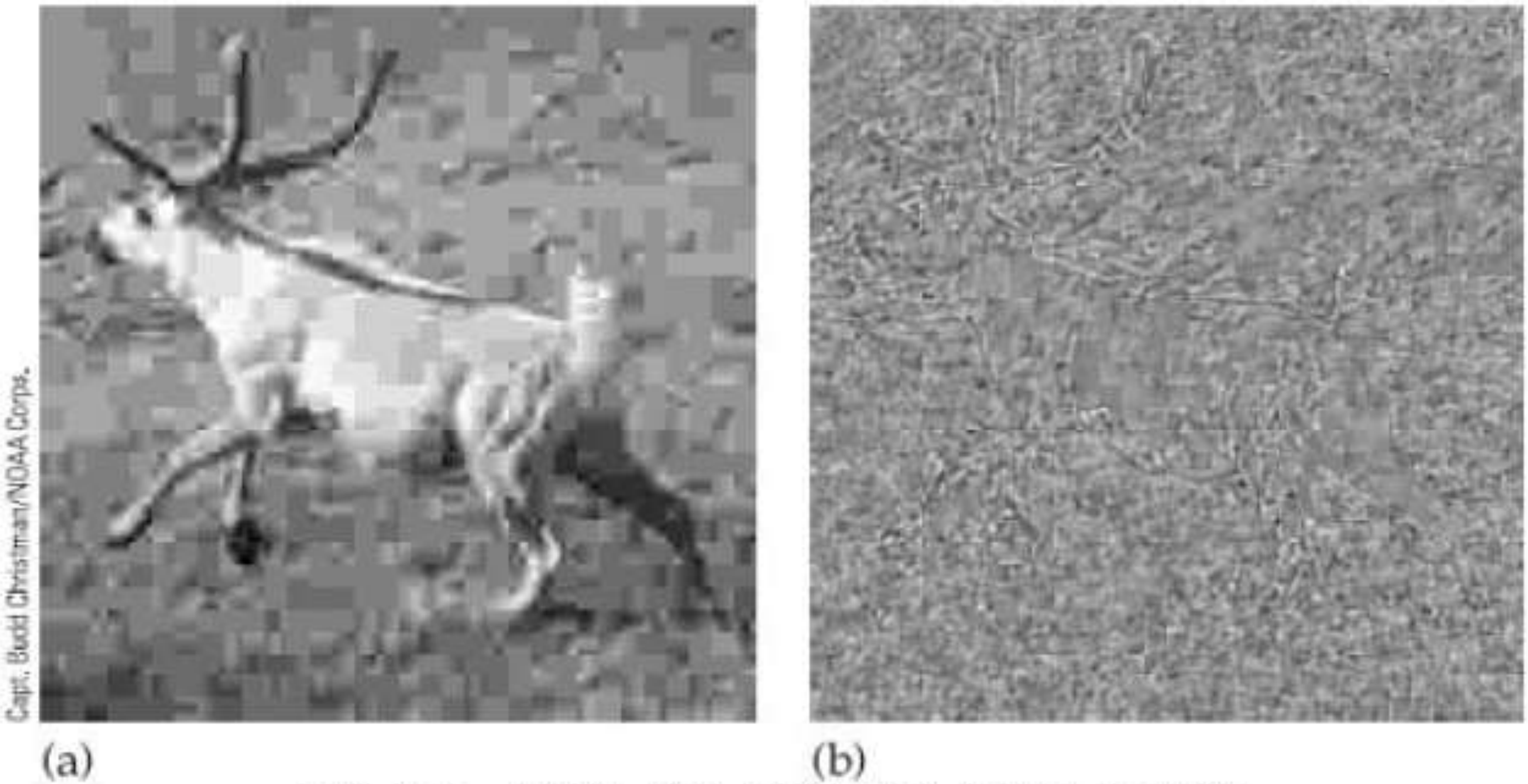


그림 14.9 스케일 계수 10일 때의 JPEG 알고리즘

우리는 이 영상을 close-up을 보면서 JPEG 압축 및 복원의 결과를 볼 수 있다. 여기서 동물의 머릿부분을 살펴보려면 아래와 같이 처리할 수 있다.

```
>> imshow(imresize(c(68-31:68+32,56-31:56+32),4))
```

이를 그림 14.10에 보였다. 같은 영역에서 스케일 계수를 각각 1과 2를 적용한 것이 그림 14.11과 같다. 또 5와 10을 적용한 결과는 그림 14.12에 보였다. 스케일 계수가 증가할수록 블록화 현상이 심해진다는 것을 알 수 있다. 이 블록화 현상은 각 8X8블록이 다른 블록의 값들과는 완전히 독립적이기 때문에 이 알고리즘의 처리에 기인된다.

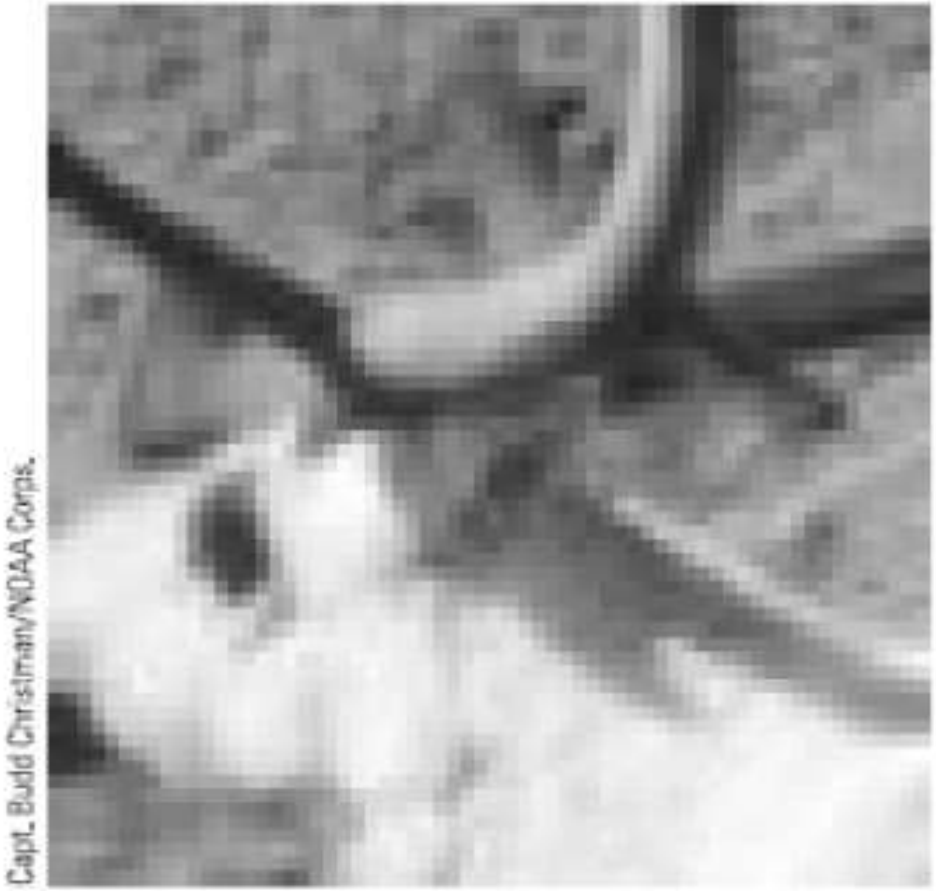


그림 14.10 close-up 영상

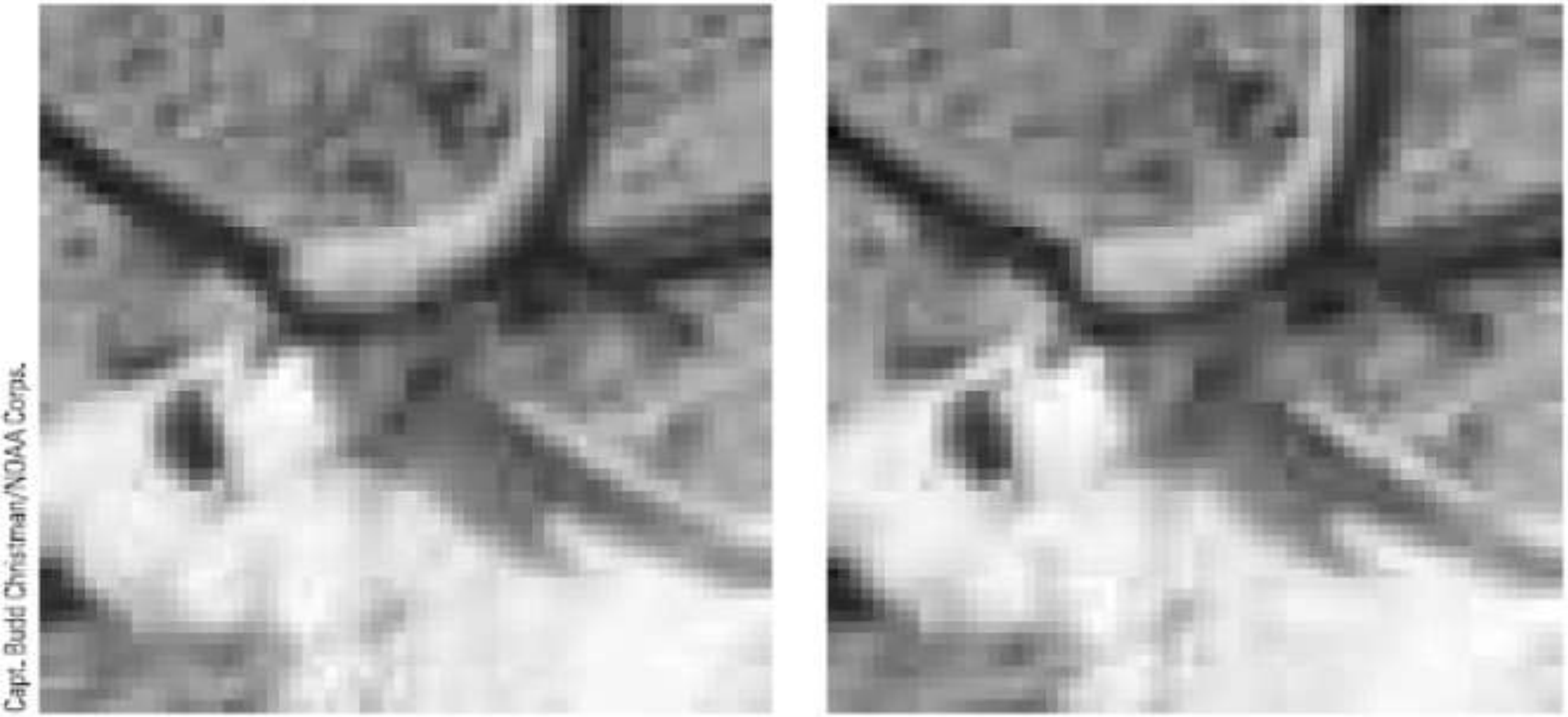
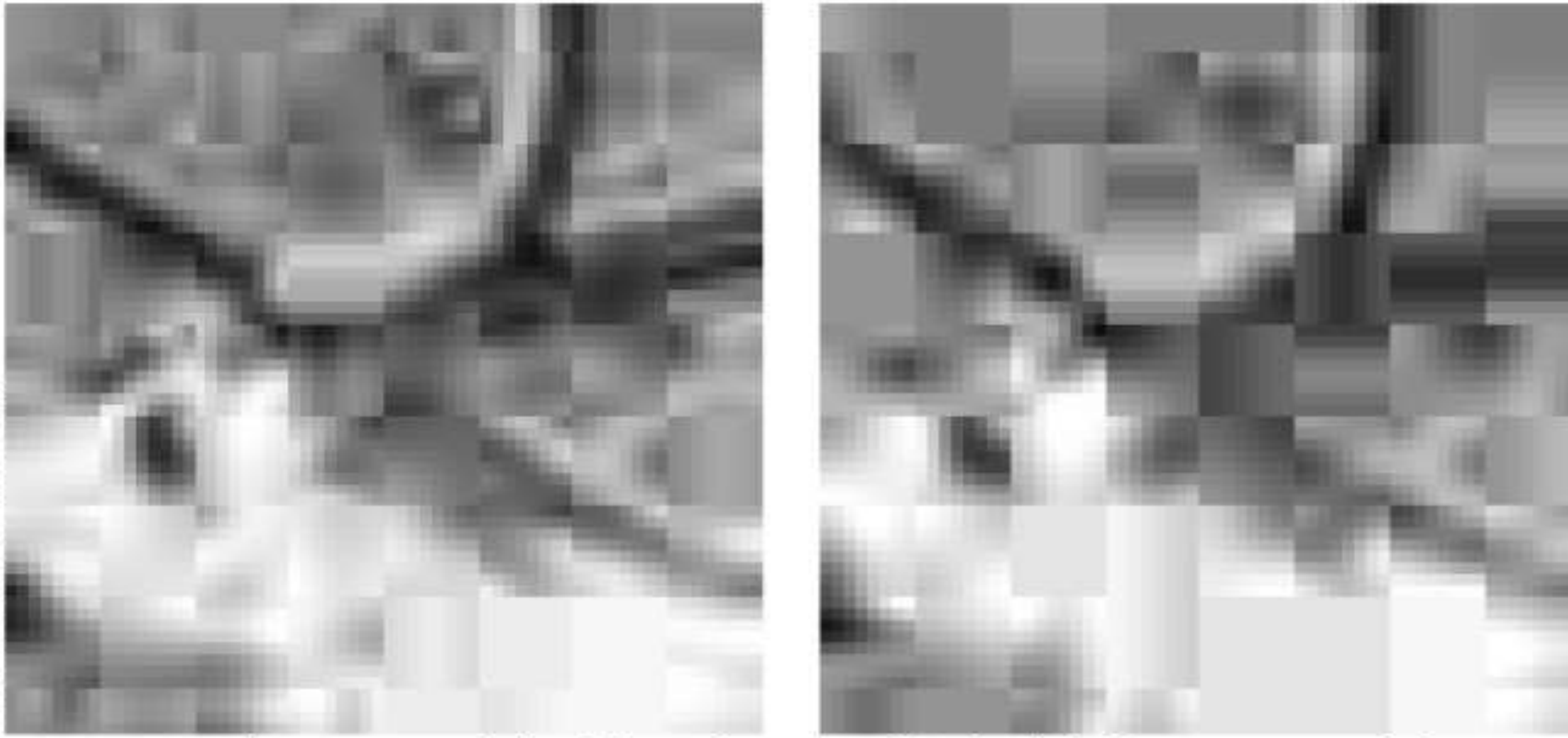


그림 14.11 스케일 계수 1과 2를 적용한 결과의 close-up 영상



Capt. Budd Christman/NOAA Corps.

그림 14.12 스케일 계수 5와 10을 적용한 결과의 close-up 영상

VQ (Vector Quantization) 느낌

실제 데이터 대신 00으로 coding

영상데이터

23	**32**	**33**	**45**	**78**	**32**
56	34	76	44	56	34
67	45	76	56	78	32
54	65	89	45	33	56
44	34	78	35	78	32
44	23	43	21	45	23

2 bit Codebook

00

34	**23**
45	56

01

34	**53**
76	43

10

55	**67**
32	45

11

25	**32**
11	23

00 codebook 과의 차이: |23-34|+|32-23|+|56-45|+|34-56|=51
01 codebook 과의 차이: 61
10 codebook 과의 차이: 102
11 codebook 과의 차이: 58