히스토그램

계산

- [0, L-1] 사이의 명암값 각각이 영상에 몇 번 나타나는지 표시
- 히스토그램 h와 정규화 히스토그램

$$h(l) = |\{(j,i) \mid f(j,i) = l\}| \tag{2.1}$$

$$\hat{h}(l) = \frac{h(l)}{M \times N} \tag{2.2}$$

알고리즘 2-1 명암 영상에서 히스토그램 계산

입력 : 명암 영상 $f(j,i)$, $0 \le j \le M-1$, $0 \le i \le N-1$

출력 : 히스토그램 $h(l)$과 정규 히스토그램 $\hat{h}(l)$, $0 \le l \le L-1$

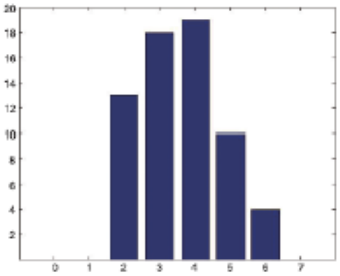
```
1  for(l=0 to L-1)    h(l)=0; // 초기화
2  for(j=0 to M-1)
3    for(i=0 to N-1)  // f의 화소 (j,i) 각각에 대해
4      h(f(j,i))++;   // 그곳 명암값에 해당하는 히스토그램 칸을 1만큼 증가
5  for(l=0 to L-1)
6    ĥ(l)=h(l)/(M×N); // 정규화한다.
```

예제 2-1 명암 영상에서 히스토그램 계산

[그림 2-7(a)]는 M과 N이 8이고 L=8인 아주 작은 영상이다. 이 영상에서 명암값이 2인 화소는 13개이므로 $h(2)$=13이다. 다른 명암값에 대해서도 화소의 개수를 세어보면 h=(0, 0, 13, 18, 19, 10, 4, 0)이고, $\hat{h}(l)$=(0, 0, 0.203, 0.281, 0.297, 0.156, 0.063, 0)이다. 이것을 그래프로 그리면 [그림 2-7(b)]와 같다.

3	2	2	2	2	3	3	4
3	2	2	2	3	4	3	3
4	3	3	4	4	4	3	3
5	4	4	4	5	4	3	3
5	4	3	4	5	4	3	2
6	5	4	4	5	4	3	2
6	6	5	5	4	3	2	2
6	5	4	5	4	3	2	2

(a) 8×8 영상 (8 명암 단계)

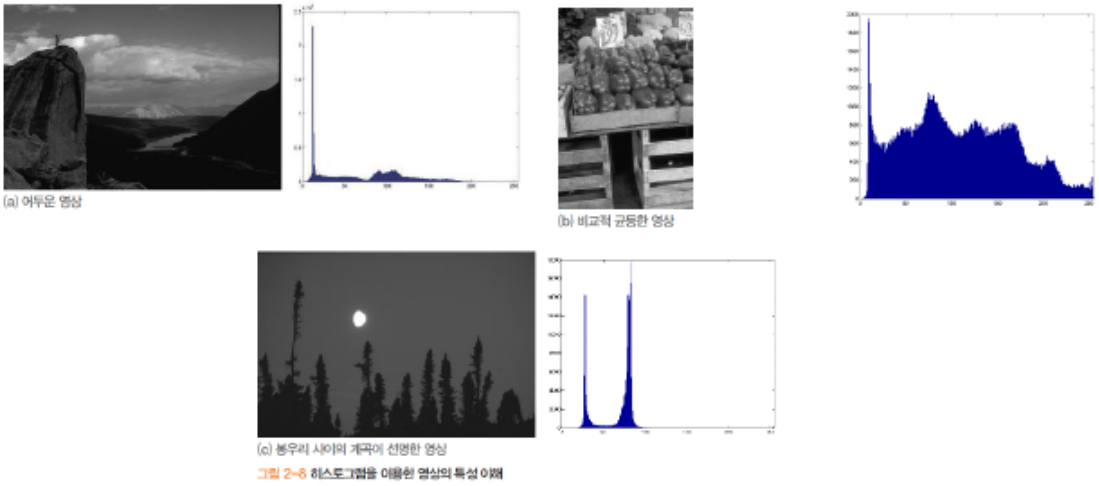


(b) 히스토그램

그림 2-7 히스토그램 예

용도

- 영상의 특성 파악



(a) 어두운 영상 (b) 비교적 균등한 영상 (c) 봉우리 사이의 계곡이 선명한 영상

그림 2-8 히스토그램을 이용한 영상의 특성 이해

- 평활화 (Histogram Equalization)
  - 히스토그램을 평평하게 만들어 주는 연산
  - 명암의 동적 범위를 확장하여 영상의 품질을 향상시켜줌
  - 누적 히스토그램 c(.)를 매핑 함수로 사용

$$I_{out} = T(I_{in}) = round(c(I_{in}) \times (L-1))$$
$$\text{이때 } c(I_{in}) = \sum_{l=0}^{I_{in}} \hat{h}(l) \qquad (2.3)$$

예제 2-2 **히스토그램 평활화**

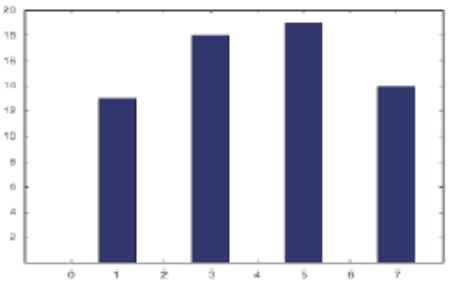
[예제 2-1]의 영상을 재활용하기로 하자. [그림 2-9(a)]에 제시된 표는 매핑 함수 $T(.)$를 구하는 과정을 보여준다. 결국 입력 영상의 명암값 0은 0, 1은 0, 2는 1, 3은 3, …, 7은 7로 매핑해 주는 함수를 얻었다. [그림 2-9(b)]는 매핑하여 얻은 평활화된 영상이다. [그림 2-9(c)]는 새로운 영상의 히스토그램이다. 이 히스토그램을 이전 영상의 히스토그램인 [그림 2-7(b)]와 비교해 보자. 이전 것은 동적 범위가 [2, 6]이었는데 새로운 영상은 [1, 7]로 보다 넓어졌음을 알 수 있다.

I_{in}	$\hat{h}(I_{in})$	$c(I_{in})$	$c(I_{in}) \times 7$	I_{out}
0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	0
2	0.203	0.203	1.421	1
3	0.281	0.484	3.388	3
4	0.297	0.781	5.467	5
5	0.156	0.937	6.559	7
6	0.063	1.0	7.0	7
7	0.0	1.0	7.0	7

(a) 매핑 표 $T(.)$

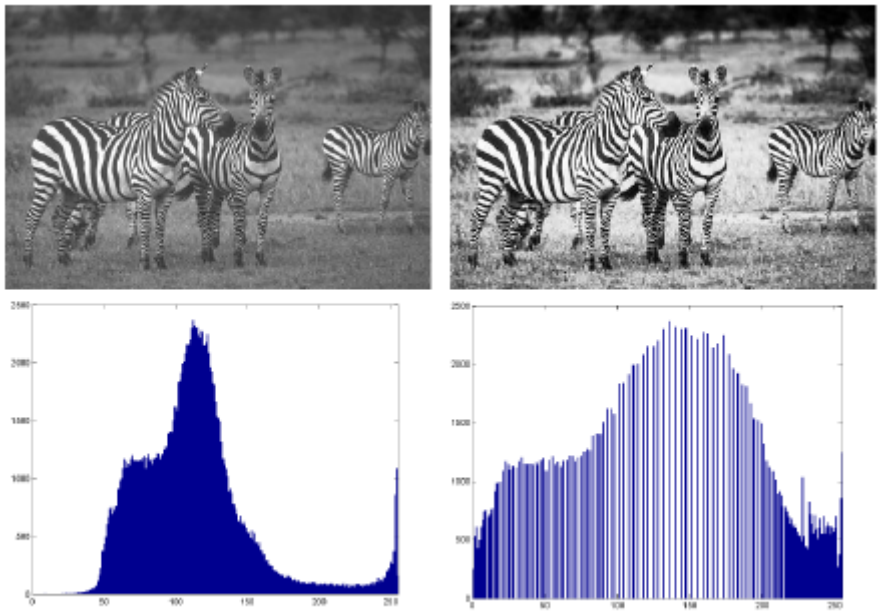
3 1 1 1 1 3 3 5
3 1 1 1 3 5 3 3
5 3 3 5 5 5 3 3
7 5 5 5 7 5 3 3
7 5 3 5 7 5 3 1
7 7 5 5 7 5 3 1
7 7 7 7 5 3 1 1
7 7 5 7 5 3 1 1

(b) 평활화된 영상



(c) 평활화된 영상의 히스토그램

그림 2-9 히스토그램 평활화 예



(a) 원래 영상 (b) 히스토그램 평활화된 영상

그림 2-10 히스토그램 평활화를 적용해 품질이 향상된 예

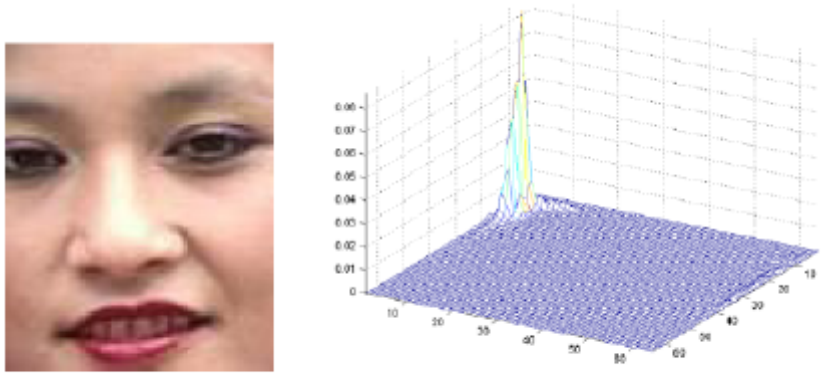
- 영상처리 연산은 분별력을 가지고 활용 여부 결정해야 함

(a) 원래 영상 (b) 히스토그램 평활화된 영상

그림 2-11 히스토그램 평활화를 적용해 시각적 느낌이 나빠진 예

히스토그램 역투영과 얼굴 검출

- 히스토그램 역투영
  - 히스토그램을 매핑 함수로 사용하여, 화소 값을 신뢰도 값으로 변환
- 얼굴 검출 예 : 모델 얼굴과 2차원 히스토그램



(a) 모델 얼굴 (b) 2차원 히스토그램(HS 공간)

그림 2-12 얼굴 검출을 위한 모델 얼굴과 히스토그램

- 2차원 히스토그램

알고리즘 2-2 2차원 히스토그램 계산(HS 공간)

입력 : H와 S채널 영상 $f_H(j,i), f_S(j,i), 0 \le j \le M-1, 0 \le i \le N-1$

출력 : 히스토그램 $h(j,i)$와 정규 히스토그램 $\hat{h}(j,i), 0 \le j,i \le q-1$ // L단계를 q단계로 양자화

```
1  h(j,i), 0≤j,i≤q−1을 0으로 초기화한다.
2  for(j=0 to M−1)
3    for(i=0 to N−0)  // 화소 (j,i) 각각에 대해
4      h(quantize(f_H(j,i)), quantize(f_S(j,i)))++;  // 해당 칸을 1 증가시킴
5  for(j=0 to q−1)
6    for(i=0 to q−1)
7      ĥ(j,i)=h(j,i)/(M×N);  // 정규화
```

- 얼굴 검출
  - 모델 얼굴에서 구한 히스토그램 h_m은
    화소의 컬러 값을 얼굴에 해당하는 신뢰도 값으로 변환해줌
  - 실제로는 비율 히스토그램 h_r을 사용

$$h_r(j,i) = \min\left(\frac{\hat{h}_m(j,i)}{\hat{h}_i(j,i)}, 1.0\right), \quad 0 \le j, \ i \le q-1 \qquad (2.4)$$

- 히스토그램 역투영 알고리즘

알고리즘 2-3 히스토그램 역투영

입력 : H와 S채널 영상 $g_H(j,i), g_S(j,i), 0 \le j \le M-1, 0 \le i \le N-1$ // 얼굴을 검출하려는 영상
모델 히스토그램 $\hat{h}_m(j,i), 0 \le j,i \le q-1$

출력 : 가능성 맵 $o(j,i), 0 \le j \le M-1, 0 \le i \le N-1$

```
1  영상 g_H, g_S에 [알고리즘 2-2]를 적용하여 정규 히스토그램 ĥ_i를 만든다.
2  식 (2.4)를 이용하여 ĥ_r을 구한다.
3  for(j=0 to M−1)
4    for(i=0 to N−1)
5      o(j,i)=ĥ_r(quantize(g_H(j,i)), quantize(g_S(j,i)));  // 역투영
```

- 히스토그램 역투영 결과
  - 얼굴 영역은 높은 신뢰도 값, 손 영역도 높은 값
  - 한계
    - 비슷한 색 분포를 갖는 다른 물체 구별 못함
    - 검출 대상이 여러 색 분포를 갖는 경우 오류 가능성
  - 장점 : 배경을 조정할 수 있는 상황에 적합 (이동과 회전에 불변, 가림(occlusion)에 강인)

(a) 입력 영상 (b) 역투영 영상

그림 2-13 히스토그램 역투영을 이용한 얼굴 검출

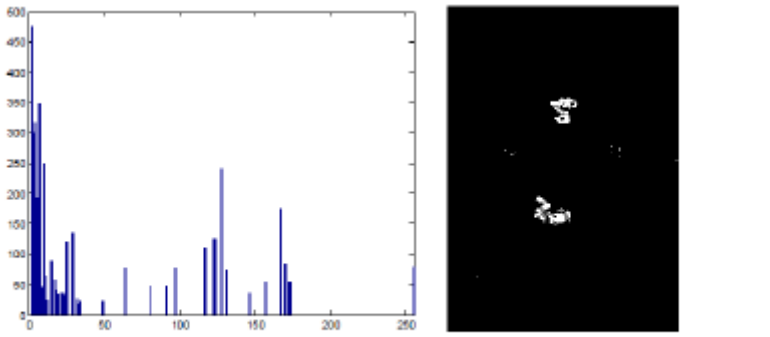
이진 영상

이진화와 오츄 알고리즘

- 이진화 : 명암 영상을 흑과 백만 가진 이진 영상으로 변환

$$b(j,i) = \begin{cases} 1, & f(j,i) \geq T \\ 0, & f(j,i) < T \end{cases} \qquad (2.5)$$

- 임계값 방법
  - 두 봉우리 사이의 계곡을 임계값 T로 결정
  - 자연 영상에서는 계곡 지점 결정이 어려움



(a) 히스토그램 (b) 임계값을 이용하여 구한 이진 영상(T=50)

그림 2-14 이진화

- 오츄 알고리즘 [Otsu79]
  - 이진화 했을 때 흑 그룹과 백 그룹 각각이 균일할수록 좋다는 원리에 근거
  - 균일성은 분산으로 측정 (분산이 작을수록 균일성 높음)
  - 분산의 가중치 합 V_within(.)을 목적 함수로 이용한 최적화 알고리즘

$$T = \underset{t \in \{0,1,\cdots,L-1\}}{\operatorname{argmin}} \; v_{within}(t) \tag{2.6}$$

$$v_{within}(t) = w_0(t)\,v_0(t) + w_1(t)\,v_1(t)$$

$$\begin{aligned} w_0(t) &= \sum_{i=0}^{t} \hat{h}(i), & w_1(t) &= \sum_{i=t+1}^{L-1} \hat{h}(i) \\ \mu_0(t) &= \frac{1}{w_0(t)} \sum_{i=0}^{t} i\,\hat{h}(i), & \mu_1(t) &= \frac{1}{w_1(t)} \sum_{i=t+1}^{L-1} i\,\hat{h}(i) \\ v_0(t) &= \frac{1}{w_0(t)} \sum_{i=0}^{t} \hat{h}(i)(i-\mu_0(t))^2, & v_1(t) &= \frac{1}{w_1(t)} \sum_{i=t+1}^{L-1} \hat{h}(i)(i-\mu_1(t))^2 \end{aligned} \tag{2.7}$$

- t-1 번째의 계산 결과를 t번째에 활용하여 빠르게 계산

$$T = \underset{t \in \{0,1,\cdots,L-1\}}{\operatorname{argmax}} \; v_{between}(t) \tag{2.8}$$

여기에서 $v_{between}(t) = w_0(t)(1-w_0(t))(\mu_0(t)-\mu_1(t))^2$

$$\begin{aligned} &\text{초깃값}\,(t=0): w_0(0) = \hat{h}(0), \quad \mu_0(0) = 0 \\ &\text{순환식}\,(t>0): \\ &\quad w_0(t) = w_0(t-1) + \hat{h}(t) \\ &\quad \mu_0(t) = \frac{w_0(t-1)\mu_0(t-1) + t\hat{h}(t)}{w_0(t)} \\ &\quad \mu_1(t) = \frac{\mu - w_0(t)\mu_0(t)}{1-w_0(t)} \end{aligned} \tag{2.9}$$

알고리즘 2-4 오츄 알고리즘(효율적인 버전)

입력 : 영상 $f(j,i)$, $0 \le j \le M-1$, $0 \le i \le N-1$

출력 : 이진 영상 $b(j,i)$, $0 \le j \le M-1$, $0 \le i \le N-1$

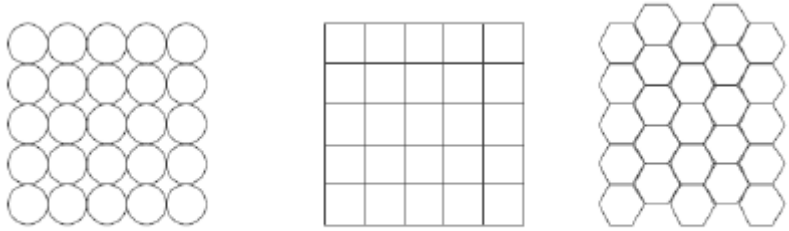
```
1  [알고리즘 2-1]을 이용하여 f의 정규 히스토그램 ĥ을 만든다.
2  식 (2.9)의 초기 조건을 이용하여 w0(0)과 μ0(0)을 계산한다.
3  for(t=1 to L-1) {
4    식 (2.9)의 순환식을 이용하여 w0(t), μ0(t), μ1(t)를 계산한다.
5    식 (2.8)을 이용하여 vbetween(t)를 계산한다.
6  }
7  앞의 for 루프에서 가장 큰 vbetween(t)를 보인 t를 임계값 T로 취한다.
8  식 (2.5)로 f를 이진화하여 b를 만든다.
```

(a) T=70 (b) T=111 (c) T=54

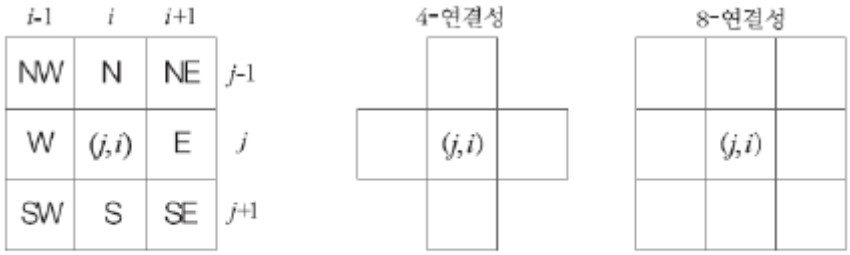
그림 2-15 오츄 알고리즘이 찾아준 임계값 T로 이진화한 영상

연결 요소

- 화소의 모양과 연결성



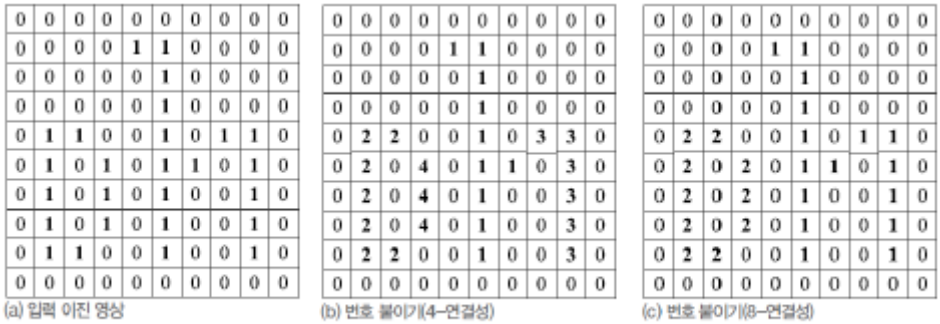
(a) 생각해 볼 수 있는 화소의 여러 가지 모양



(b) 화소의 연결성

그림 2-16 화소의 모양과 연결성

- 연결요소 번호 붙이기
  - 4-연결성과 8-연결성



(a) 입력 이진 영상 (b) 번호 붙이기(4-연결성) (c) 번호 붙이기(8-연결성)

그림 2-17 연결요소 번호 붙이기

- 범람 채움
  - 스택 오버플로우 위험

알고리즘 2-5 범람 채움(4-연결성 버전)

입력 : 이진 영상 $b(j,i), 0 \le j \le M-1, 0 \le i \le N-1$

출력 : 번호를 매긴 영상 $l(j,i), 0 \le j \le M-1, 0 \le i \le N-1$

```
1  b를 l로 복사한다. 이때 0은 0, 1은 -1로 복사한다. // -1은 아직 번호를 안 붙였음을 표시
2  l의 경계, 즉 j=0, j=M-1, i=0, i=N-1인 화소를 0으로 설정한다. // 영상 바깥으로 나가는 것을 방지
3  label=1;
4  for(j=1 to M-2)
5    for(i=1 to N-2) {
6      if(l(j,i)=-1) {
7        flood_fill4(l, j, i, label);
8        label++;
9      }
10   }
11
12 // 4-연결성 범람 채움 함수
13 function flood_fill4(l, j, i, label) {
14   if(l(j,i)=-1) { // 아직 번호를 안 붙인 화소이면
15     l(j,i)=label;
16     flood_fill4(l, j, i+1, label); // east
17     flood_fill4(l, j-1, i, label); // north
18     flood_fill4(l, j, i-1, label); // west
19     flood_fill4(l, j+1, i, label); // south
20   }
21 }
```

- 열 단위로 처리하는 알고리즘

알고리즘 2-6 범람 채움(메모리를 적게 사용하는 버전)

입력 : 이진 영상 $b(j,i), 0 \le j \le M-1, 0 \le i \le N-1$

출력 : 번호를 매긴 영상 $l(j,i), 0 \le j \le M-1, 0 \le i \le N-1$

```
1  b를 l로 복사한다. 이때 0은 0, 1은 -1로 복사한다. // -1 은 아직 번호를 안 붙였음을 표시
2  l의 경계, 즉 j=0, j=M-1, i=0, i=N-1인 화소를 0으로 설정한다. // 영상 바깥으로 나가는 것 방지
3  label=1;
4  for(j=1 to M-2)
5    for(i=1 to N-2) {
6      if(l(j,i)=-1 ) {
7        efficient_floodfill4(l,j,i,label );
8        label++;
9      }
10   }
11
12 // 메모리를 적게 사용하는 효율적인 4-연결성 범람 채움 함수
13 function efficient_floodfill4(l,j,i,label ) {
14   Q=Ø; // 빈 큐 Q를 생성한다.
15   push(Q,(j,i));
16   while(Q≠Ø) {
17     (y,x)=pop(Q); // Q에서 원소를 하나 꺼낸다.
18     if(l(y,x)=-1) {
19       left=right=x;
20       while(l(y,left-1)=-1) left--;  // 아직 미처리 상태인 열을 찾는다.
21       while(l(y,right+1)=-1) right++;
22       for(c=left to right) {
23         l(y,c)=label;
24         if(l(y-1,c)=-1 and (c=left or l(y-1,c-1)≠-1)) push(Q, (y-1,c));
25         if(l(y+1,c)=-1 and (c=left or l(y+1,c-1)≠-1)) push(Q, (y+1,c));
26       }
27     }
28   }
29 }
```