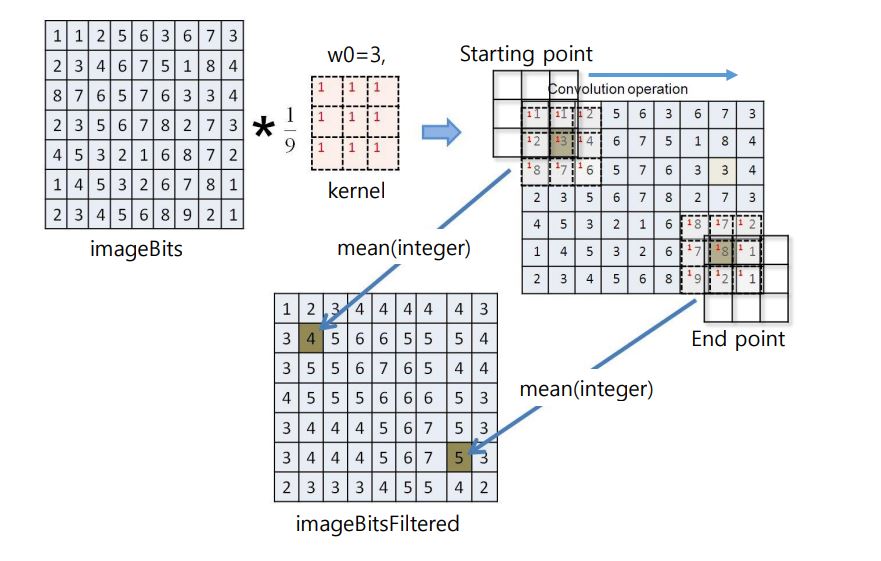
고급소프트웨어

2주차 과제

20170101

이은지

1. **Average 필터링을 효율적으로 구현할 수 있는 Moving Average에 대하여 공식을 이용하여 설명한다.**



Image의 row와 column을 간편하게 r, c라고 할 것이다.

그리고 평균 처리 과정에서 픽셀의 위치에 따른 가중치는 없기 때문에 mean이라는 하나의 변수에 모든 픽셀값을 저장하면 되고, 별도로 average 배열을 사용할 필요가 없지만 빠른 이해를 위해서 개념적으로만 2차원의 average array를 사용할 예정이다.

예시로 3가지의 경우를 살펴본 후 공식을 도출할 수 있다.

1. **r = 0, c = 0**

average array = [[0,0,0],

[0,1,1],

[0,2,3]]

1. **r = 0, c =1**

average array = [[0,0,0],

[1,1,2],

[2,3,4]]

1. **r= 0, c =2**

average array = [[0,0,0],

[1,2,5],

[3,4,6]]

위에서 볼 수 있다시피 가장 왼쪽 열이 제거되고 오른쪽 열을 새로 채워 3 x 3 배열을 형성한다. 우선 c = 0인 경우부터 커널의 사이즈에 해당하는 픽셀 값들의 평균을 구한다. 그리고 같은 r인 경우에 한하여 왼쪽열과 오른쪽 열 처리를 각각 나누어 왼쪽 열 값을 제거한 후 오른쪽 열 값을 추가하는 형태로 연산을 최소화한다. 만약 왼쪽이나 오른쪽 열 픽셀의 좌표가 이미지를 벗어난 경우에는 값을 0으로 처리한다.

커널 크기를 3x3으로 가정하였을 때,

(r,c)에서의 평균값은 1/(3\*3) \*

[[imageBits[r -1, c - 1], imageBits [r - 1, c], imageBits [r – 1, c + 1]],

[imageBits [r, c - 1], imageBits [r, c], imageBits [r, c+1]],

imageBits [r + 1 , c - 1], imageBits [r + 1, c], imageBits [r + 1, c + 1]]의 원소들의 합이다.

따라서

w0 = the width of kernel

w0 \* w0 = the size of kernel

w =

If c = 0,

mean(r, c)= 1/(w0\*w0) \*

(if r-k < 0 or r-k width or c- l<0 or c-l height, the value is 0)

If c >0,

mean(r, c) = mean(r,c-1) – 1/(w0\*w0) \*(

+ )

(if r-k < 0 or r-k width or c-w-1<0 or c + w height, the value is 0)

1. **Median 필터링을 효율적으로 구현할 수 있는 방법에 관하여 도식적으로 설명해보자. 3x3 윈도우 사이즈를 사용하는 경우를 가정하여 구체적으로 설명한다.**
2. **Moving Median**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1** | **1** | **2** | **5** | **6** | **3** |
| **2** | **3** | **4** | **6** | **7** | **5** |
| **8** | **7** | **6** | **5** | **7** | **6** |
| **2** | **3** | **4** | **5** | **6** | **7** |
| **4** | **5** | **3** | **2** | **1** | **6** |
| **1** | **4** | **5** | **3** | **2** | **6** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1** | **1** | **2** | **5** | **6** | **3** |
| **2** | **3** | **4** | **6** | **7** | **5** |
| **8** | **7** | **6** | **5** | **7** | **6** |
| **2** | **3** | **4** | **5** | **6** | **7** |
| **4** | **5** | **3** | **2** | **1** | **6** |
| **1** | **4** | **5** | **3** | **2** | **6** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1** | **1** | **2** | **5** | **6** | **3** |
| **2** | **3** | **4** | **6** | **7** | **5** |
| **8** | **7** | **6** | **5** | **7** | **6** |
| **2** | **3** | **4** | **5** | **6** | **7** |
| **4** | **5** | **3** | **2** | **1** | **6** |
| **1** | **4** | **5** | **3** | **2** | **6** |

실습 때 익혔던 moving average와 유사한 방식으로 가장 왼쪽 열을 제거하고, 오른쪽 열을 추가하여 median을 새롭게 구하는 것이다. 이 방식의 장점은 median을 이전보다 빠르게 구할 수 있다는 장점이 있다. 하지만 동일한 행에 위치한 픽셀들의 median값을 구할 때마다 제거하는 열과 추가하는 열을 매번 계산하는 것이 지나치게 복잡하다.

1. **Improved Median**

Median filtering의 기본 목적은 이미지의 노이즈를 제거하기 위함이다. 노이즈는 주위 픽셀들과 차이가 많이 나므로 그러한 픽셀들을 제거함으로써 더 이미지 화질을 높일 수 있다.

|  |  |  |
| --- | --- | --- |
| 0 | 50 | 2 |
| 90 | 255 | 70 |
| 80 | 60 | 250 |

Median array = [0,2,50,60,**70**,90,250,255]

1. 여기에서 0과 255와 같은 양극단 픽셀 값을 median 값으로 바꾼다.

|  |  |  |
| --- | --- | --- |
| 70 | 50 | 2 |
| 90 | 70 | 70 |
| 80 | 60 | 250 |

Median array = [2,50,60,70,**70**,70,90,250]

1. 픽셀 간 최대 차이 d를 두어 배열의 원소개수가 총 n개라면

Array[0]와 array[1]의 차이, 그리고 array[n]과 array[n-1]의 차이가 d보다 클 경우에는 median 값으로 대체한다.

d = 10으로 둔다면

이 예시의 경우에는 50-2 = 48>10, 250 – 90 = 160 > 10이므로

|  |  |  |
| --- | --- | --- |
| 70 | 50 | 70 |
| 90 | 70 | 70 |
| 80 | 60 | 70 |

Median array = [50,60,70,70,**70**,70,70,90]

위의 경우로 median filtering을 한다면 노이즈를 최소화할 수 있다..