영상처리프로그래밍 : 영상의 리샘플링과 보간 (Interpolation)

한림대학교 소프트웨어융합대학
박섭형

2022년 1학기

[2]:
```
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
```

0.1 영상의 표현

- $f(x,y)$
  - x,y: 화소의 위치를 나타내는 좌표
  - $f(x,y)$: x,y 위치에 있는 화소의 색 또는 밝기
- NumPy의 ndarray로 영상 데이터를 표현하는 경우

[3]:
```
file_name = 'clock_260.jpg'
img1 = cv2.imread(file_name)
img1.shape
```

[3]:
```
(260, 260, 3)
```

[4]:
```
cv2.imshow("clock-260", img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[5]:
```
img1[10,20]
```

[5]:
```
array([ 81, 116, 149], dtype=uint8)
```

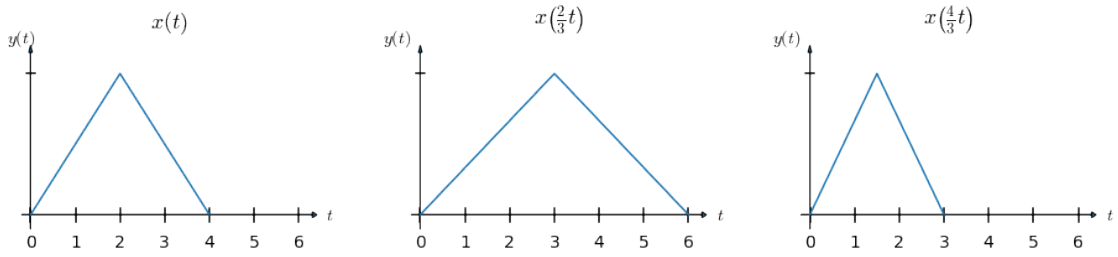
1 영상의 크기 변환

- 업 샘플링(up-sampling): 영상 확대
- 다운 샘플링 (down-sampling): 영상 축소

1차원 연속 시간 신호의 확대와 축소

$$y(t) = x(bt)$$

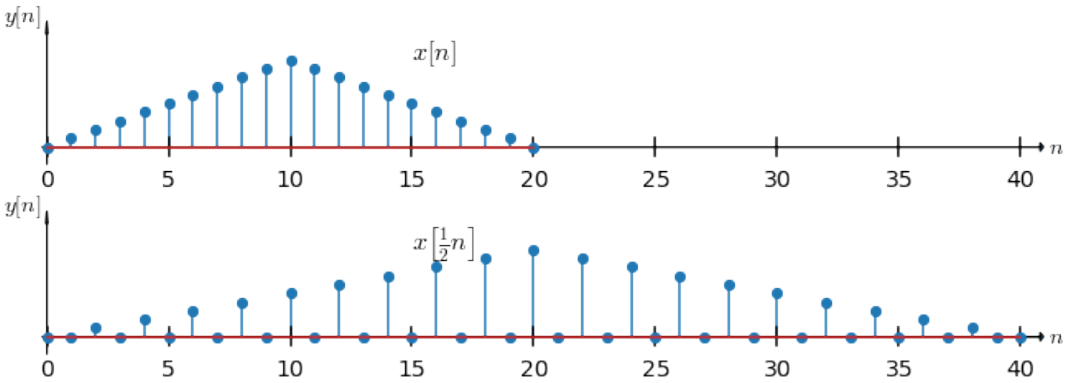
- t: 실수
- 확대 : $0 < b < 1$
- 축소 : $b > 1$



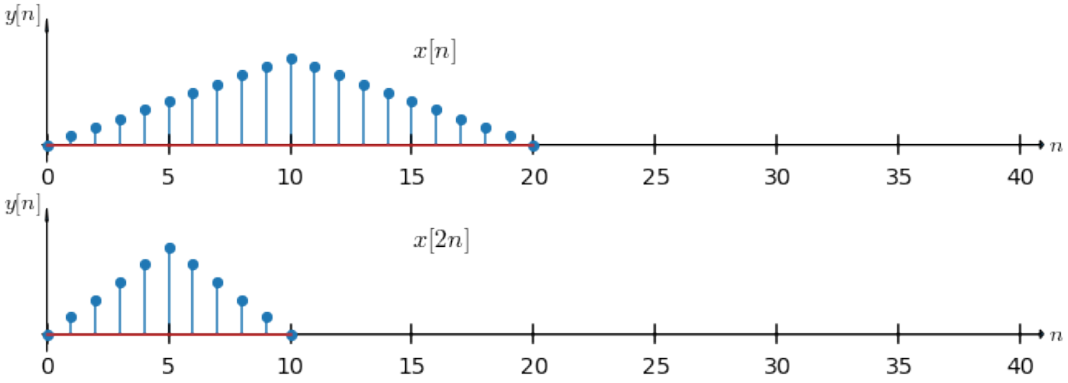
1차원 이산시간 신호의 확대와 축소

$$y[n] = x[bn]$$

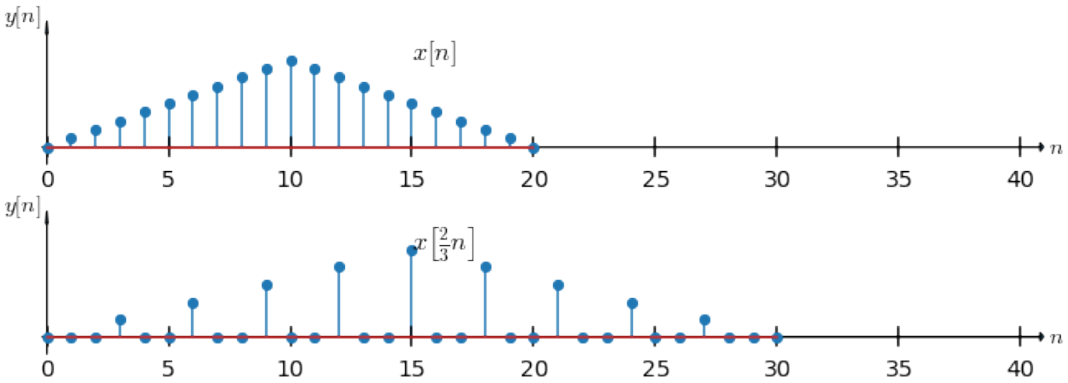
- n: 정수
- 확대 : $0 < b < 1$
- 축소 : $b > 1$



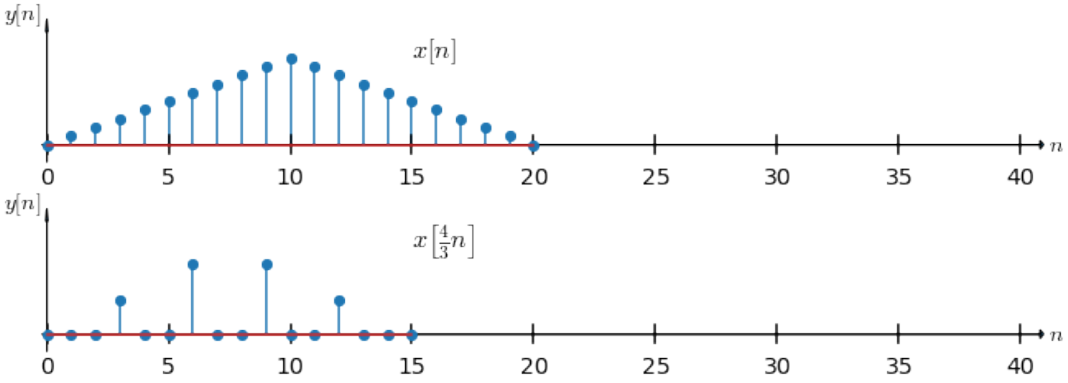
n	0	1	2	3	4	5	6	7	8	9	10	11	...
$\frac{1}{2}n$	0	$\frac{1}{2}$	1	$\frac{3}{2}$	2	$\frac{5}{2}$	3	$\frac{7}{2}$	4	$\frac{9}{2}$	5	$\frac{11}{2}$	...



n	0	1	2	3	4	5	6	7	8	9	10	11	...
$2n$	0	2	4	6	8	10	12	14	16	18	20	22	...

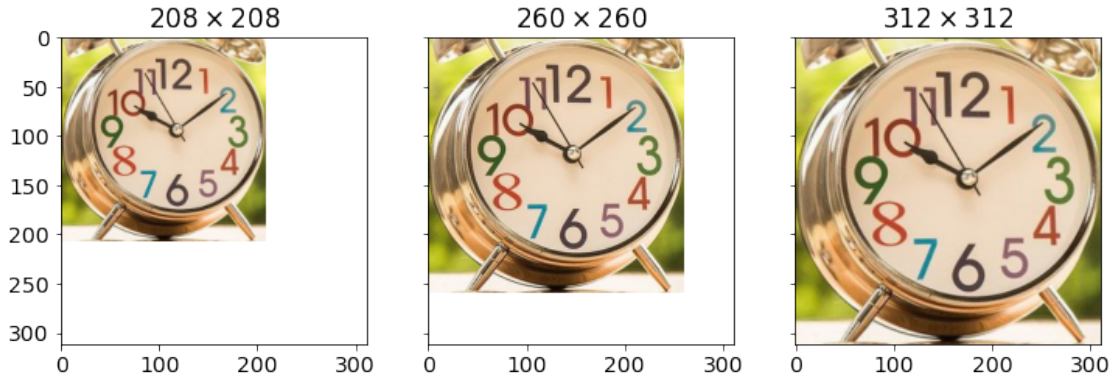


n	0	1	2	3	4	5	6	7	8	9	10	11	...
$\frac{2}{3}n$	0	$\frac{2}{3}$	$\frac{4}{3}$	2	$\frac{8}{3}$	$\frac{10}{3}$	4	$\frac{14}{3}$	$\frac{16}{3}$	6	$\frac{20}{3}$	$\frac{22}{3}$	...



n	0	1	2	3	4	5	6	7	8	9	10	11	...
$\frac{4}{3}n$	0	$\frac{4}{3}$	$\frac{8}{3}$	4	$\frac{16}{3}$	$\frac{20}{3}$	8	$\frac{28}{3}$	$\frac{32}{3}$	12	$\frac{40}{3}$	$\frac{44}{3}$	...

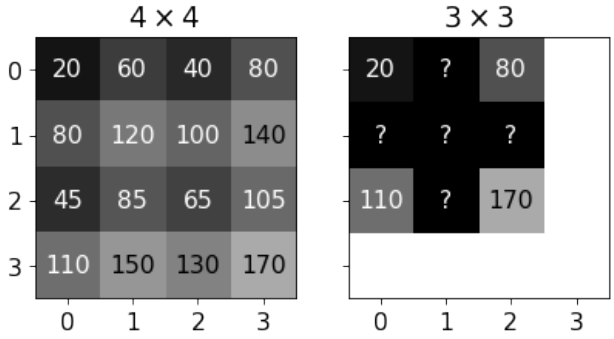
1.1 영상 확대와 축소



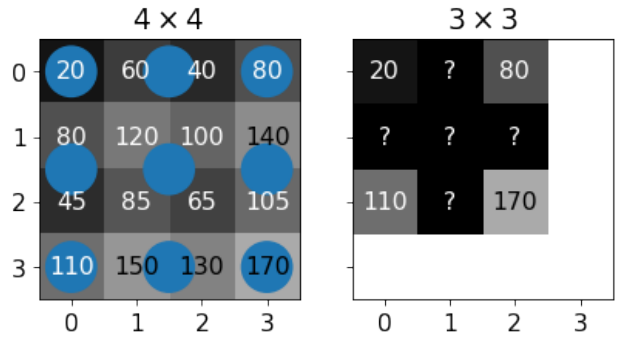
영상 확대와 축소 과정

- 같은 영역을 표현하는 화소 수를 변경하는 것
- 크기가 변경된 영상의 일부 화소들은 원래 영상에서 대응되는 화소가 없는 경우 발생

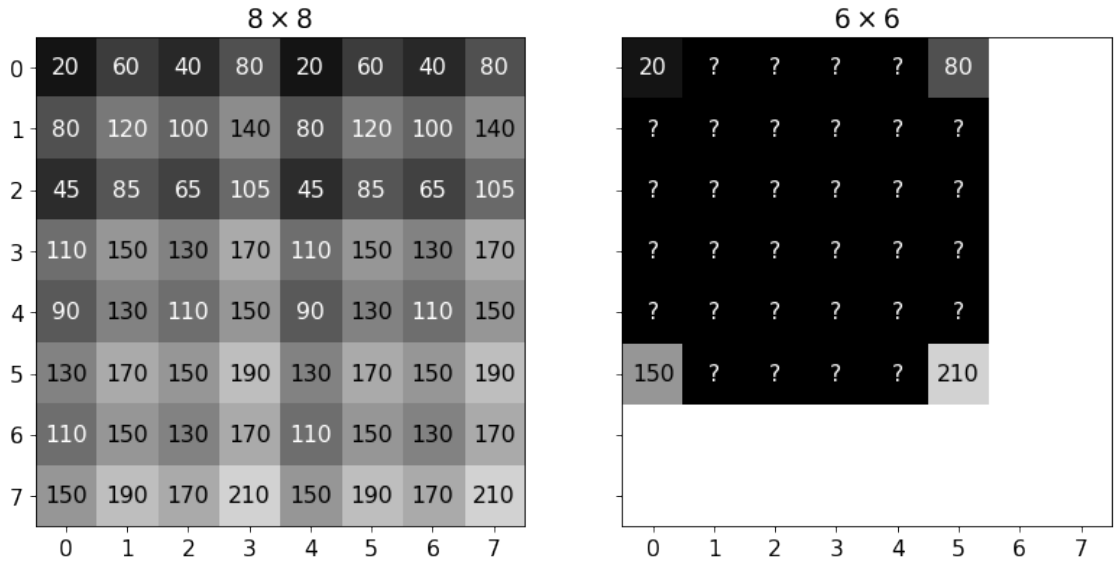
영상 축소

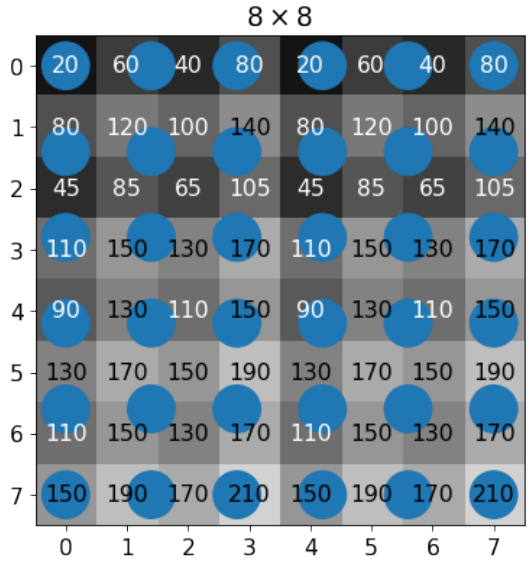


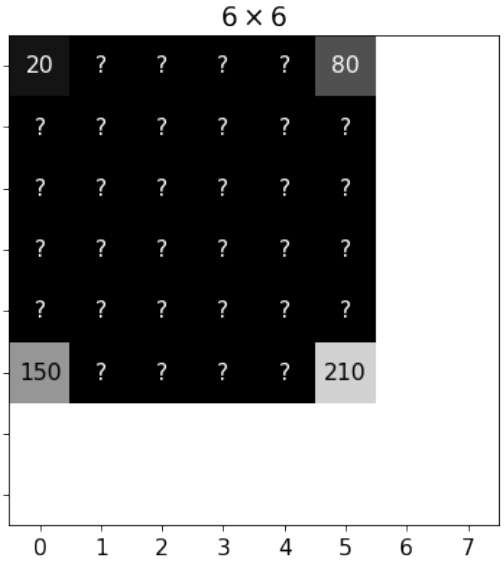
- 16 개의 화소로부터 9 개의 화소를 어떻게 만들까?

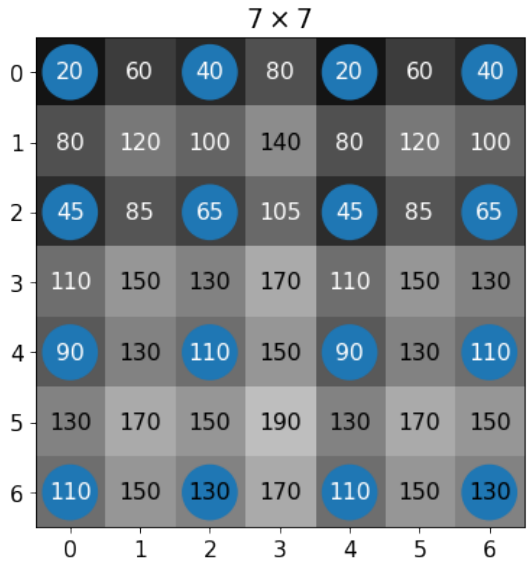


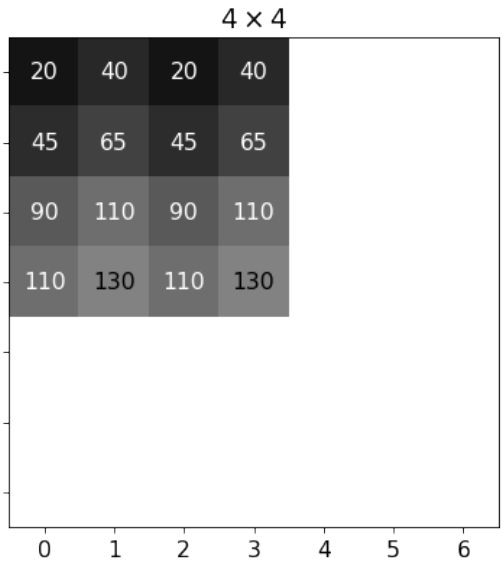
- 부화소 (subpixel): 정수 인덱스가 아닌 곳에 위치한 화소
- 보간 (interpolation): 부화소 값을 예측하는 것



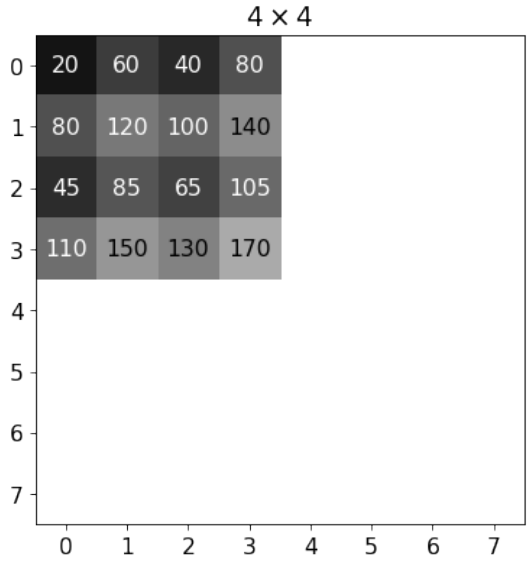

8 × 8
0 20 60 40 80 20 60 40 80
1 80 120 100 140 80 120 100 140
2 45 85 65 105 45 85 65 105
3 110 150 130 170 110 150 130 170
4 90 130 110 150 90 130 110 150
5 130 170 150 190 130 170 150 190
6 110 150 130 170 110 150 130 170
7 150 190 170 210 150 190 170 210
0 1 2 3 4 5 6 7

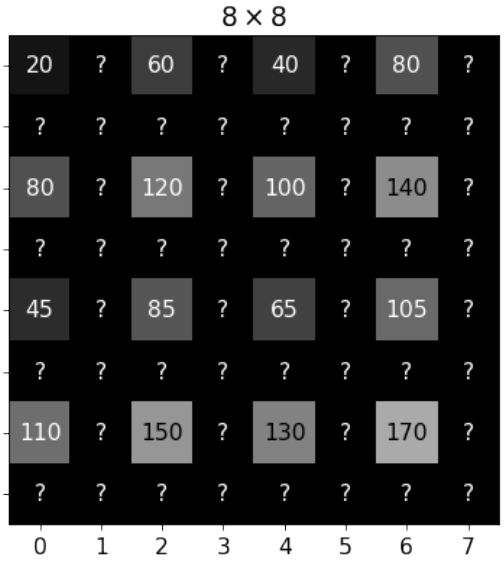

6 × 6
20 ? ? ? ? 80
? ? ? ? ? ?
? ? ? ? ? ?
? ? ? ? ? ?
? ? ? ? ? ?
150 ? ? ? ? 210
0 1 2 3 4 5 6 7

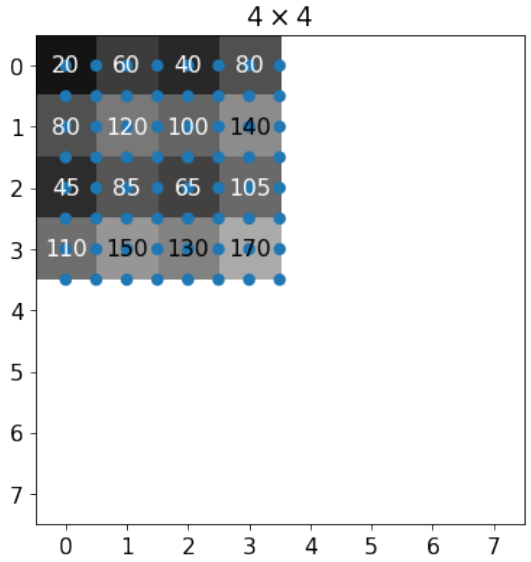

7 × 7
0 20 60 40 80 20 60 40
1 80 120 100 140 80 120 100
2 45 85 65 105 45 85 65
3 110 150 130 170 110 150 130
4 90 130 110 150 90 130 110
5 130 170 150 190 130 170 150
6 110 150 130 170 110 150 130
0 1 2 3 4 5 6

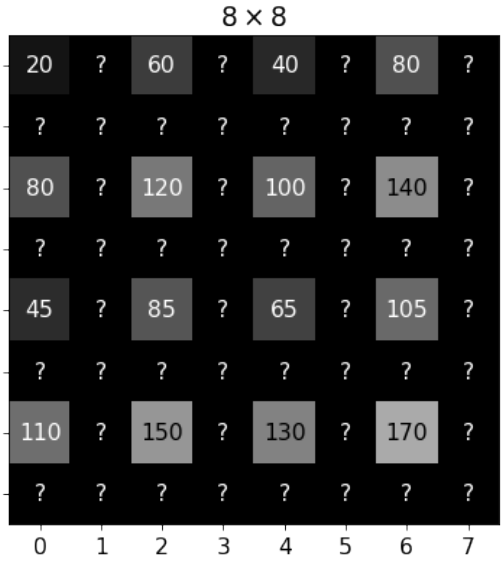

4 × 4
20 40 20 40
45 65 45 65
90 110 90 110
110 130 110 130
0 1 2 3 4 5 6

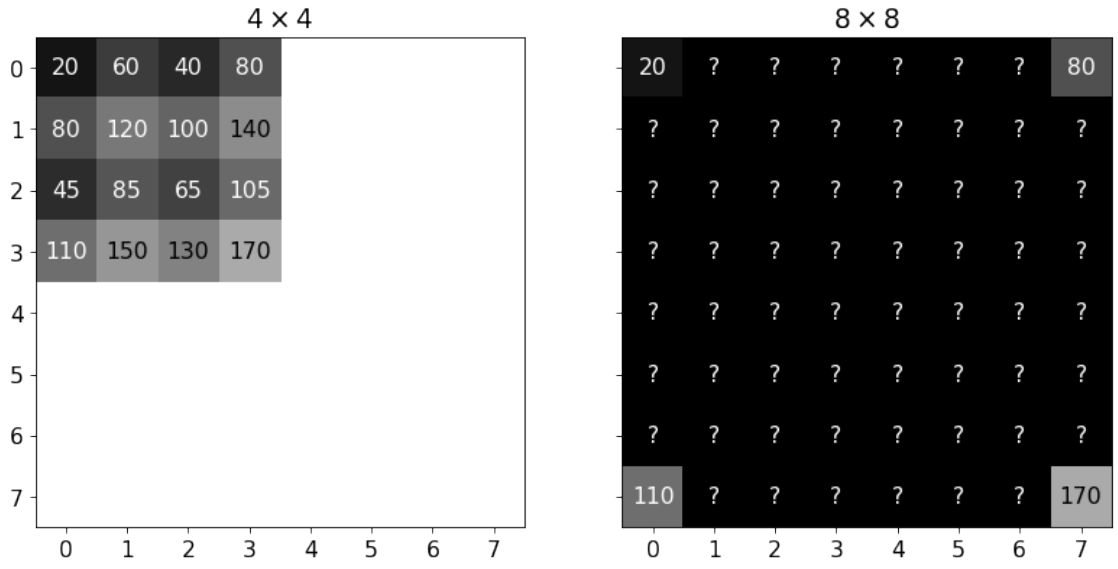
영상 확대

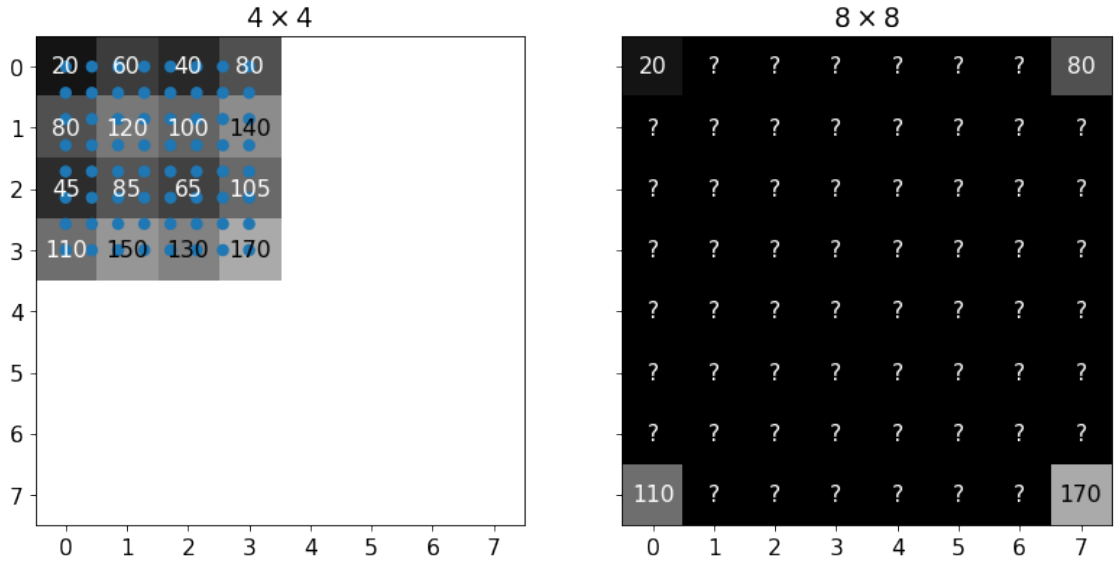

4 × 4
20 60 40 80
80 120 100 140
45 85 65 105
110 150 130 170


8 × 8
20 ? 60 ? 40 ? 80 ?
? ? ? ? ? ? ? ?
80 ? 120 ? 100 ? 140 ?
? ? ? ? ? ? ? ?
45 ? 85 ? 65 ? 105 ?
? ? ? ? ? ? ? ?
110 ? 150 ? 130 ? 170 ?
? ? ? ? ? ? ? ?


4 × 4
20 60 40 80
80 120 100 140
45 85 65 105
110 150 130 170


8 × 8
20 ? 60 ? 40 ? 80 ?
? ? ? ? ? ? ? ?
80 ? 120 ? 100 ? 140 ?
? ? ? ? ? ? ? ?
45 ? 85 ? 65 ? 105 ?
? ? ? ? ? ? ? ?
110 ? 150 ? 130 ? 170 ?
? ? ? ? ? ? ? ?

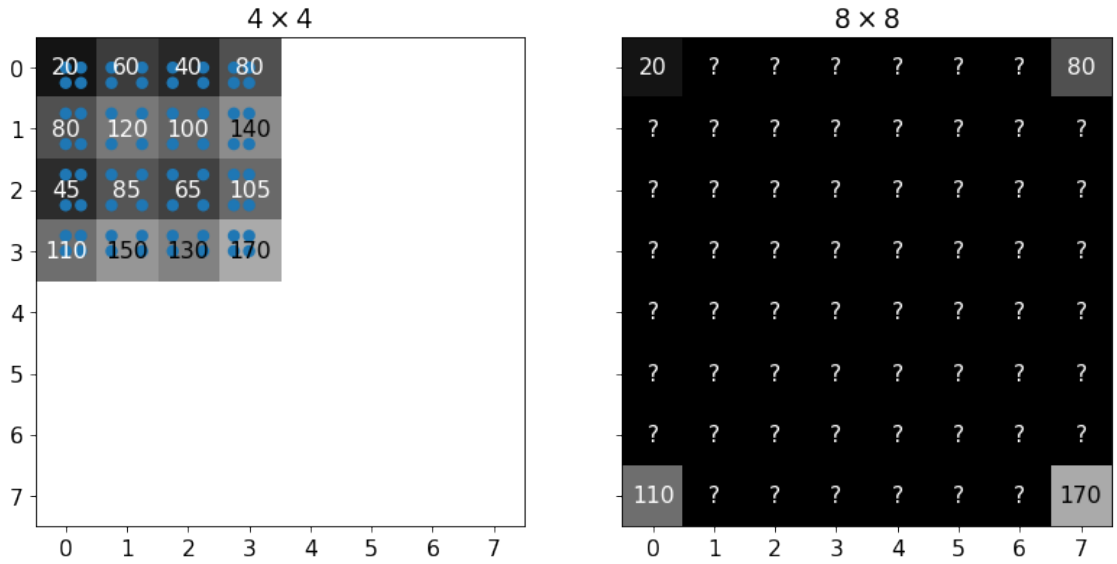




영상 축소와 확대

- 영상의 샘플 수를 변경하는 문제
- 크기가 변경된 영상의 화소 위치에 대응하는 원 영상의 화소 위치 결정
- 크기가 변경된 영상의 화소값 결정
- OpenCV: cv2.resize()

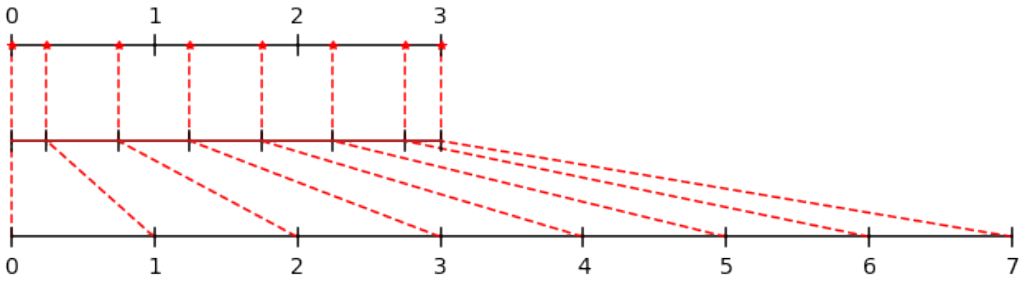
OpenCV의 영상 확대



1 차원 신호의 축소, 확대

OpenCV resize 함수의 보간 위치

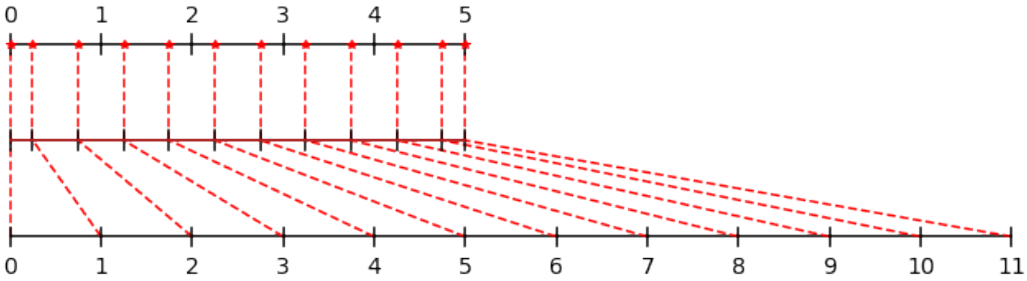
```
[0.   0.25 0.75 1.25 1.75 2.25 2.75 3.  ]
```



```
intervals:  [0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25]
```

[ ]:

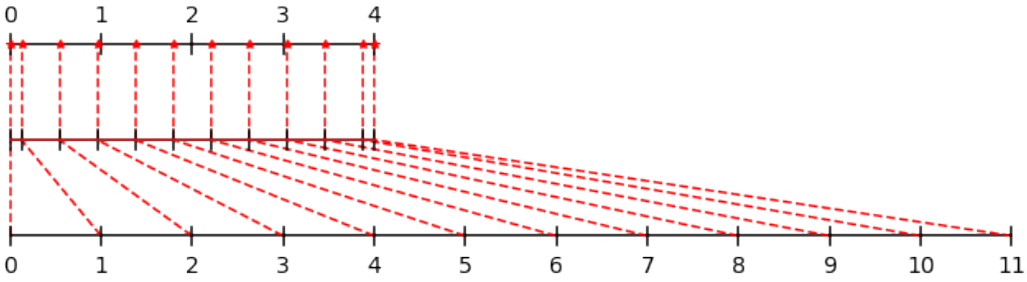
```
[0.   0.25 0.75 1.25 1.75 2.25 2.75 3.25 3.75 4.25 4.75 5.  ]
```




```
intervals:  [0.25, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.25]
```

[ ]:

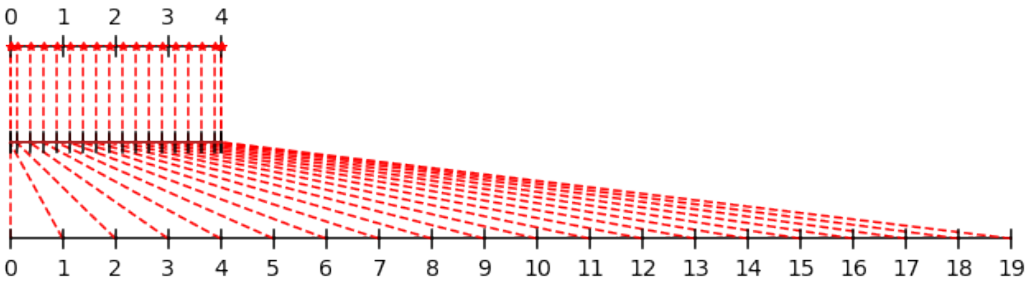
```
[0.        0.125     0.5416667 0.9583333 1.375     1.7916666 2.2083333
 2.625     3.0416667 3.4583333 3.875     4.        ]
```




```
intervals:  [0.125, 0.4166667, 0.41666663, 0.4166667, 0.41666663, 0.41666663,
0.41666675, 0.41666675, 0.4166665, 0.41666675, 0.125]
```

[ ]:

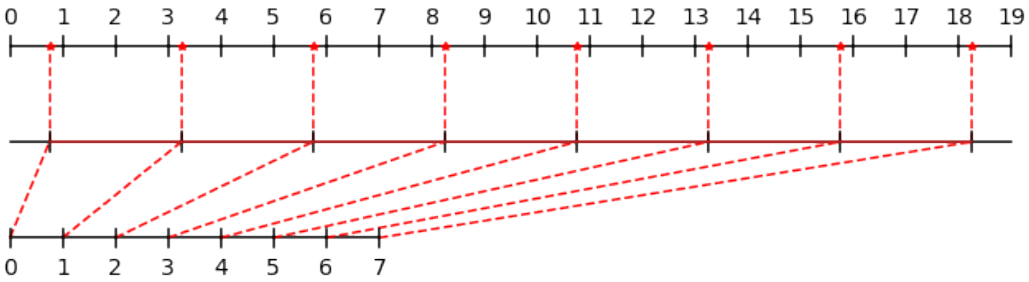
```
[0.    0.    0.125 0.375 0.625 0.875 1.125 1.375 1.625 1.875 2.125 2.375
 2.625 2.875 3.125 3.375 3.625 3.875 4.    4.   ]
```




```
intervals:  [0.0, 0.125, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25,
0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.125, 0.0]
```

[]:

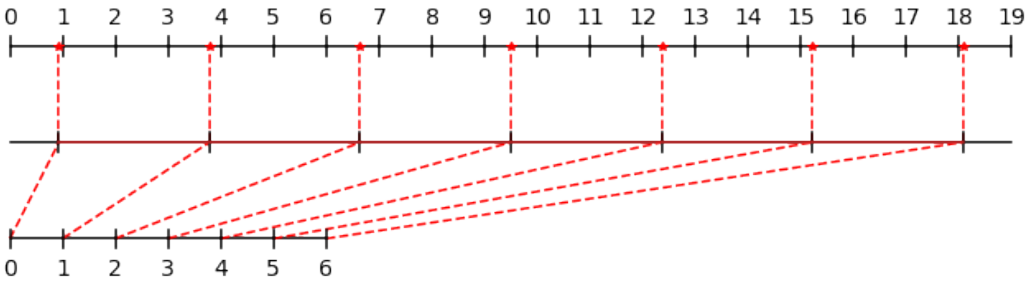
```
[ 0.75  3.25  5.75  8.25 10.75 13.25 15.75 18.25]
```




```
intervals:  [2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5]
```

[]:

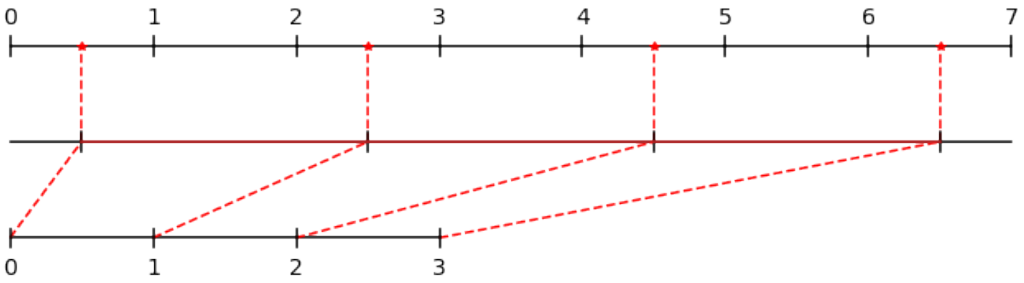
```
[ 0.9285714  3.7857144  6.642857   9.5        12.357142  15.214286
 18.071428 ]
```




```
intervals:  [2.857143, 2.8571427, 2.857143, 2.8571424, 2.8571434, 2.8571424]
```

[]:

```
[0.5 2.5 4.5 6.5]
```




```
intervals:  [2.0, 2.0, 2.0]
```

1.2 OpenCV의 resize 함수가 제공하는 보간 방법

[29]:

```
help(cv2.resize)
```

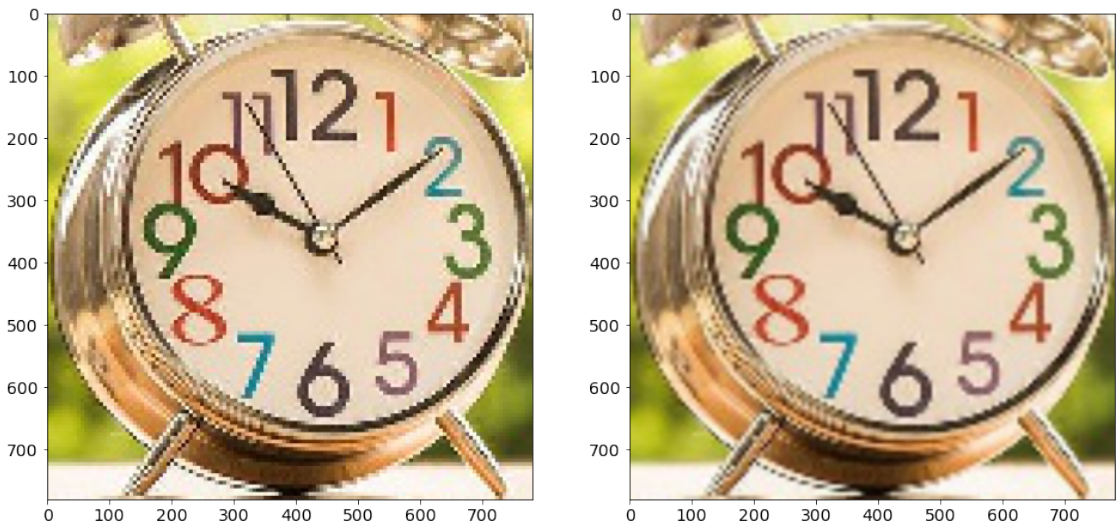
```
Help on built-in function resize:

resize(...)
    resize(src, dsize[, dst[, fx[, fy[, interpolation]]]]) -> dst
    .   @brief Resizes an image.
```

```
    .
    .   The function resize resizes the image src down to or up to the specified
size. Note that the
    .   initial dst type or size are not taken into account. Instead, the size
and type are derived from
    .   the `src`,`dsize`,`fx`, and `fy`. If you want to resize src so that it
fits the pre-created dst,
    .   you may call the function as follows:
    .   @code
    .       // explicitly specify dsize=dst.size(); fx and fy will be computed
from that.
    .       resize(src, dst, dst.size(), 0, 0, interpolation);
    .   @endcode
    .   If you want to decimate the image by factor of 2 in each direction, you
can call the function this
    .   way:
    .   @code
    .       // specify fx and fy and let the function compute the destination
image size.
    .       resize(src, dst, Size(), 0.5, 0.5, interpolation);
    .   @endcode
    .   To shrink an image, it will generally look best with #INTER_AREA
interpolation, whereas to
    .   enlarge an image, it will generally look best with c#INTER_CUBIC (slow)
or #INTER_LINEAR
    .   (faster but still looks OK).
    .
    .   @param src input image.
    .   @param dst output image; it has the size dsize (when it is non-zero) or
the size computed from
    .   src.size(), fx, and fy; the type of dst is the same as of src.
    .   @param dsize output image size; if it equals zero, it is computed as:
    .    \f[\texttt{dsize = Size(round(fx*src.cols), round(fy*src.rows))}\f]
    .    Either dsize or both fx and fy must be non-zero.
    .   @param fx scale factor along the horizontal axis; when it equals 0, it
is computed as
```

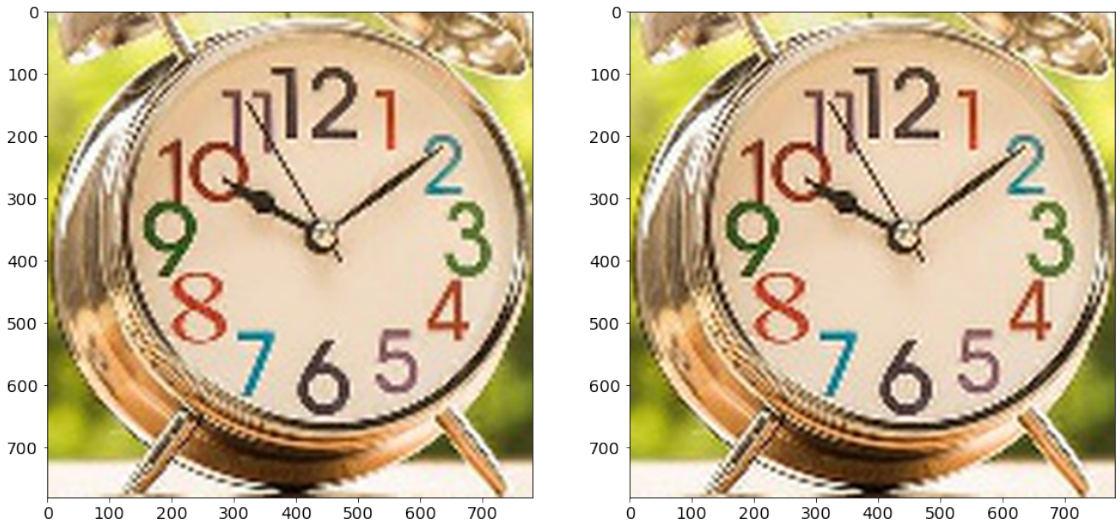
```
.   \f[\texttt{(double)dsize.width/src.cols}\f]
.   @param fy scale factor along the vertical axis; when it equals 0, it is
computed as
.   \f[\texttt{(double)dsize.height/src.rows}\f]
.   @param interpolation interpolation method, see #InterpolationFlags
.
.   @sa  warpAffine, warpPerspective, remap
```

- INTER_NEAREST: 최근접(nearest-neighbor) 보간. 처리 시간이 가장 짧음.
- INTER_LINEAR: 양방향 선형(bilinear) 보간, 기본값
- INTER_AREA: 화소 영역의 관계를 이용한 리샘플링. 영상 축소에 주로 사용되고, 영상 확대에 사용하면 INTER_NEAREST 방법과 유사한 효과
- INTER_CUBIC: 4x4 영역에서 양방향 cubic spline 보간.
- INTER_LANCZOS4: 8x8 영역에서 Lanczos 보간. 처리 시간이 가장 느림.

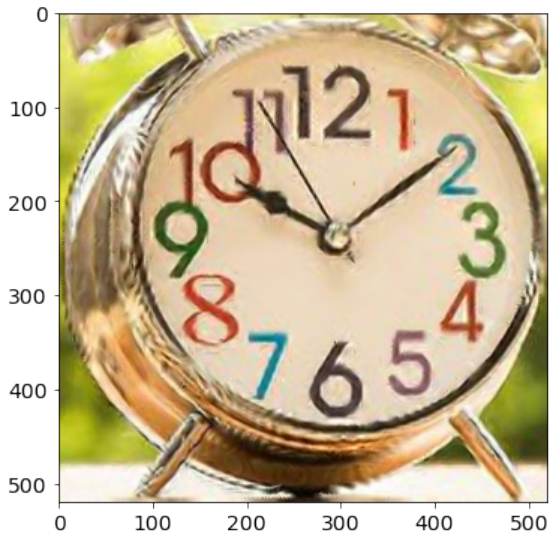
resize 함수를 이용하여 cv2.INTER_NEAREST**와** cv2.INTER_LINEAR**로** 6x6 **배 확대한 영상**



resize 함수를 이용하여 cv2.INTER_CUBIC**과** cv2.INTER_LANCZOS4**로** 6x6 **배 확대한 영상**



참고: edge **방향을 고려한 영상 보간 방법**



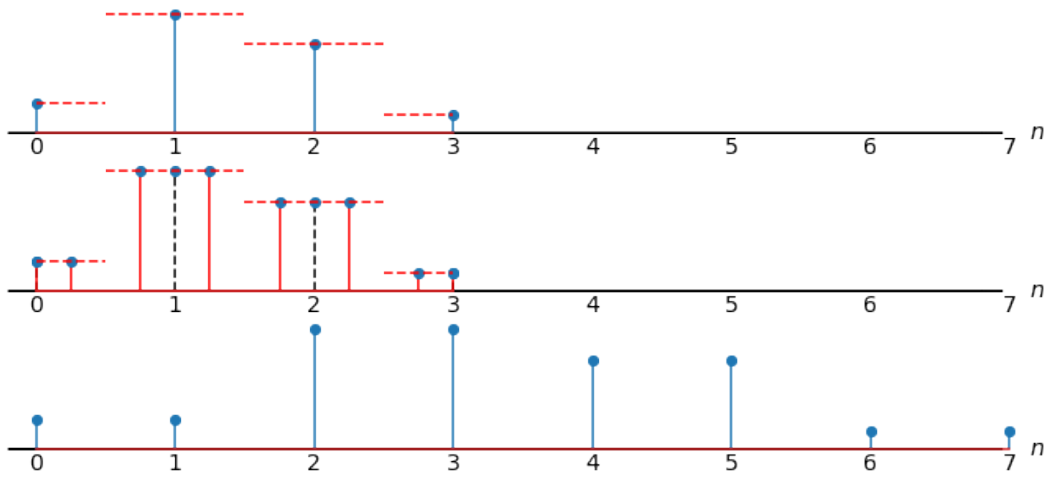
1.3 예: 일차원 신호의 확대 문제

- 확대된 신호의 일부 샘플들은 원래 신호에서 매치되는 샘플이 없다

OpenCV에서 사용하는 보간(interpolation) 방법

[15]:
```
x = np.array([20, 100, 75, 15]).reshape(1, 4)
x2 = cv2.resize(x, (8,1), interpolation=cv2.INTER_NEAREST)
x2
```

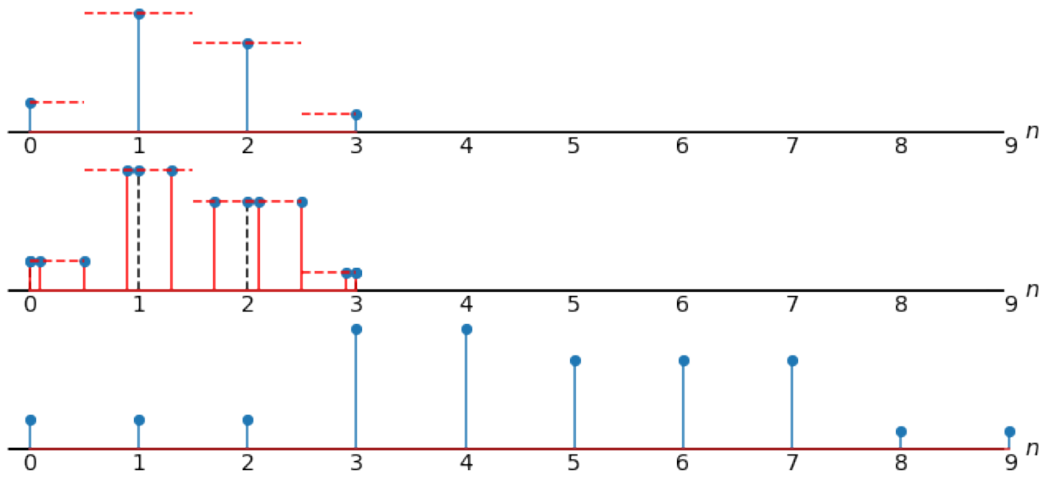
[15]:
```
array([[ 20,  20, 100, 100,  75,  75,  15,  15]], dtype=int32)
```



[ ]:
```
array([0.  , 0.25, 0.75, 1.25, 1.75, 2.25, 2.75, 3.  ])
```

[16]:
```
x = np.array([20, 100, 75, 15]).reshape(1, 4)
x2 = cv2.resize(x, (10,1), interpolation=cv2.INTER_NEAREST)
x2
```

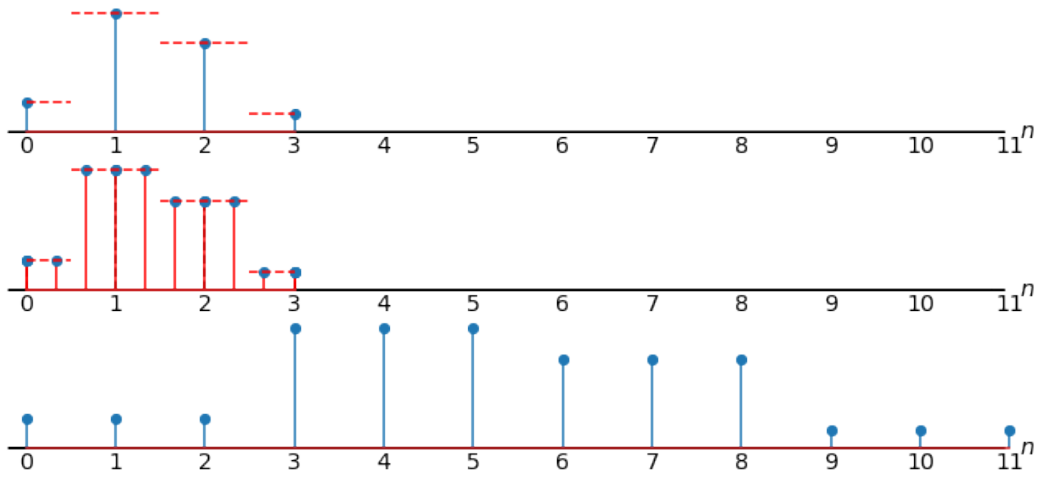
[16]:
```
array([[ 20,  20,  20, 100, 100,  75,  75,  75,  15,  15]], dtype=int32)
```



[6]:
```
array([0.        , 0.1       , 0.5       , 0.89999998, 1.29999995,
       1.70000005, 2.0999999 , 2.5       , 2.9000001 , 3.        ])
```

[17]:
```
x = np.array([20, 100, 75, 15]).reshape(1, 4)
x2 = cv2.resize(x, (12,1), interpolation=cv2.INTER_NEAREST)
x2
```

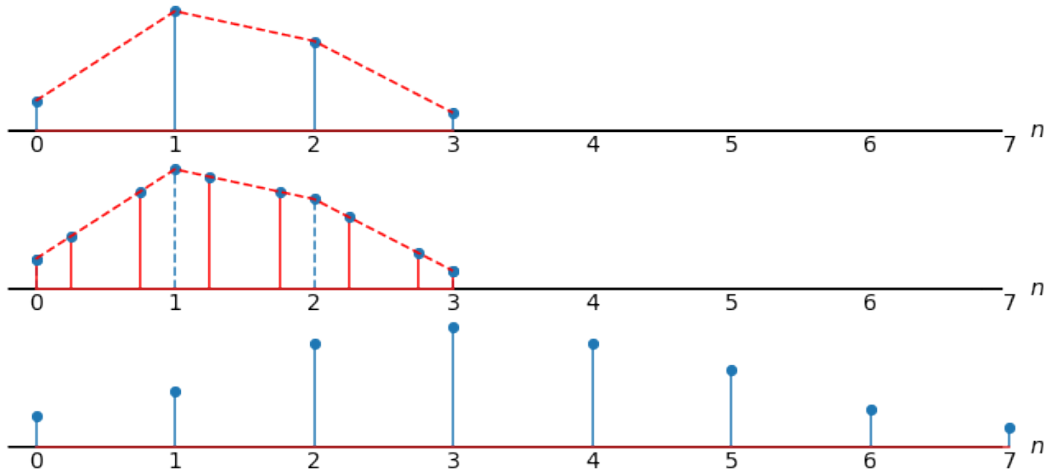
[17]:
```
array([[ 20,  20,  20, 100, 100, 100,  75,  75,  75,  15,  15,  15]],
      dtype=int32)
```



[7]:
```
array([0.        , 0.        , 0.33333334, 0.66666669, 1.        ,
       1.33333337, 1.66666663, 2.        , 2.33333325, 2.66666675,
       3.        , 3.        ])
```

[21]:
```
x = np.array([20, 100, 75, 15], dtype=float).reshape(1, 4)
x2 = cv2.resize(x, (8,1), interpolation=cv2.INTER_LINEAR)
x2
```

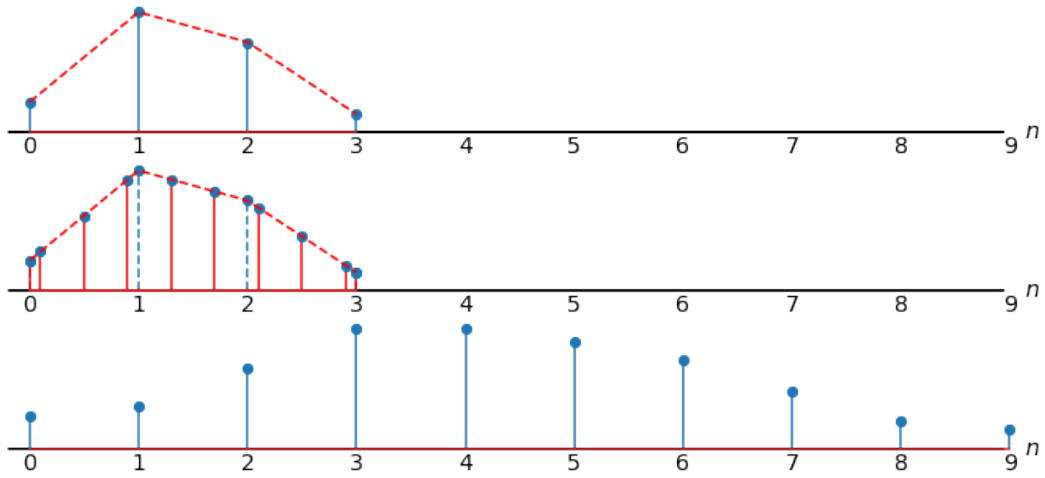
[21]:
```
array([[20.  , 40.  , 80.  , 93.75, 81.25, 60.  , 30.  , 15.  ]])
```



[9]:
```
array([0.  , 0.25, 0.75, 1.25, 1.75, 2.25, 2.75, 3.  ])
```

[22]:
```
x = np.array([20, 100, 75, 15], dtype=float).reshape(1, 4)
x2 = cv2.resize(x, (10,1), interpolation=cv2.INTER_LINEAR)
x2
```

[22]:
```
array([[20.        , 27.99999967, 60.        , 91.99999809, 92.50000119,
        82.49999881, 69.00000572, 45.        , 20.99999428, 15.        ]])
```



[8]:
```
array([0.        , 0.1       , 0.5       , 0.89999998, 1.29999995,
       1.70000005, 2.0999999 , 2.5       , 2.9000001 , 3.        ])
```

양선형 보간