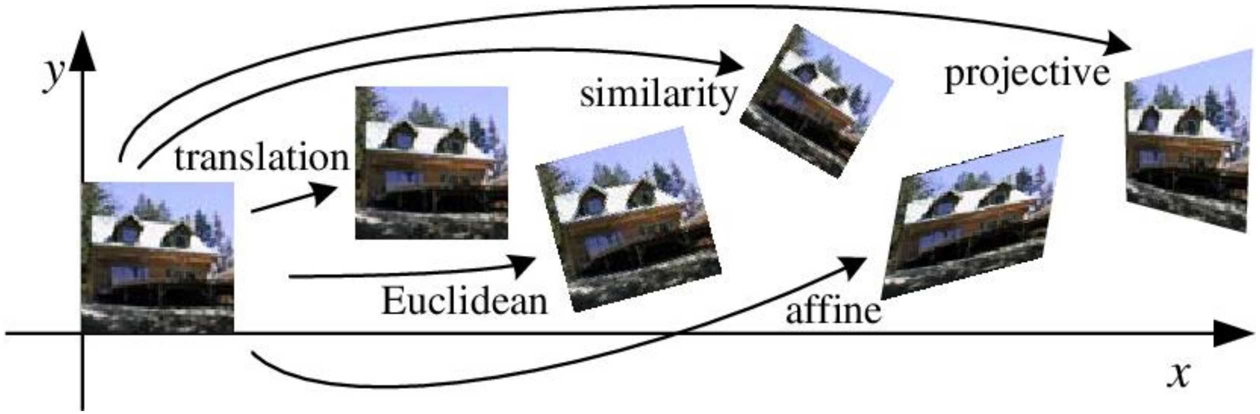
Computer Vision

CVI620

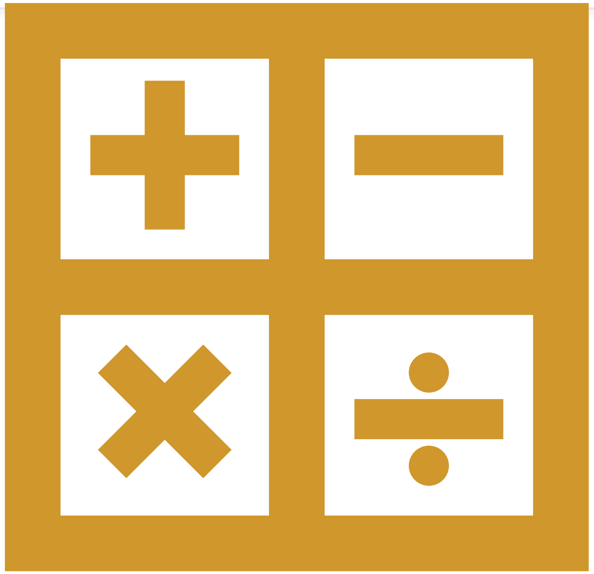
Session 6
01/2025

Image Geometric Transformations



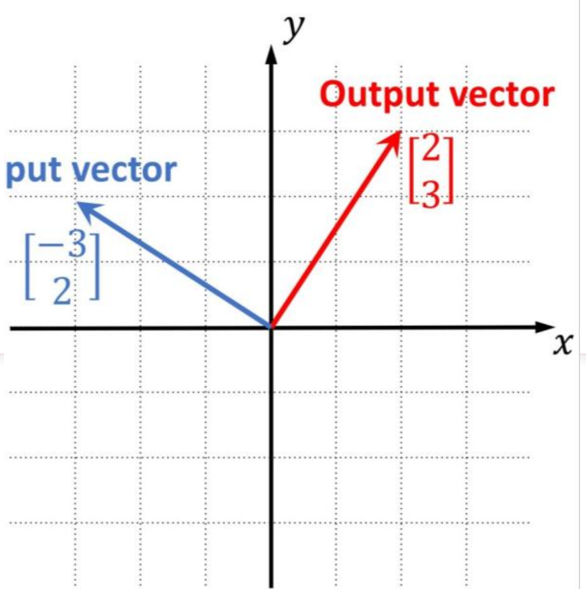
Question?

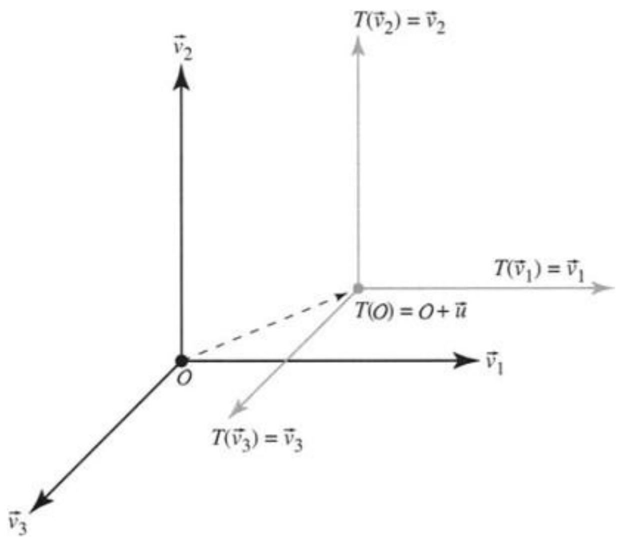
- What do we mean when we had matrix multiplication?
- Why did we learn vector/matrix multiplication/division/addition,…?
- What was the use-case?
- Are we just doing some calculations?



Tip

- If you want to understand any mathematical calculation, imagine it in the space.
- The intuition behind matrix calculations is transformation what do we mean?
- If we are multiplying something to a vector, it means we are changing the vector in the space (this applies to any transformation)
- In transformations, we sometimes change the vector or the space itself

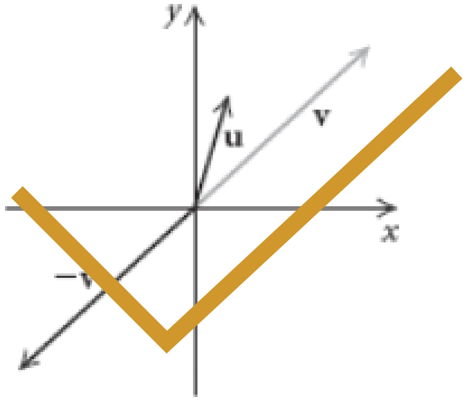




Geometric Transformation

- Mathematical operation that changes the position, size, orientation, or shape of a geometric object in a space

$$Ax = \cancel{\begin{bmatrix} 6 & 2 & 4 \\ -1 & 4 & 3 \\ -2 & 9 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ -2 \\ 1 \end{bmatrix}} = \begin{bmatrix} 24 \\ -9 \\ 23 \end{bmatrix}$$

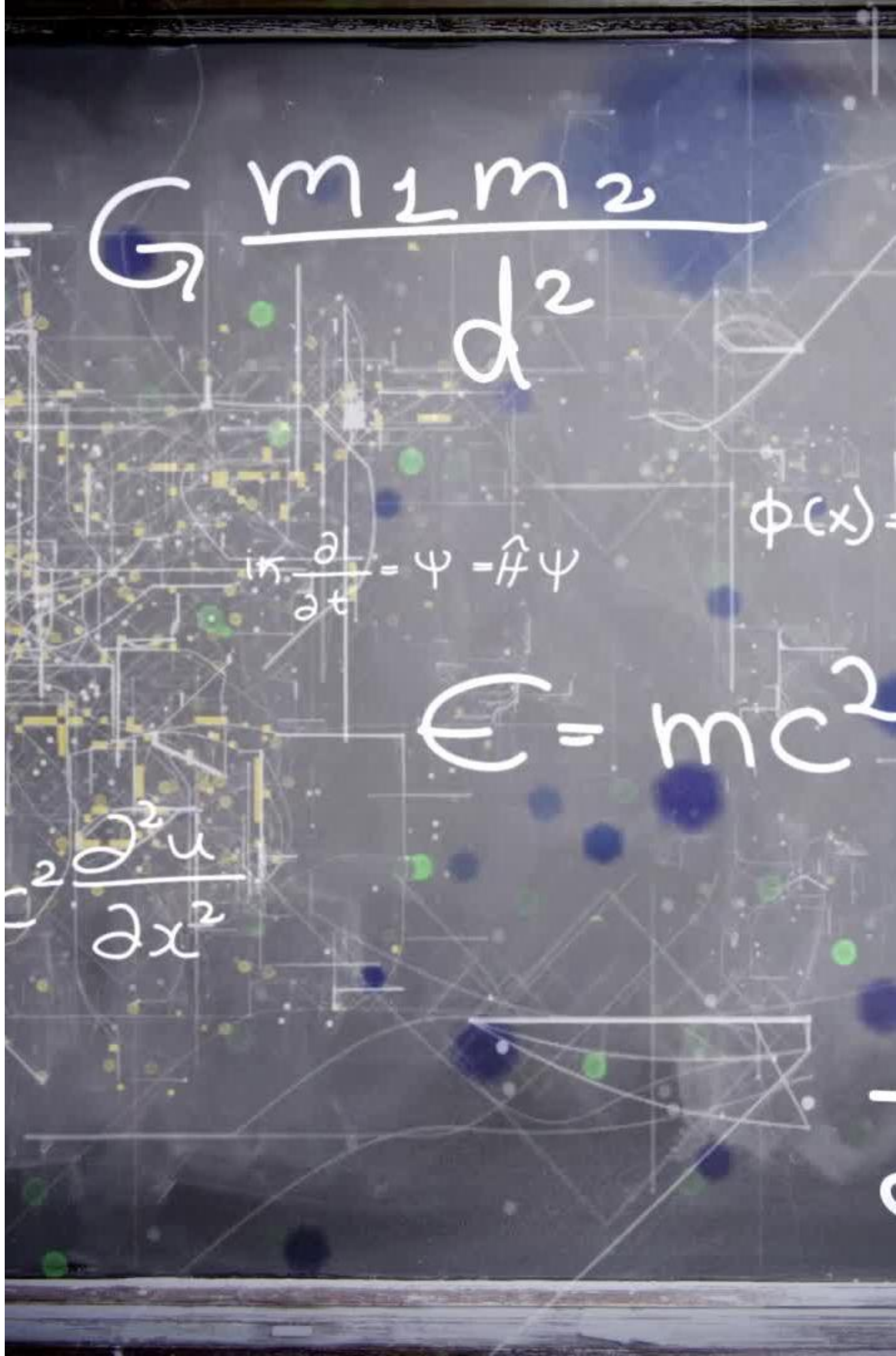


Sketch −v.

Image Transformation

- Any transformation in math or physics is basically doable with a matrix operation
- The same in Computer Vision since images are vectors or matrices

- Rotation: Certain matrices rotate the vector around an origin.
- Shearing: Changes the shape by shifting components of the vector.
- Reflection: Flips the vector across an axis or plane.



Resize

- Resizing adjusts the dimensions of an image (width and height) while maintaining its visual content.
- Normalization, Efficiency, Aspect Ratio Preservation

UpScale

DownScale

Question?

What is happening behind the scene?

Answer!

Just a matrix operation is being applied!

Resizing Methodology

Either new pixels are created, or some pixels are removed

The key challenge in resizing is determining how to map the pixel values from the original image to the resized image in a way that maintains visual quality.

Interpolation

It is the algorithm of that **matrix operation**

In math: Interpolation means finding a new set of values for a function given a set of prior values for the same function.

In Computer Vision: A techniques to approximate the color and intensity values of pixels in the resized image.

```
1/2  (src: MatLike, dsize: Size | None, dst: MatLike |
     None = ..., fx: float = ..., fy: float = ...,
     interpolation: int = ...) -> MatLike
cv2.resize()
```

cv2.resize

```
cv2.resize(src, dsize, dst=None, fx=0, fy=0, interpolation=cv2.INTER_LINEAR)
```

- src: source image to be resized
- dsize: desired size of the destination image – tuple
  must be specified if both fx and fy are zero (or not specified). If dsize is set to (0, 0), then fx and fy must be specified to determine the size.
- dst: destination image – usually ignored in Python
- fx, fy: multiplier that scales the width/height of the image
- interpolation: default value is cv2.INTER_LINEAR

Upsampling

Interpolation is used to estimate the color and intensity values of the new pixels based on the existing pixels.

- Area interpolation
- Bilinear interpolation
- Nearest-neighbor interpolation
- Bicubic interpolation
- Lanczos interpolation

Nearest Neighbor

We can apply one of these methods based on **N nearest neighbor**

Averaging	Max choice	Max
Min	Linear/non-linear function based on nearest neighbors (General form of Bilinear interpolation)	• • •

Nearest Neighbor

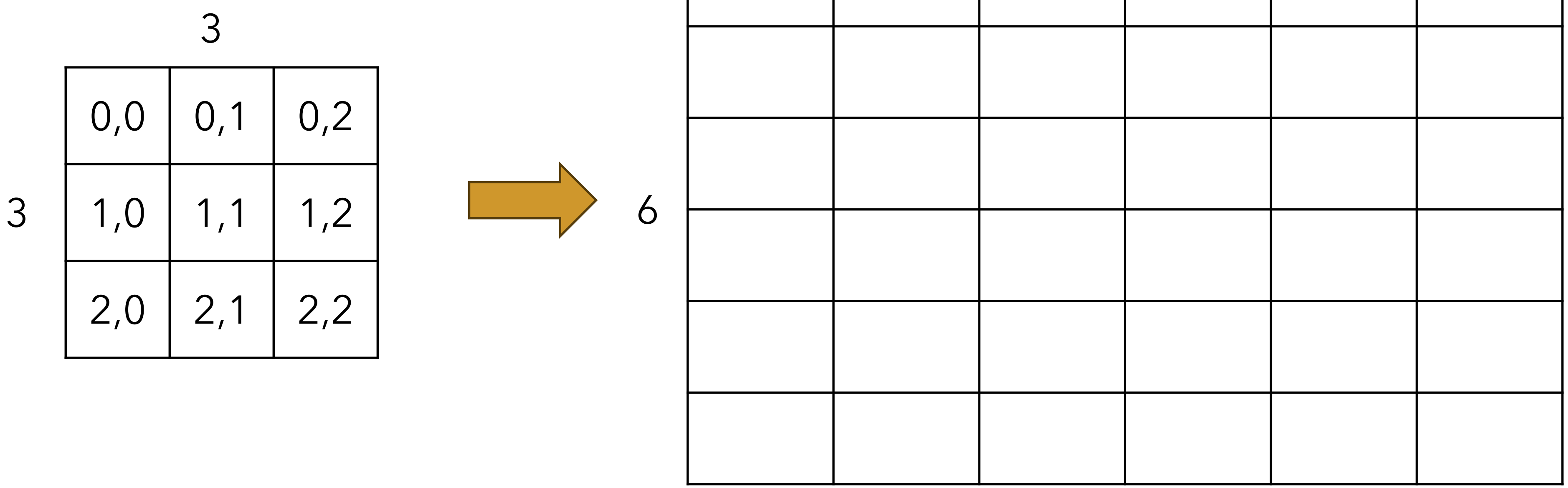
selects the value of the nearest pixel from the original image and assigns it to the corresponding pixel in the resized image

N is a hyperparameter

Nearest Neighbor – Max Choice

Numbers are coordinates
and not pixel values

N=1



Nearest Neighbor - Max Choice

Numbers are coordinates
and not pixel values

N=1

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0		0,1		0,2	
1,0		1,1		1,2	
2,0		2,1		2,2	

Nearest Neighbor - Max Choice

Numbers are coordinates
and not pixel values

N=1

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,0	0,1		0,2	
0,0	0,0				
1,0		1,1		1,2	
2,0		2,1		2,2	

Nearest Neighbor - Max Choice

Numbers are coordinates
and not pixel values

N=1

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,0	0,1	0,1	0,2	0,2
0,0	0,0	0,1	0,1	0,2	0,2
1,0	1,0	1,1	1,1	1,2	1,2
1,0	1,0	1,1	1,1	1,2	1,2
2,0	2,0	2,0	2,0	2,2	2,2
2,0	2,0	2,0	2,0	2,2	2,2

Nearest Neighbor - Averaging

Numbers are coordinates
and not pixel values

N=2

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,avg(0,1)	0,1		0,2	
1,0		1,1		1,2	
2,0		2,1		2,2	

Nearest Neighbor - Averaging

Numbers are coordinates
and not pixel values

N=2

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,avg(0,1)	0,1	0,avg(1,2)	0,2	0,avg(2,2)
1,0		1,1		1,2	
2,0		2,1		2,2	

Nearest Neighbor - Averaging

Numbers are coordinates
and not pixel values

N=2

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,avg(0,1)	0,1	0,avg(1,2)	0,2	0,avg(2,2)
avg(0,1), 0					
1,0		1,1		1,2	
2,0		2,1		2,2	

Nearest Neighbor - Averaging

Numbers are coordinates
and not pixel values

N=2

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,avg(0,1)	0,1	0,avg(1,2)	0,2	0,avg(2,2)
avg(0,1), 0	avg(0,1), avg(0,1)				
1,0		1,1		1,2	
2,0		2,1		2,2	

Nearest Neighbor - Averaging

Numbers are coordinates
and not pixel values

N=2

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,avg(0,1)	0,1	0,avg(1,2)	0,2	0,avg(2,2)
avg(0,1), 0	avg(0,1), avg(0,1)	x			
1,0		1,1		1,2	
2,0		2,1		2,2	

Nearest Neighbor - Averaging

Numbers are coordinates
and not pixel values

N=2

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,avg(0,1)	0,1	0,avg(1,2)	0,2	0,avg(2,2)
avg(0,1), 0	avg(0,1), avg(0,1)	x	x	x	x
1,0	x	1,1	x	1,2	x
x	x	x	x	x	x
2,0	x	2,1	x	2,2	x
x	x	x	x	x	x

Nearest Neighbor - Averaging

Gray-Scale
Same concept for colored images

N=2

180	255	9
56	127	255
0	73	0

180+255 / 2 =217.5

180	217	255		9	
56		127		255	
0		73		0	

Nearest Neighbor - Averaging

Gray-Scale
Same concept for colored images

N=2

180	255	9
56	127	255
0	73	0

255+9 / 2 =132

180	217	255	132	9	
56		127		255	
0		73		0	

Nearest Neighbor - Averaging

Gray-Scale
Same concept for colored images

N=2

180	255	9
56	127	255
0	73	0

180	217	255	132	9	9
56		127		255	
0		73		0	

9+9 / 2 = 9

Nearest Neighbor - Averaging

Gray-Scale
Same concept for colored images

N=2

180	255	9
56	127	255
0	73	0

180+56 / 2 = 118

180	217	255	132	9	9
118					
56		127		255	
0		73		0	

Nearest Neighbor - Averaging

Gray-Scale
Same concept for colored images

N=2

180	255	9
56	127	255
0	73	0

180	217	255	132	9	9
118					
56		127		255	
0		73		0	

180+56 / 2 = 118

Nearest Neighbor - Averaging

Gray-Scale
Same concept for colored images

N=2

180	255	9
56	127	255
0	73	0

180	217	255	132	9	9
118	167				
56		127		255	
0		73		0	

217+118 / 2 = 167.5

Nearest Neighbor - Averaging

Gray-Scale
Same concept for colored images

N=2

180	255	9
56	127	255
0	73	0

180	217	255	132	9	9
118	167	211			
56		127		255	
0		73		0	

255+167 / 2 = 211

Nearest Neighbor - Averaging

Gray-Scale
Same concept for colored images

N=2

180	255	9
56	127	255
0	73	0

→

180	217	255	132	9	9
118	167	211	171	90	49
56	111	127	149	255	152
28					
0		73		0	

Nearest Neighbor - Averaging

Gray-Scale
Same concept for colored images

N=2

180	255	9
56	127	255
0	73	0

180	217	255	132	9	9
118	167	211	171	90	49
56	111	127	149	255	152
28	69	98	123	189	170
0	34	73	98	0	85
17	25	49	73	36	60

Nearest Neighbor - Averaging

Default and only option is N=1

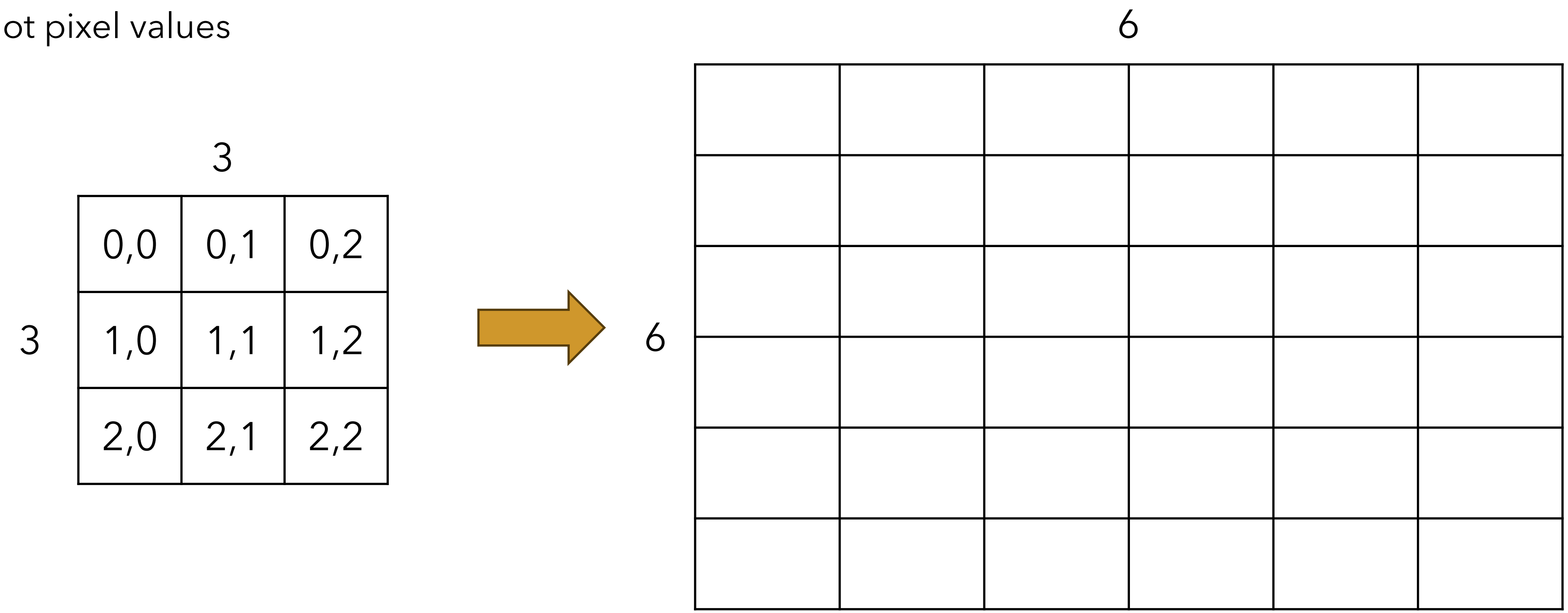
```
import cv2

image = cv2.imread('Lenna.png')
resized_image = cv2.resize(image, (image.shape[1]*3, image.shape[0]*3), interpolation=cv2.INTER_NEAREST)

cv2.imwrite('Lenna_Resized_Image.png', resized_image)
cv2.imshow('Lenna_Resized_Image', resized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```


Area Interpolation

Numbers are coordinates and not pixel values



Area Interpolation

Numbers are coordinates
and not pixel values

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,0				
0,0	0,0				

Area Interpolation

Numbers are coordinates
and not pixel values

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0	0,0	0,1	0,1	0,2	0,2
0,0	0,0	0,1	0,1	0,2	0,2
1,0	1,0	1,1	1,1	1,2	1,2
1,0	1,0	1,1	1,1	1,2	1,2
2,0	2,0	2,0	2,0	2,2	2,2
2,0	2,0	2,0	2,0	2,2	2,2

Area Interpolation

Gray-Scale
Same concept for colored images

180	255	9
56	127	255
0	73	0

180	180	255	255	9	9
180	180	255	255	9	9
56	56	127	127	255	255
56	56	127	127	255	255
0	0	73	73	0	0
0	0	73	73	0	0

Area Interpolation

```python
import cv2

image = cv2.imread('Lenna.png')
resized_image = cv2.resize(image, (image.shape[0]*3, image.shape[1]*3), interpolation=cv2.INTER_AREA)

cv2.imwrite('Lenna_Resized_Image_Averaging.png', resized_image)
cv2.imshow('Lenna_Resized_Image_Averaging', resized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Algorithm

- How would you Python code the area interpolation method if you did not have OpenCV?

Algorithm

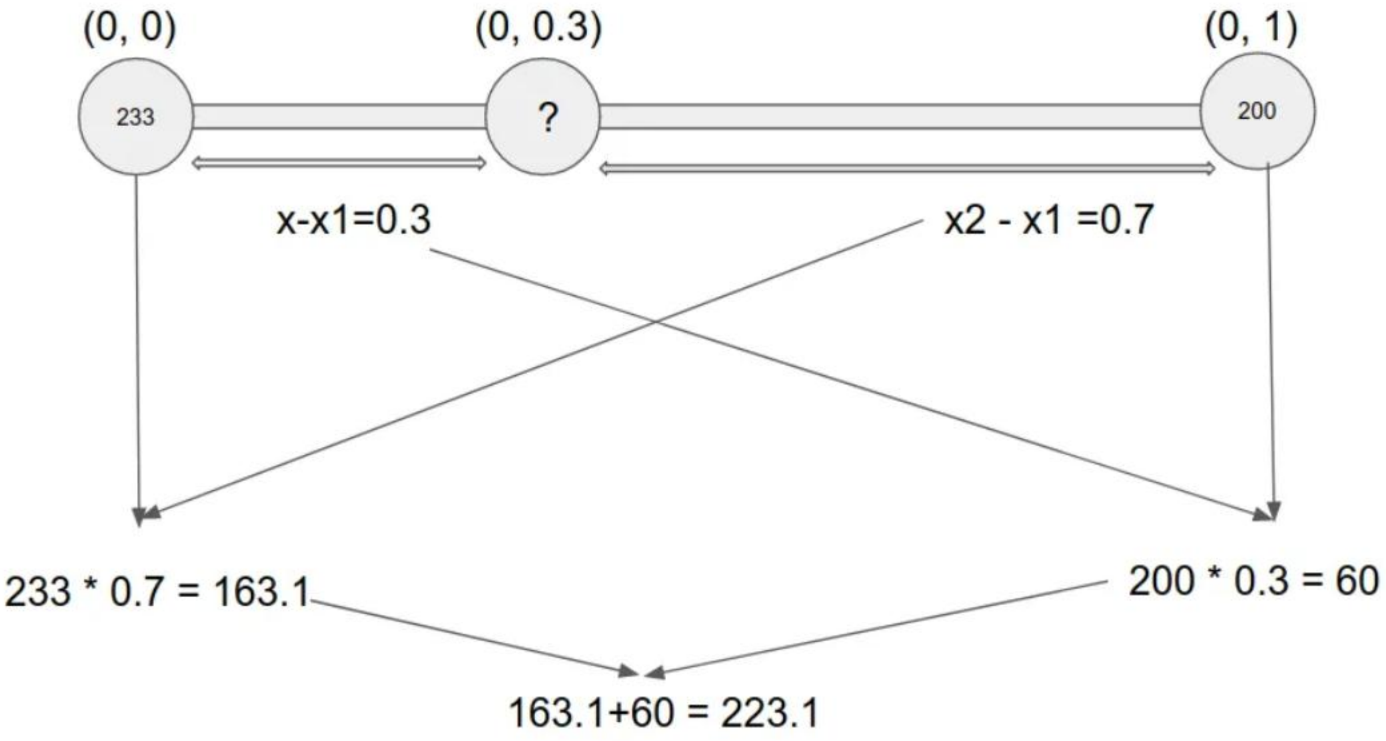
```
half_upsclaled_img[0:height:2, :, :] = resized_img[:, :, :]
half_upsclaled_img[1:height:2, :, :] = resized_img[:, :, :]

upsclaled_img[:, 0:width:2, :] = half_upsclaled_img[:, :, :]
upsclaled_img[:, 1:width:2, :] = half_upsclaled_img[:, :, :]
```

Bilinear Interpolation

$$I_{new} = \frac{x2 - x}{x2 - x1} * I_1 + \frac{x - x1}{x2 - x1} * I_2$$

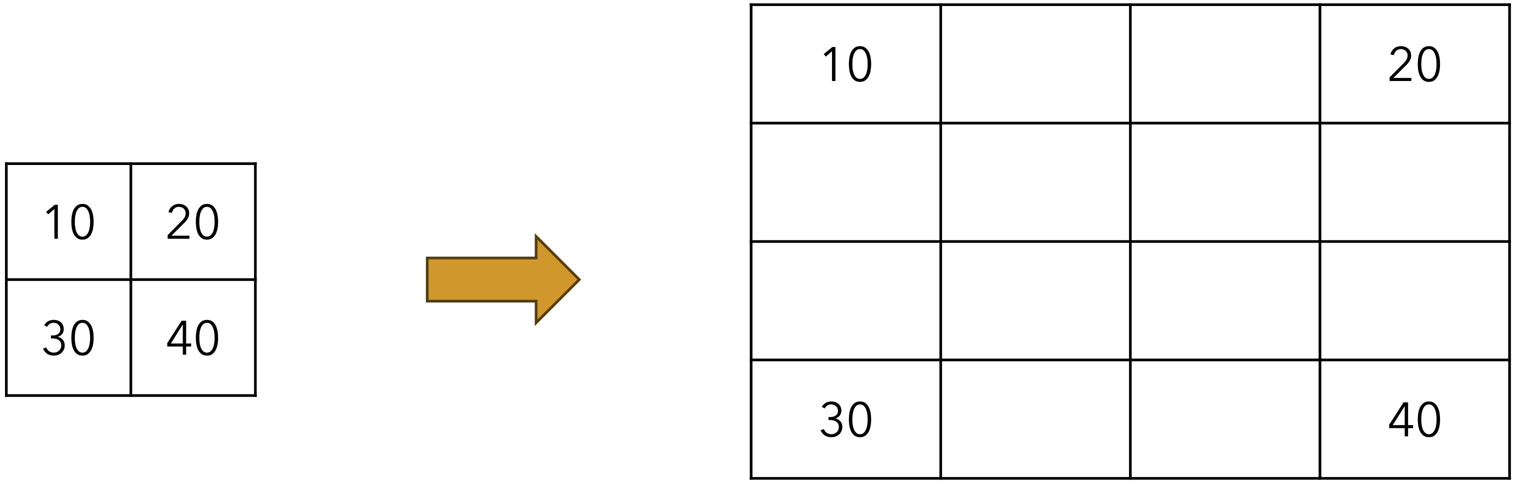
where x1 ≥ x ≥ x2



I1=233, I2=200 and I_new=223.1

Bilinear Interpolation

Initialization and filling pixels are hyperparameters

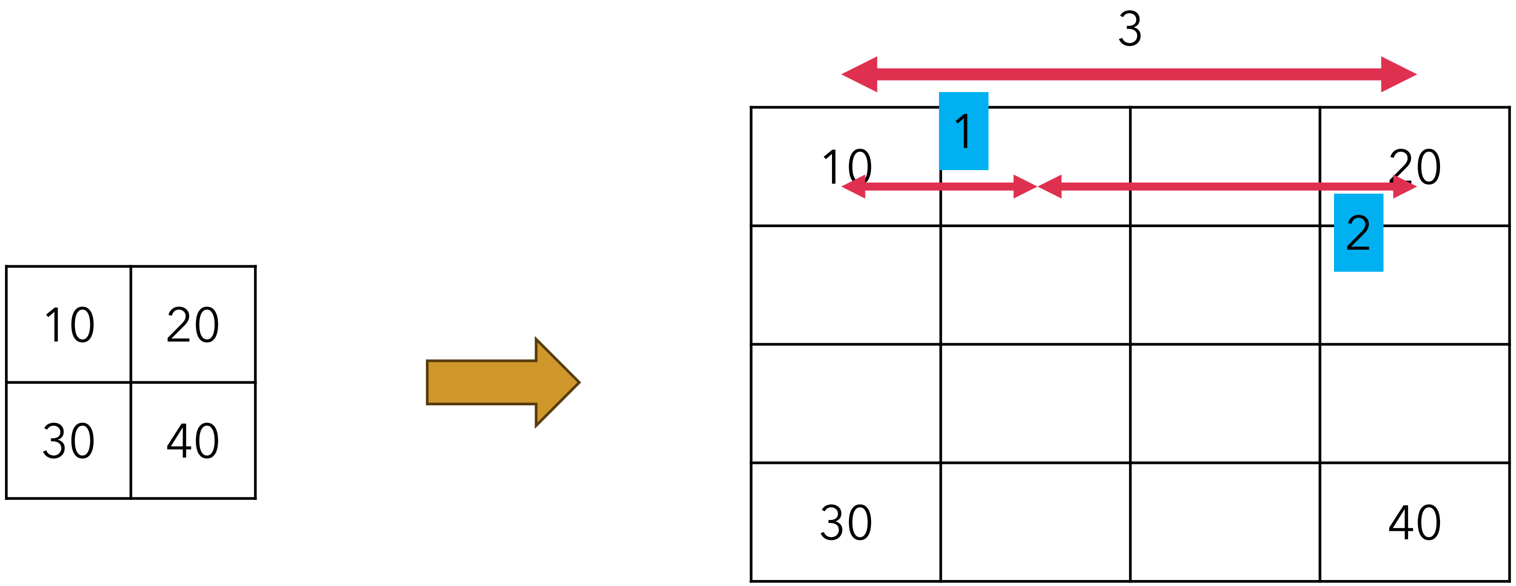


Bilinear Interpolation

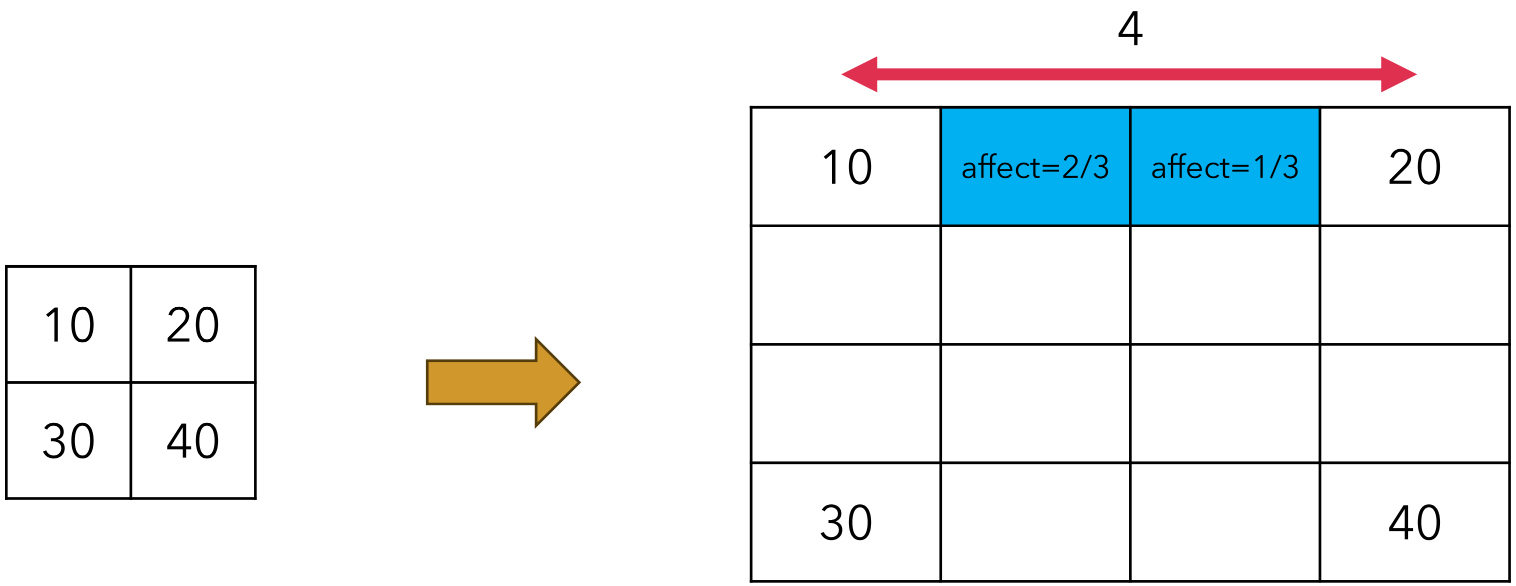
10	20
30	40

	(0,0)	(0,1)	(0,2)	(0,3)
(0,0)	10			20
(0,1)				
(0,2)				
(0,3)	30			40

Bilinear Interpolation



Bilinear Interpolation



Bilinear Interpolation

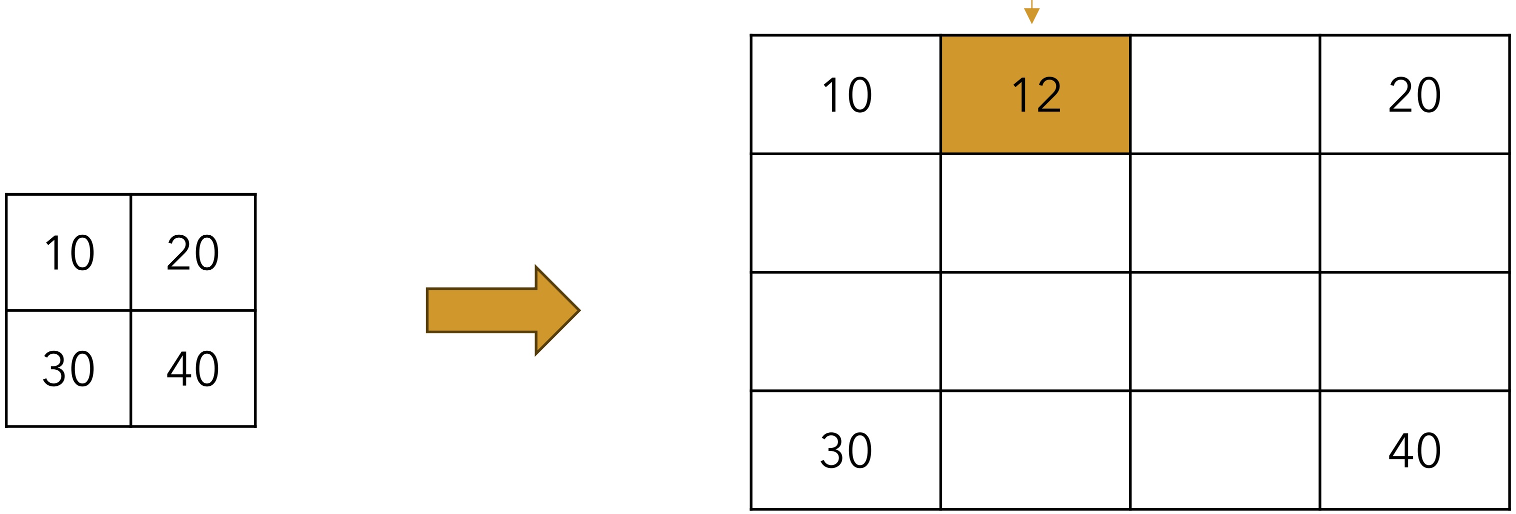
round [(2/3 * 10) + (1/3 *20) / 2] = round[(6.6 + 6.6) / 2] = 12

10	20
30	40

10	12		20
30			40

Bilinear Interpolation

round [(1/3 * 10) + (2/3 *20) / 2] = round[(6.6 + 6.6) / 2] = 12



Be careful that in the OpenCv's implementation, the distance is 4 instead of 3!!

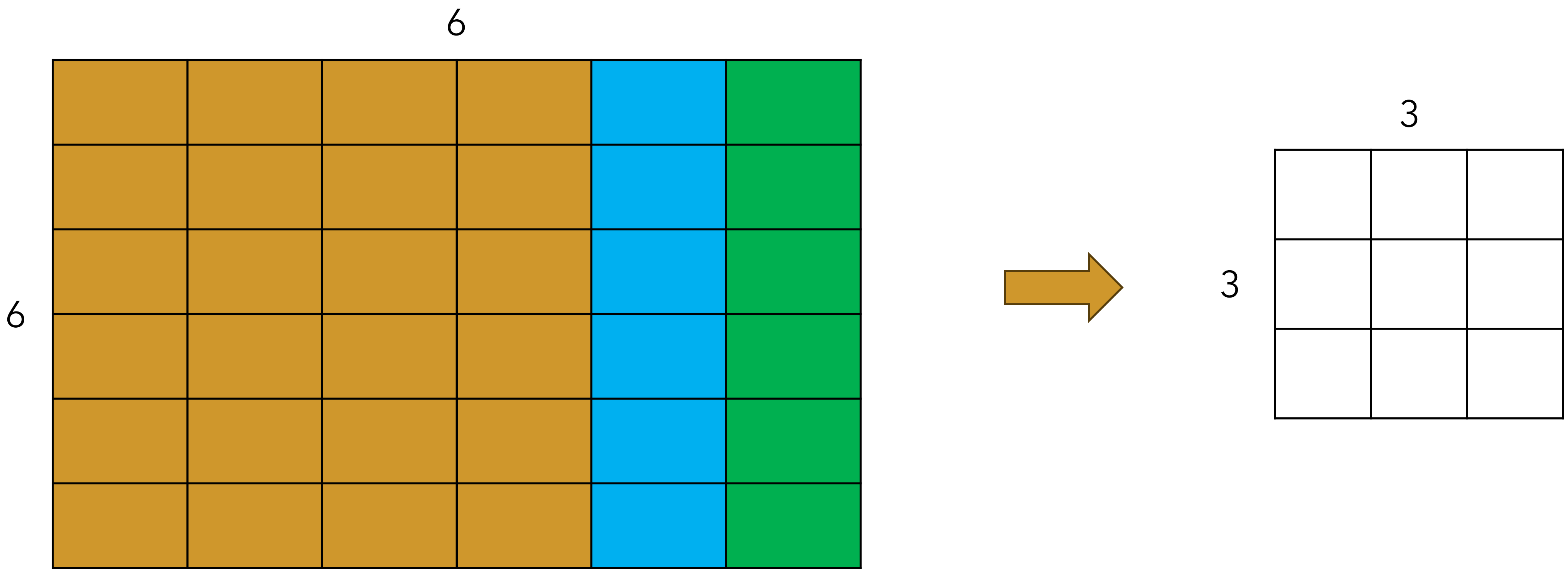
Enumerator	
INTER_NEAREST Python: cv.INTER_NEAREST	nearest neighbor interpolation
INTER_LINEAR Python: cv.INTER_LINEAR	bilinear interpolation
INTER_CUBIC Python: cv.INTER_CUBIC	bicubic interpolation
INTER_AREA Python: cv.INTER_AREA	resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire'-free results. But when the image is zoomed, it is similar to the INTER_NEAREST method.
INTER_LANCZOS4 Python: cv.INTER_LANCZOS4	Lanczos interpolation over 8x8 neighborhood
INTER_LINEAR_EXACT Python: cv.INTER_LINEAR_EXACT	Bit exact bilinear interpolation
INTER_NEAREST_EXACT Python: cv.INTER_NEAREST_EXACT	Bit exact nearest neighbor interpolation. This will produce same results as the nearest neighbor method in PIL, scikit-image or Matlab.
INTER_MAX Python: cv.INTER_MAX	mask for interpolation codes
WARP_FILL_OUTLIERS Python: cv.WARP_FILL_OUTLIERS	flag, fills all of the destination image pixels. If some of them correspond to outliers in the source image, they are set to zero
WARP_INVERSE_MAP Python: cv.WARP_INVERSE_MAP	flag, inverse transformation For example, linearPolar or logPolar transforms: • flag is **not** set: $dst(\rho, \phi) = src(x, y)$ • flag is set: $dst(x, y) = src(\rho, \phi)$

Downsamlping

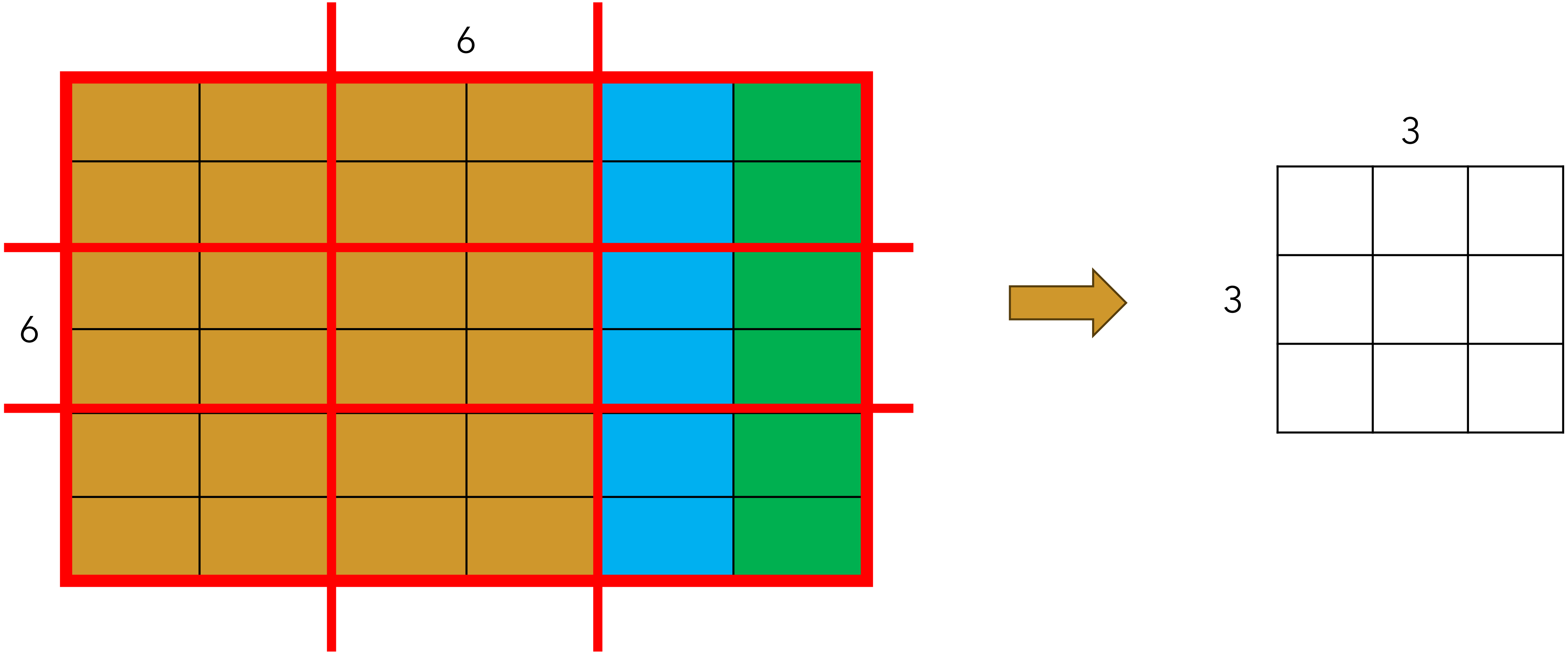
Interpolation is used to estimate the color and intensity values of the new pixels based on the existing pixels.

- Nearest-neighbor
- Bilinear interpolation
- Area interpolation
- Bicubic interpolation
- Lanczos interpolation

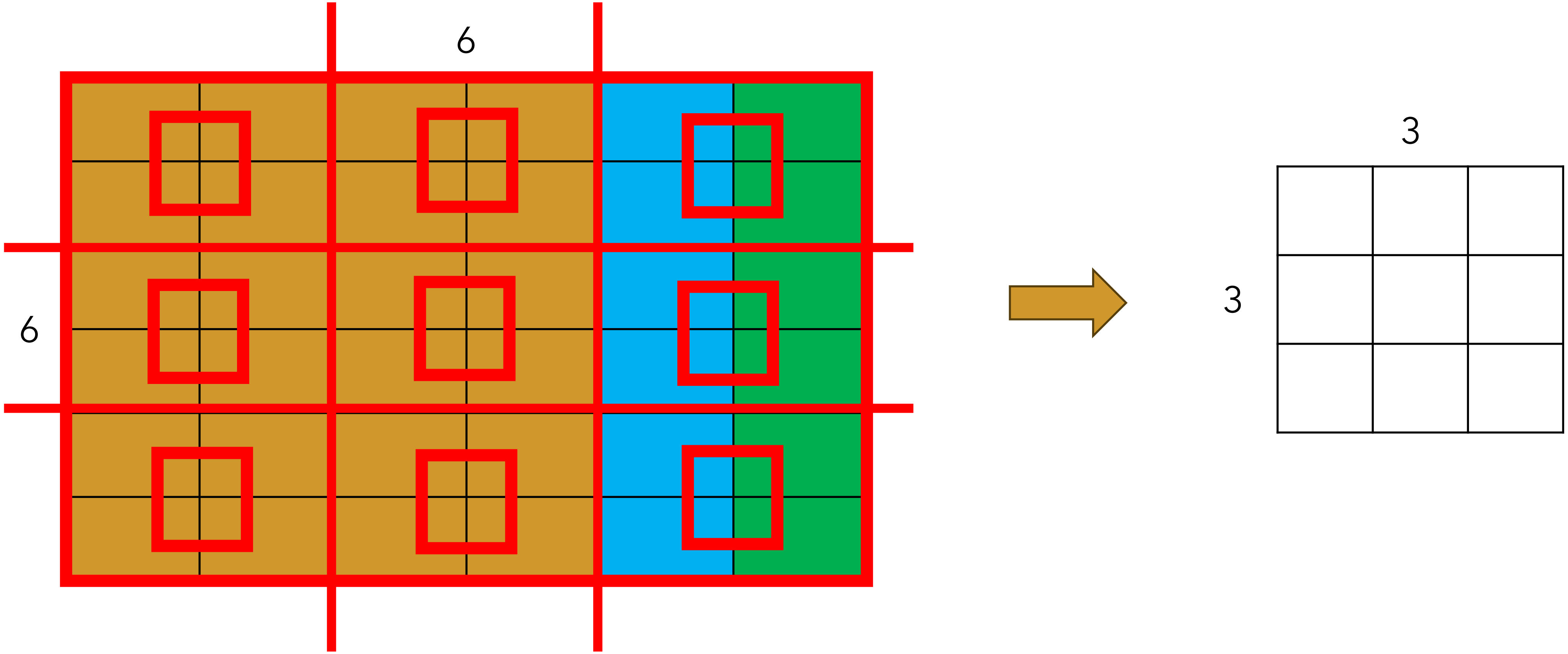
Nearest Neighbor – Max Choice



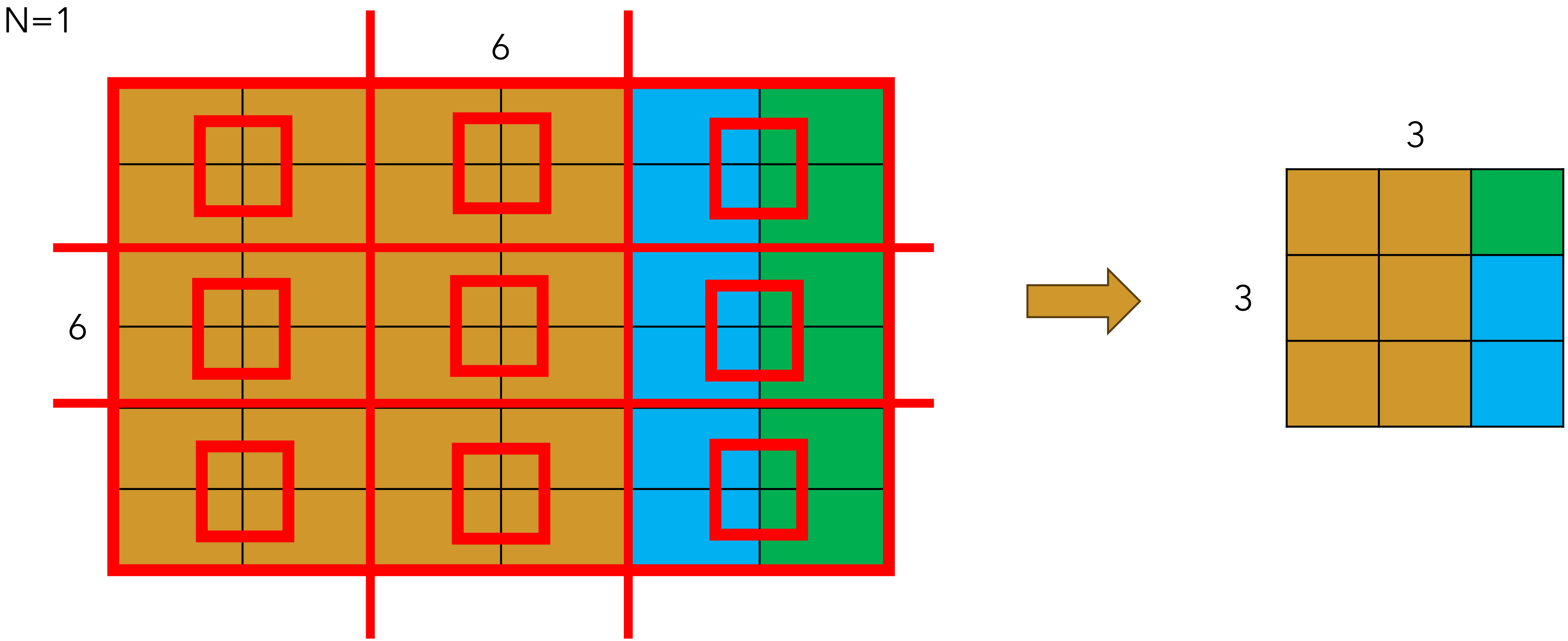
Nearest Neighbor - Max Choice



Nearest Neighbor - Max Choice



Nearest Neighbor - Max Choice



Bilinear Filtering