

# The Ancient Secrets



# Computer Vision

# Logistics:

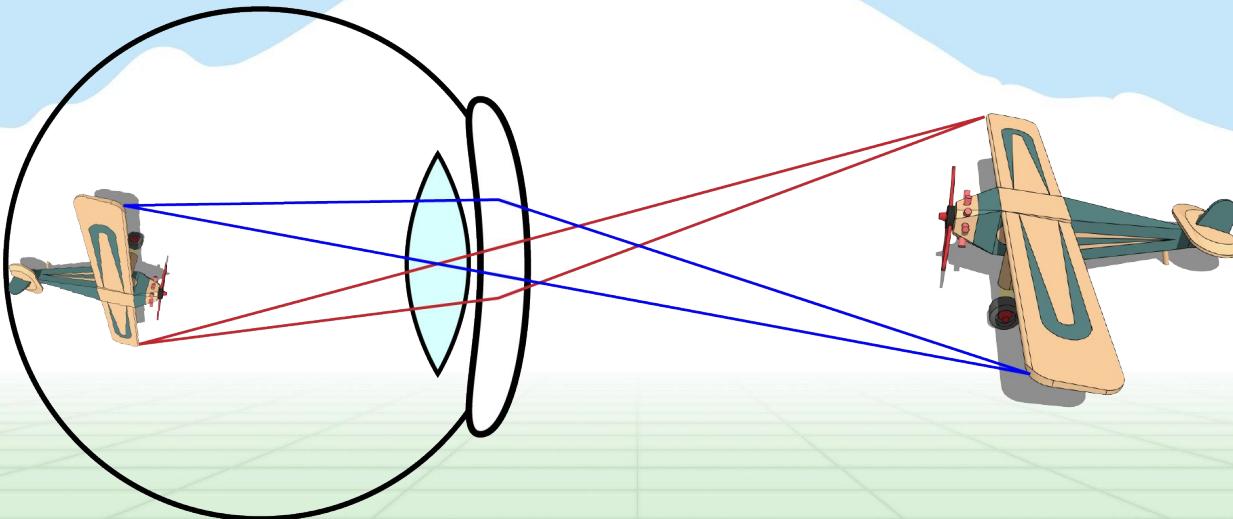
- Homework 0 due tonight!
- Homework 1 will be out tomorrow
- My office hours changed, they will be Thursdays, 3:30-5:00pm in CSE 203

Previously  
On

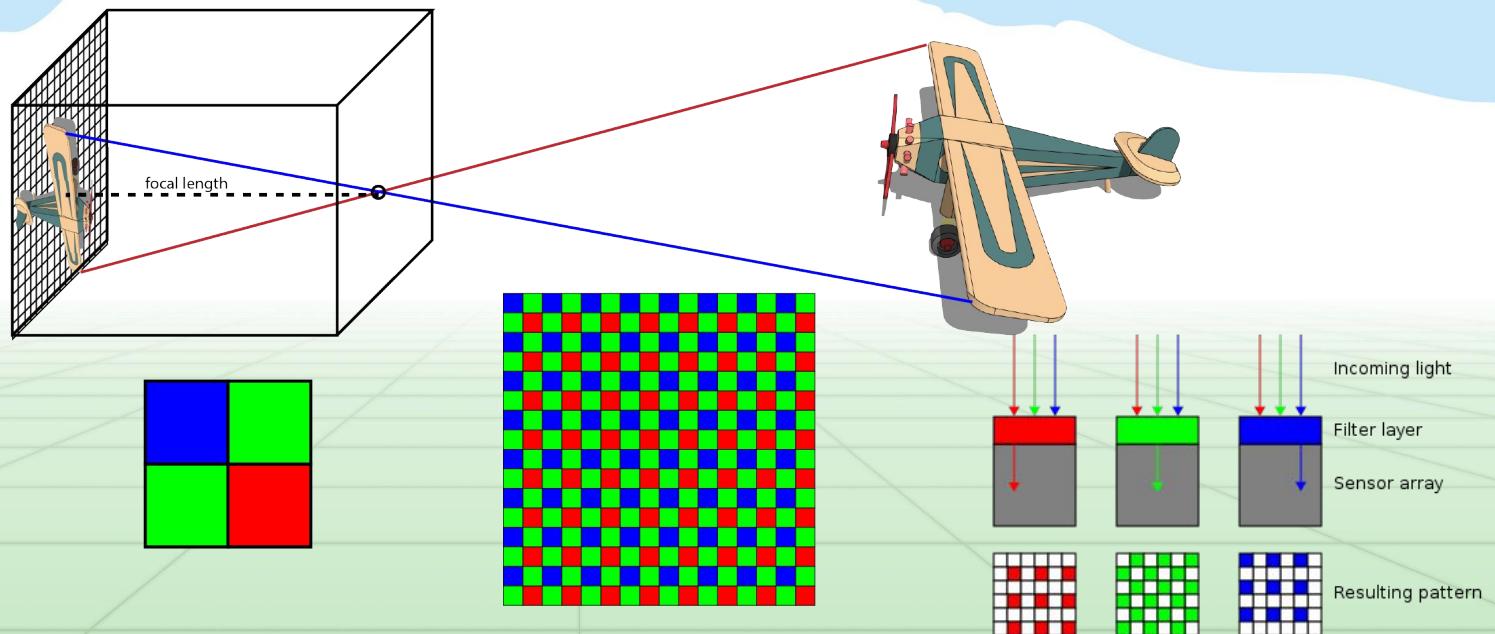


Ancient Secrets  
of Computer Vision

# Eyes: projection onto retina



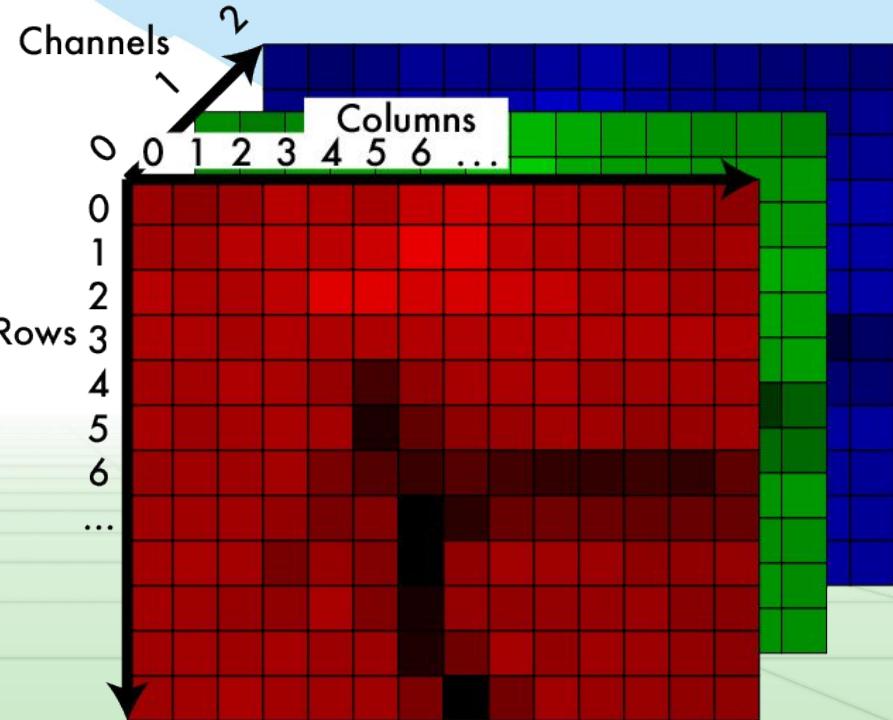
# Camera: projection onto sensor array



# Image: tensor of light values

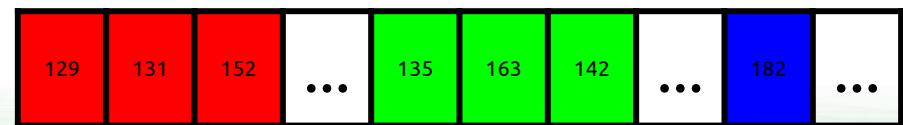
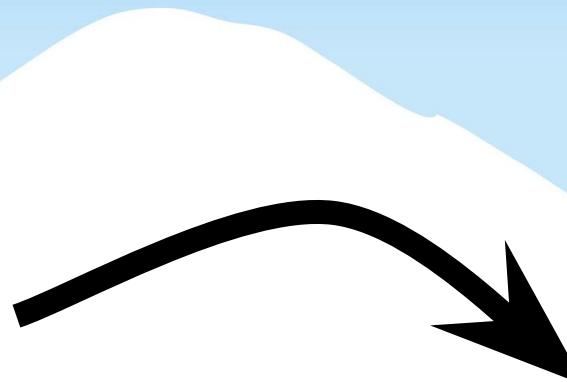
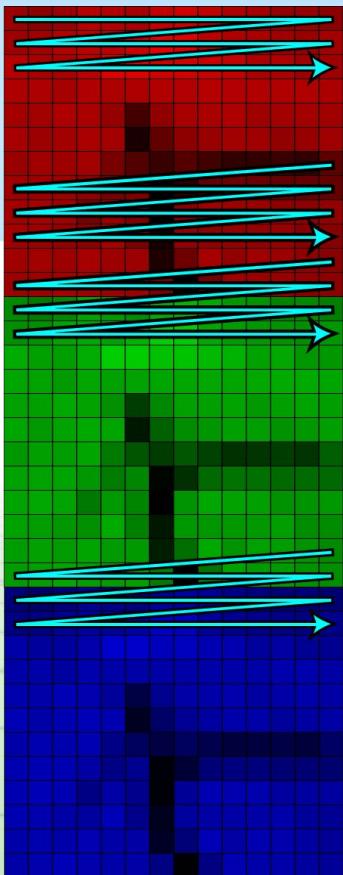
Addressing pixels:

- Size:  $W \times H \times C$
- Index:  $(x, y, z)$ 
  - Column  $x$ , row  $y$ , chan  $z$
- Like cartesian but flipped



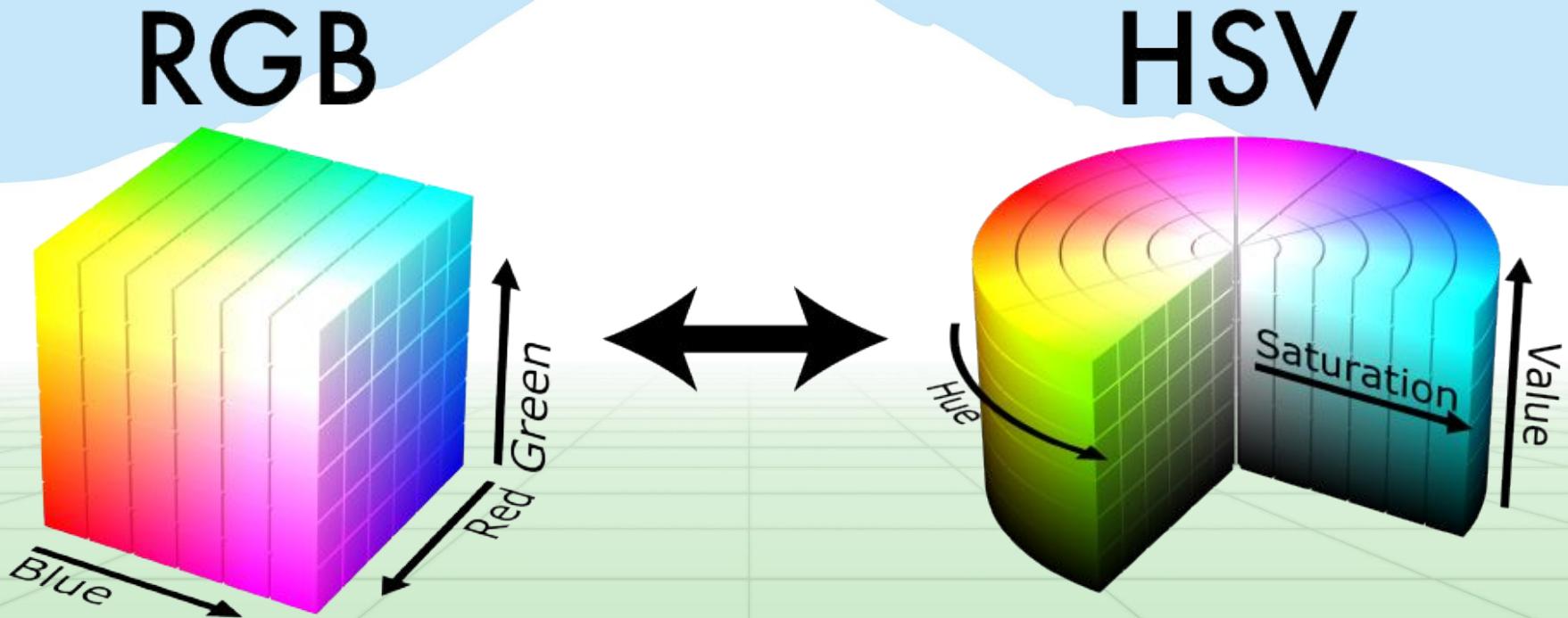
# CHW for storing in 1d array

---

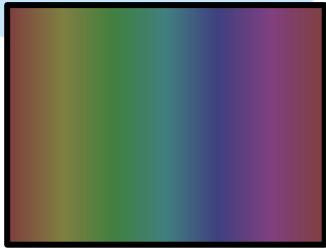
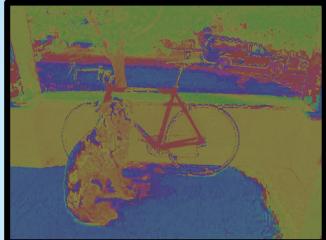


# Other colorspaces are fun!

---



# Or pattern...



# Nearest neighbor: what it sounds like

$f(x,y,z) = \text{Im}(\text{round}(x), \text{round}(y), z)$

- Looks blocky
- Common pitfall:  
Integer division  
rounds down in C
- Note: z is still int

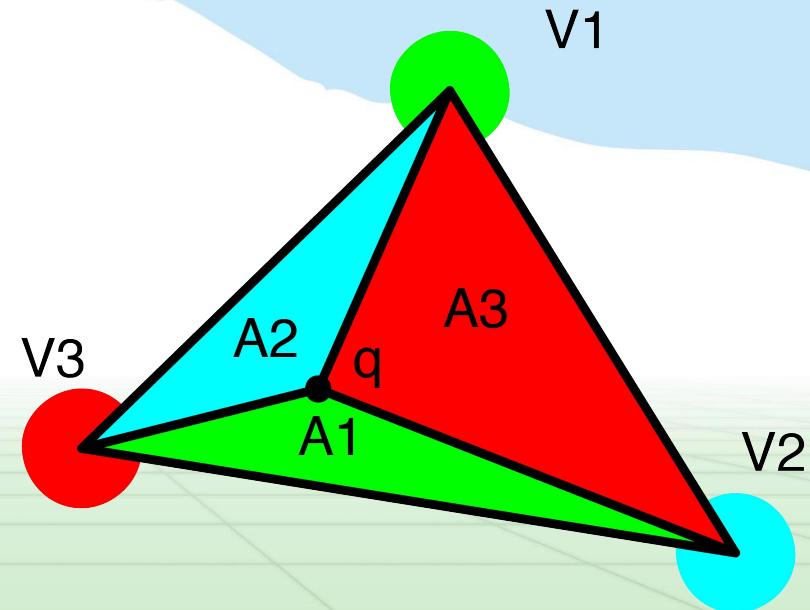


## Triangle interpolation: for less structured image

Weighted sum using of triangles:

$$Q = V1 * A1 + V2 * A2 + V3 * A3$$

Should normalize this based on total area



## Bilinear interpolation: for grids, pretty good

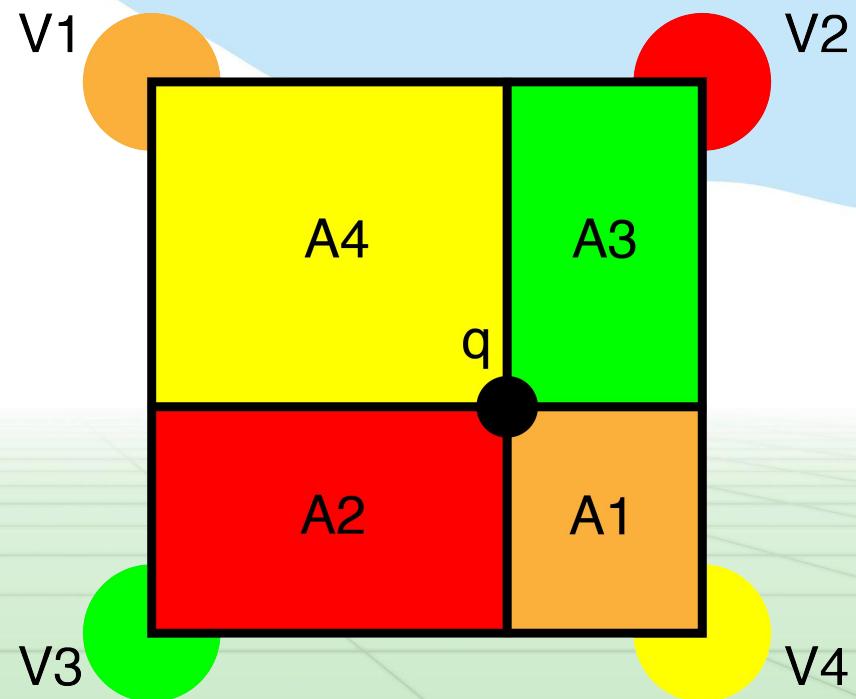
This time find the closest pixels in a box

Same plan, weighted sum based on area of opposite rectangle

$$Q = V1*A1 + V2*A2 + V3*A3 + V4*A4$$

Still need to normalize!

Or do we?



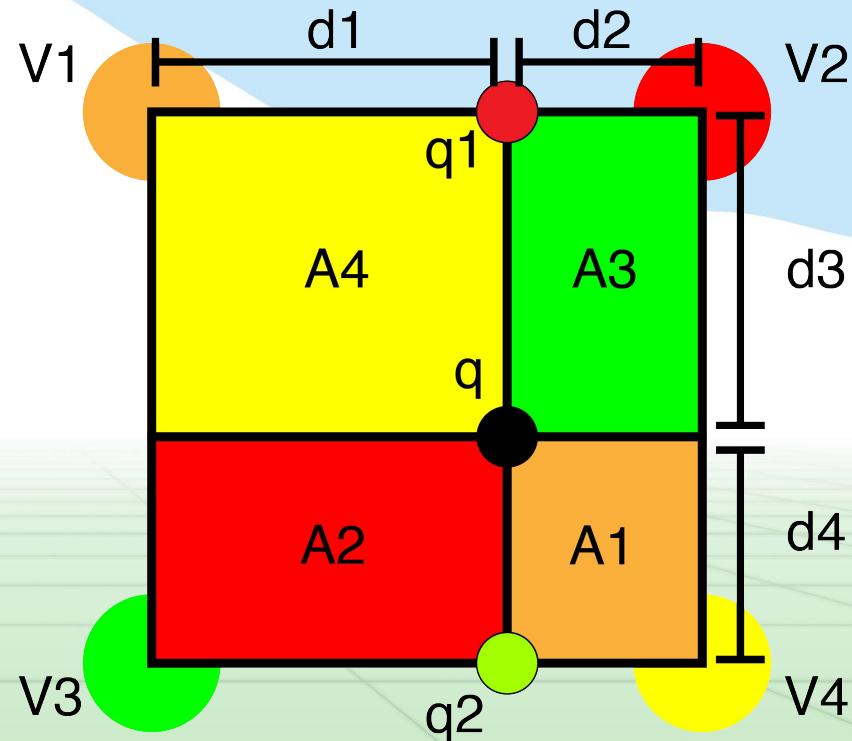
## Bilinear interpolation: for grids, pretty good

Alternatively, linear  
interpolation of linear  
interpolates

$$q_1 = V_1 \cdot d_2 + V_2 \cdot d_1$$

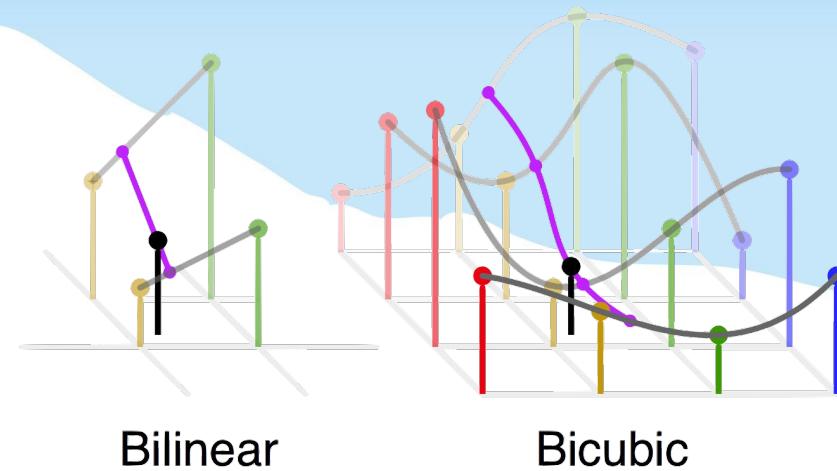
$$q_2 = V_3 \cdot d_2 + V_4 \cdot d_1$$

$$q = q_1 \cdot d_4 + q_2 \cdot d_3$$



## Bicubic sampling: more complex, maybe better?

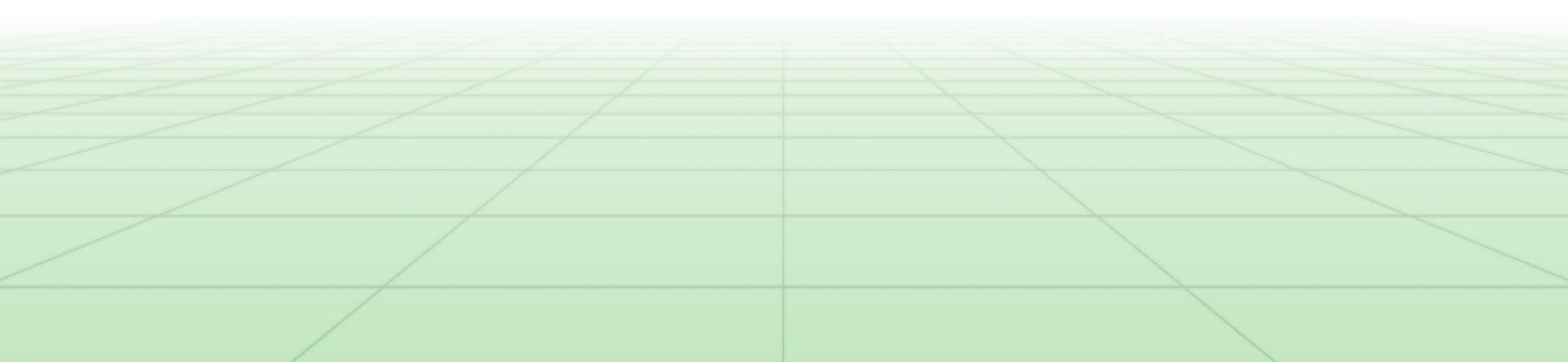
- A cubic interpolation of 4 cubic interpolations
- Smoother than bilinear, no “star”
- 16 nearest neighbors
- Fit 3rd order poly:
  - $f(x) = a + bx + cx^2 + dx^3$
- Interpolate along axis
- Fit another poly to interpolated values



# Chapter Four



# Convolutions!

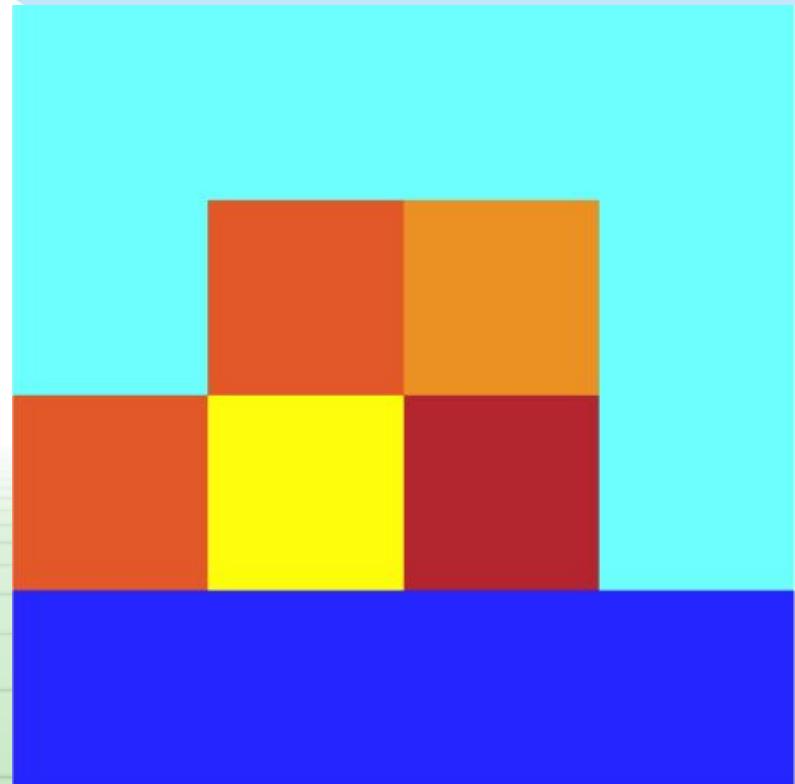


So what is this interpolation  
useful for?

# Image resizing!

Say we want to increase  
the size of an image...

This is a beautiful  
image of a sunset... it's  
just very small...

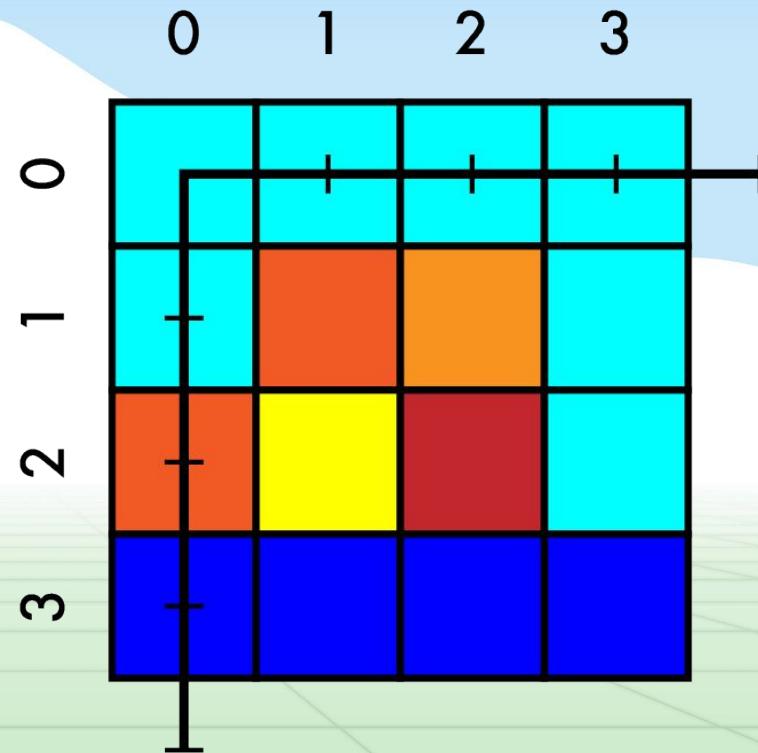


# Image resizing!

Say we want to increase  
the size of an image...

This is a beautiful  
image of a sunset... it's  
just very small...

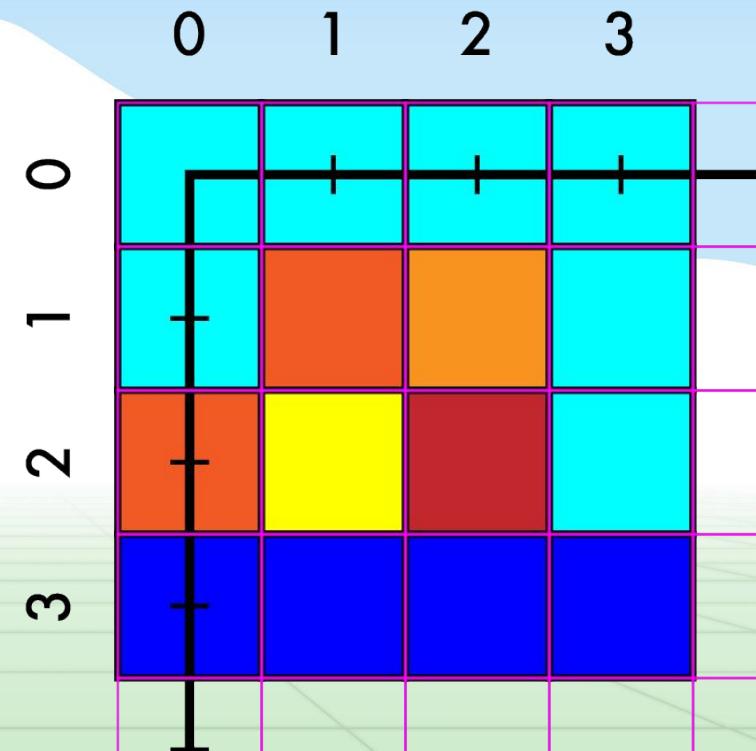
Say we want to increase  
size  $4 \times 4 \rightarrow 7 \times 7$



# Resize 4x4 -> 7x7

---

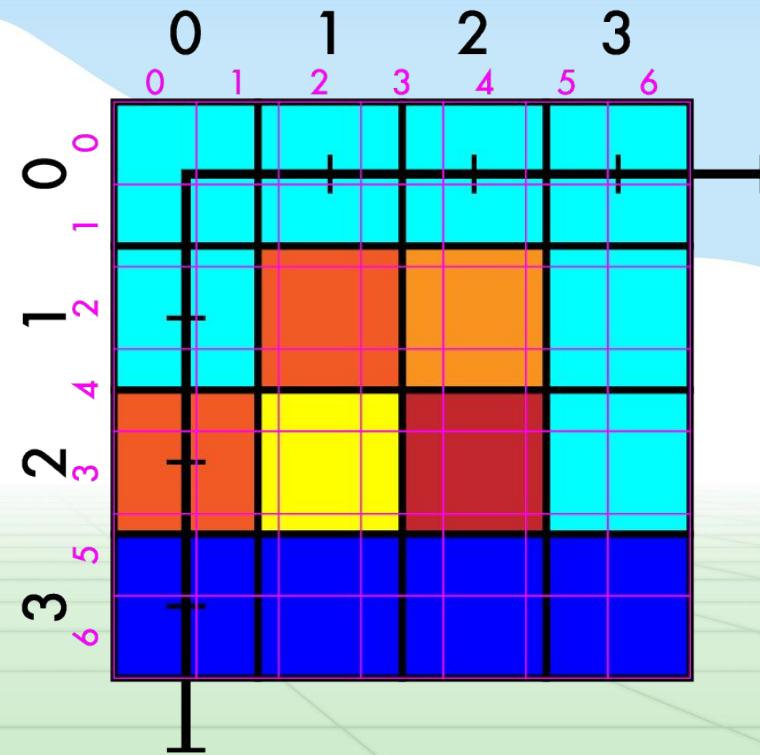
- Create our new image



# Resize 4x4 -> 7x7

---

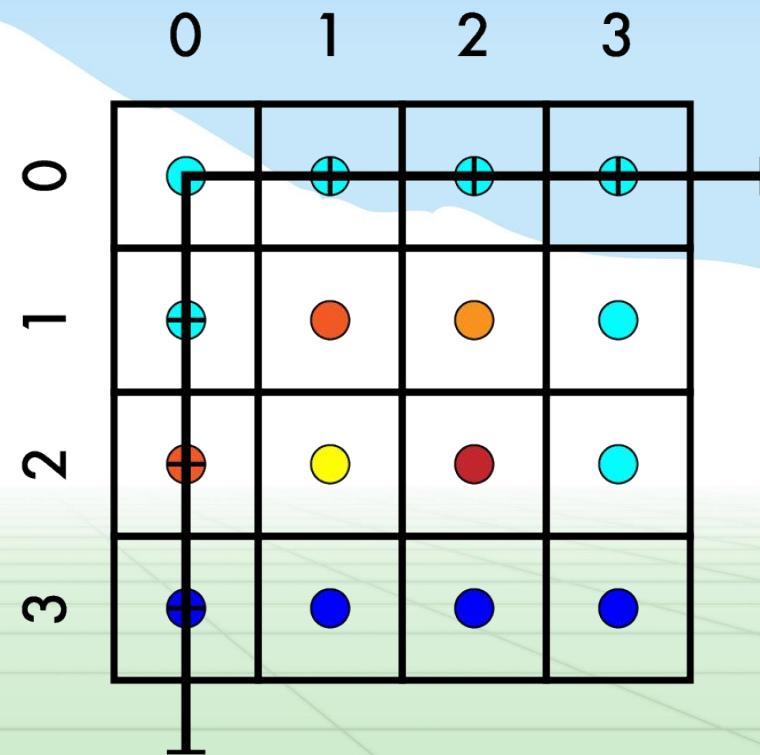
- Create our new image
- Match up coordinates



# Resize 4x4 -> 7x7

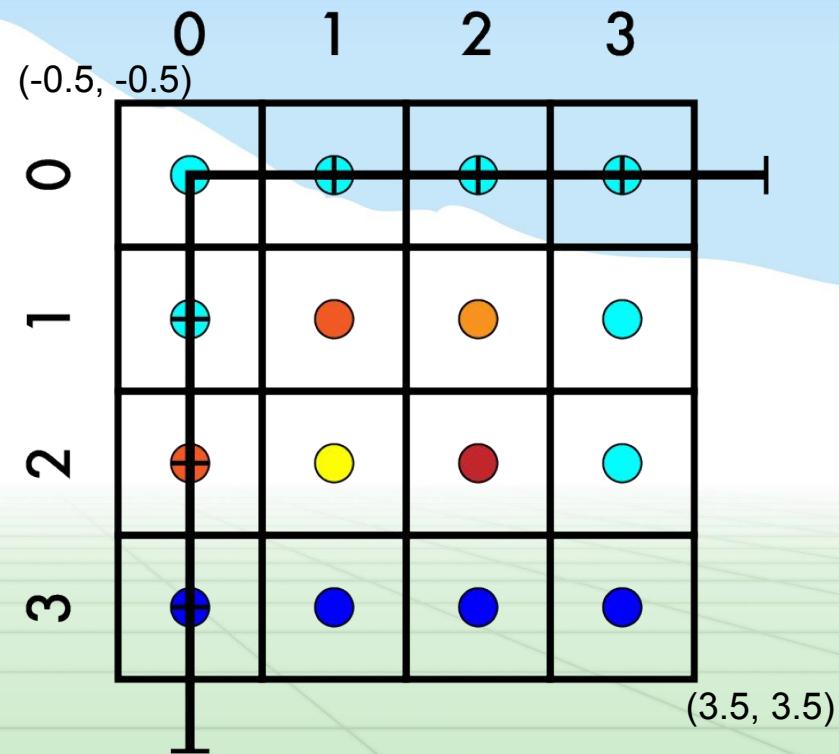
---

- Create our new image
- Match up coordinates



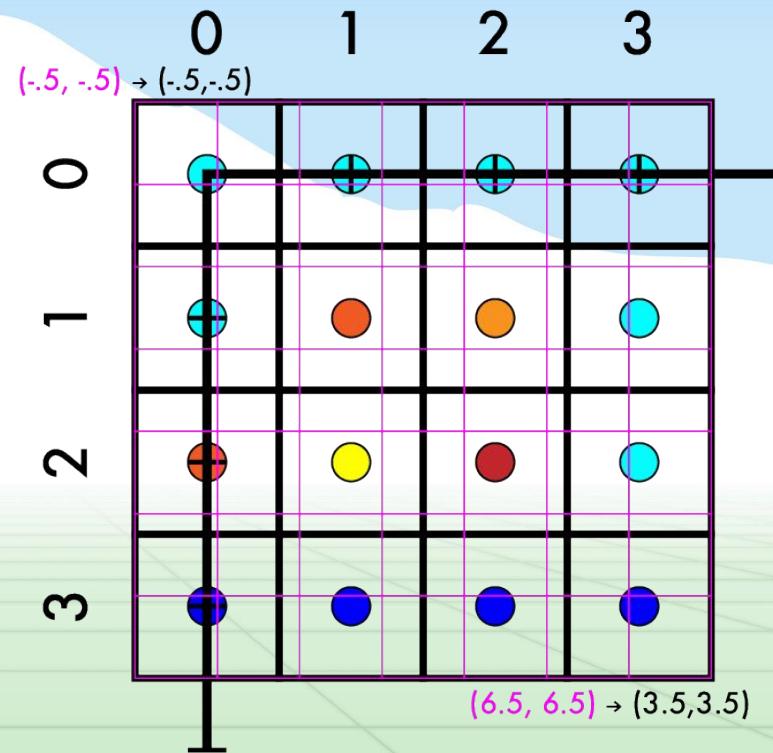
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates



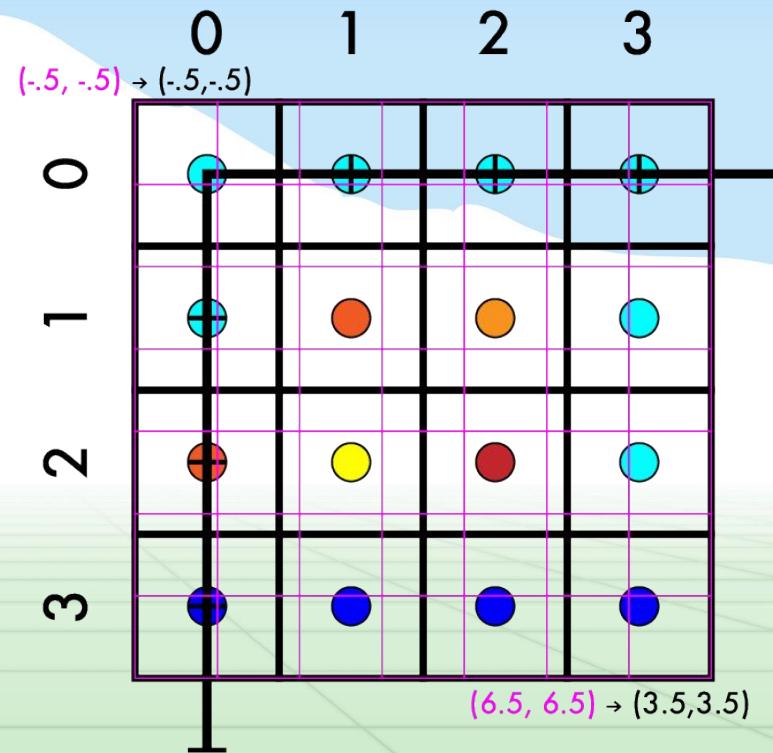
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - System of equations
  - $aX + b = Y$
  - $a * -.5 + b = -.5$
  - $a * 6.5 + b = 3.5$



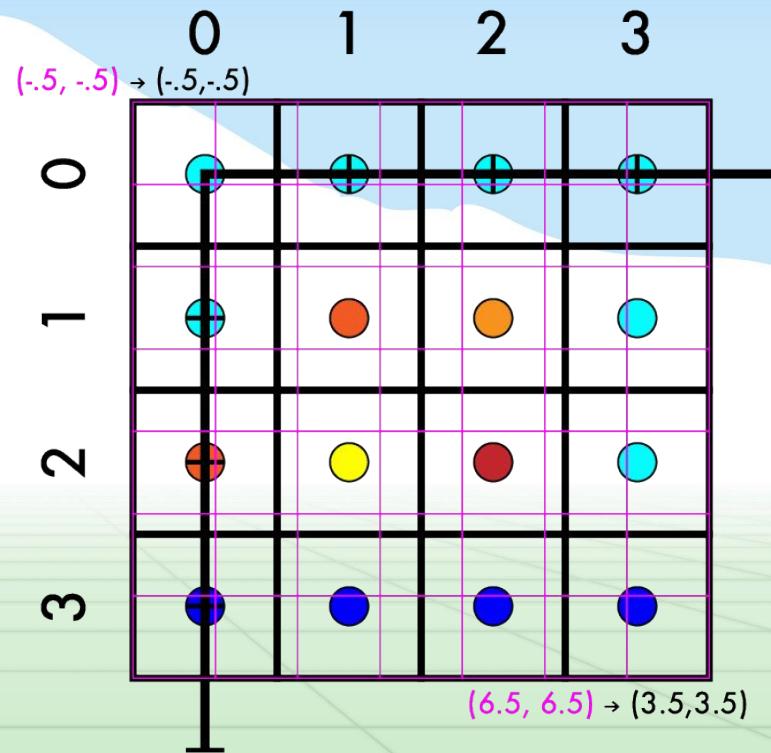
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - System of equations
  - $aX + b = Y$
  - $a * -.5 + b = -.5$
  - $a * 6.5 + b = 3.5$ 
    - $a * 7 = 4$



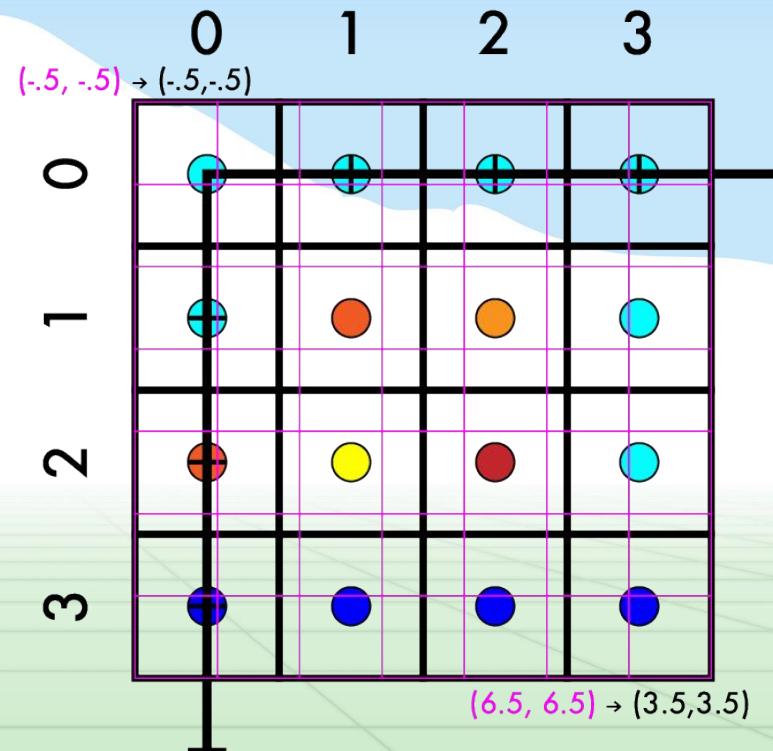
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - System of equations
  - $aX + b = Y$
  - $a * -.5 + b = -.5$
  - $a * 6.5 + b = 3.5$ 
    - $a * 7 = 4$
    - $a = 4/7$



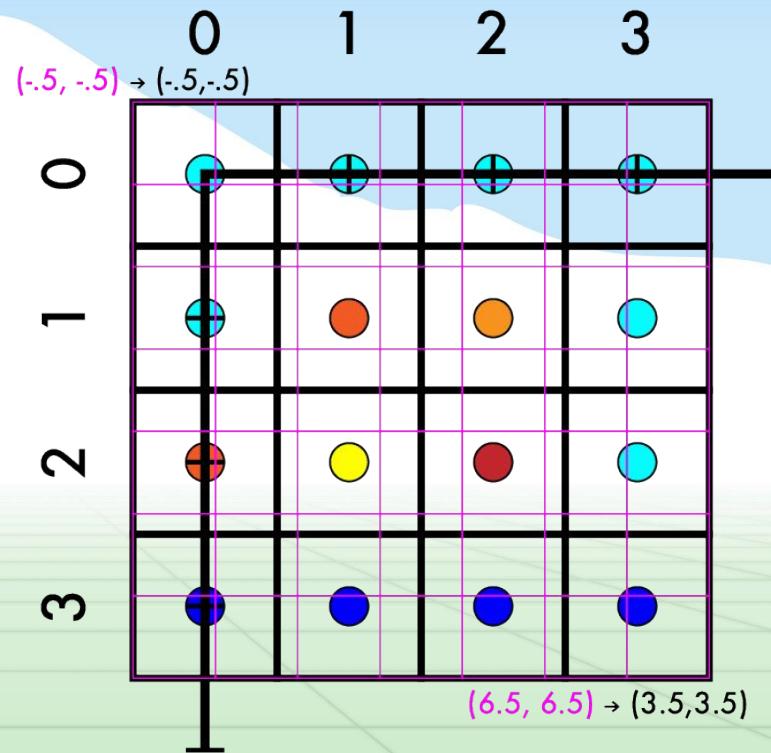
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - System of equations
  - $aX + b = Y$
  - $a * -.5 + b = -.5$
  - $a * 6.5 + b = 3.5$
  - $a = 4/7$



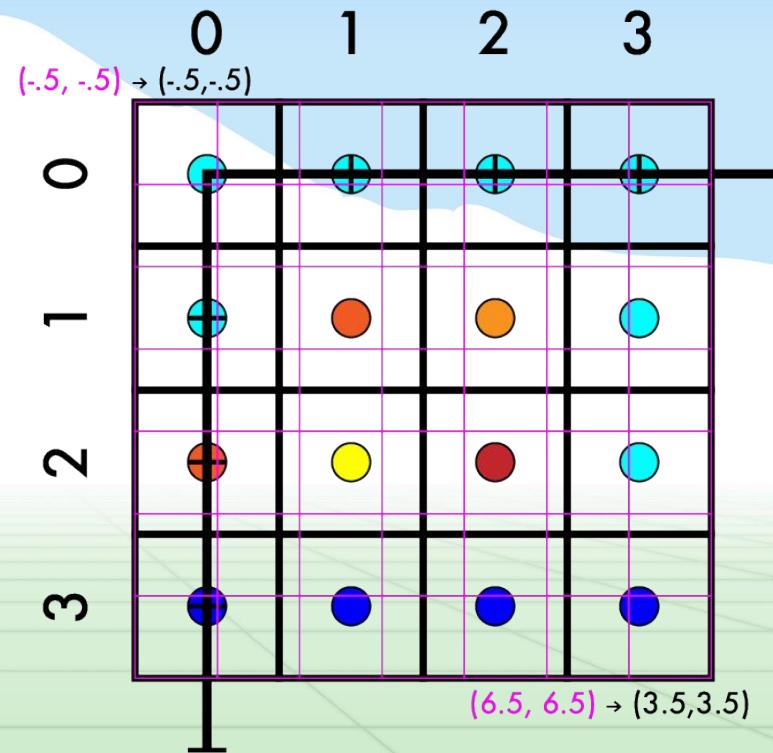
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - System of equations
  - $aX + b = Y$
  - $a * -.5 + b = -.5$
  - $a * 6.5 + b = 3.5$
  - $a = 4/7$ 
    - $a * -.5 + b = -.5$



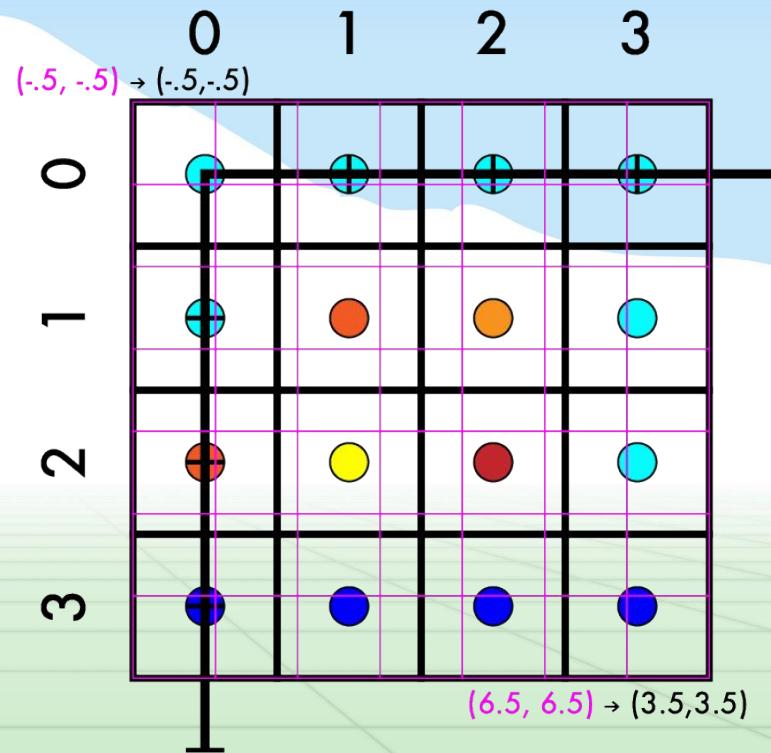
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - System of equations
  - $aX + b = Y$
  - $a * -.5 + b = -.5$
  - $a * 6.5 + b = 3.5$
  - $a = 4/7$ 
    - $a * -.5 + b = -.5$
    - $4/7 * -1/2 + b = -1/2$



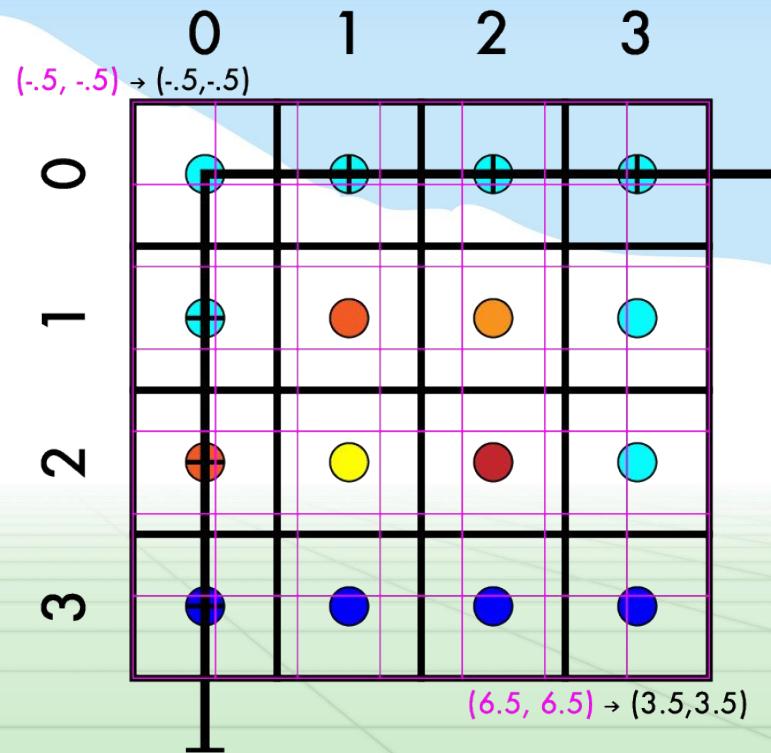
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - System of equations
  - $aX + b = Y$
  - $a * -.5 + b = -.5$
  - $a * 6.5 + b = 3.5$
  - $a = 4/7$ 
    - $a * -.5 + b = -.5$
    - $4/7 * -1/2 + b = -1/2$
    - $-4/14 + b = -7/14$



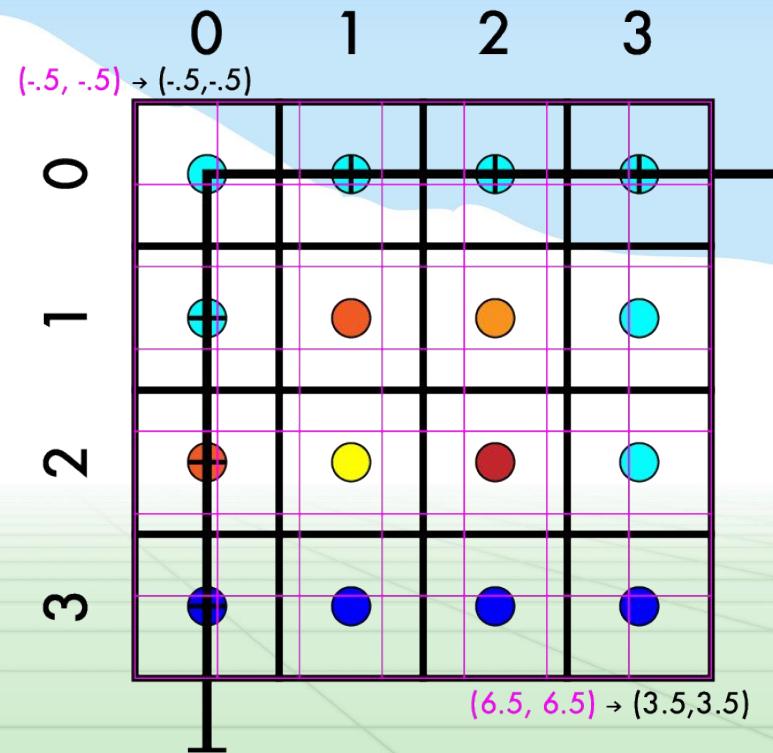
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - System of equations
  - $aX + b = Y$
  - $a * -.5 + b = -.5$
  - $a * 6.5 + b = 3.5$
  - $a = 4/7$ 
    - $a * -.5 + b = -.5$
    - $4/7 * -1/2 + b = -1/2$
    - $-4/14 + b = -7/14$
    - $b = -3/14$



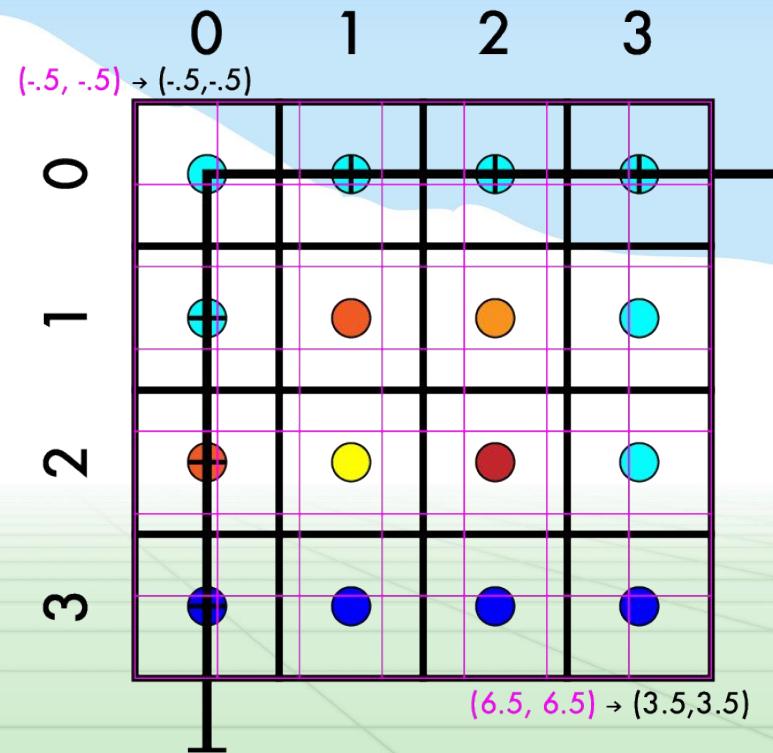
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - System of equations
  - $aX + b = Y$
  - $a * -.5 + b = -.5$
  - $a * 6.5 + b = 3.5$
  - $a = 4/7$
  - $b = -3/14$



# Resize 4x4 -> 7x7

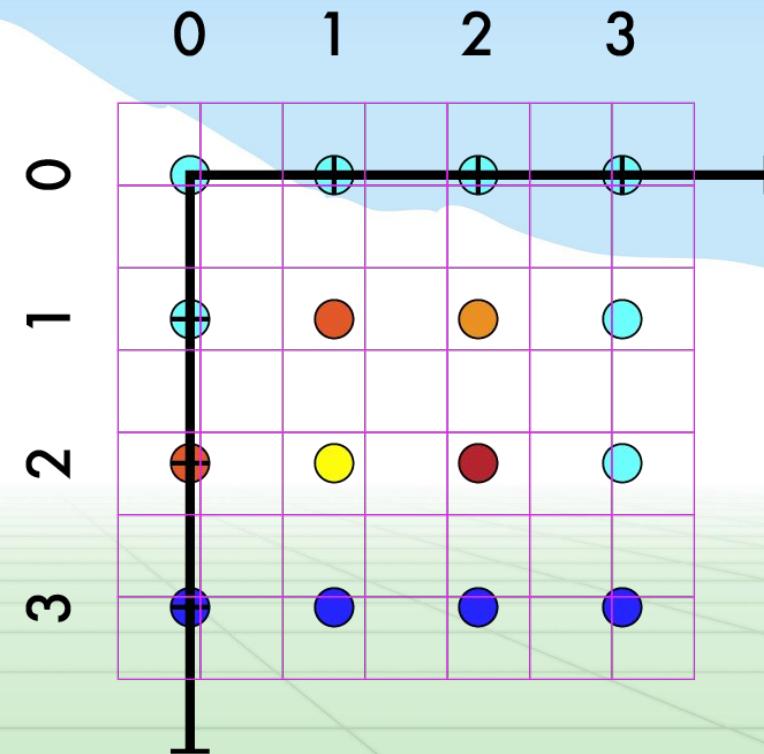
- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$



# Resize 4x4 -> 7x7

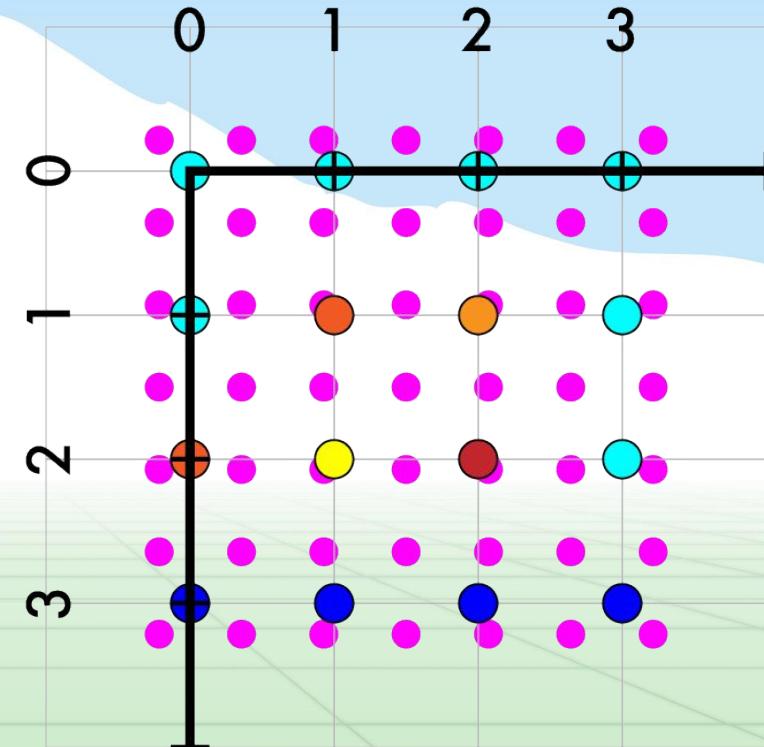
---

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$



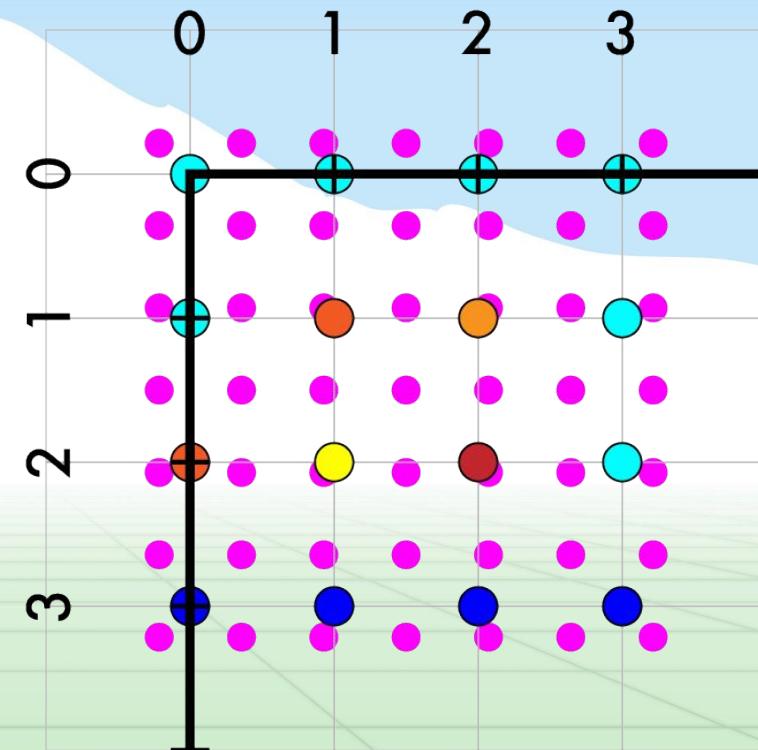
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$
- Iterate over new pts



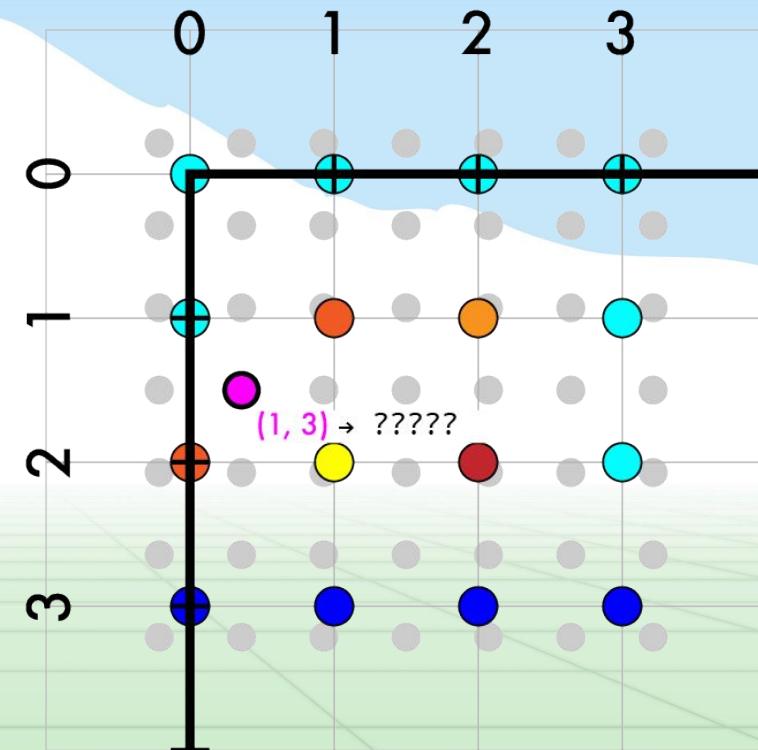
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords



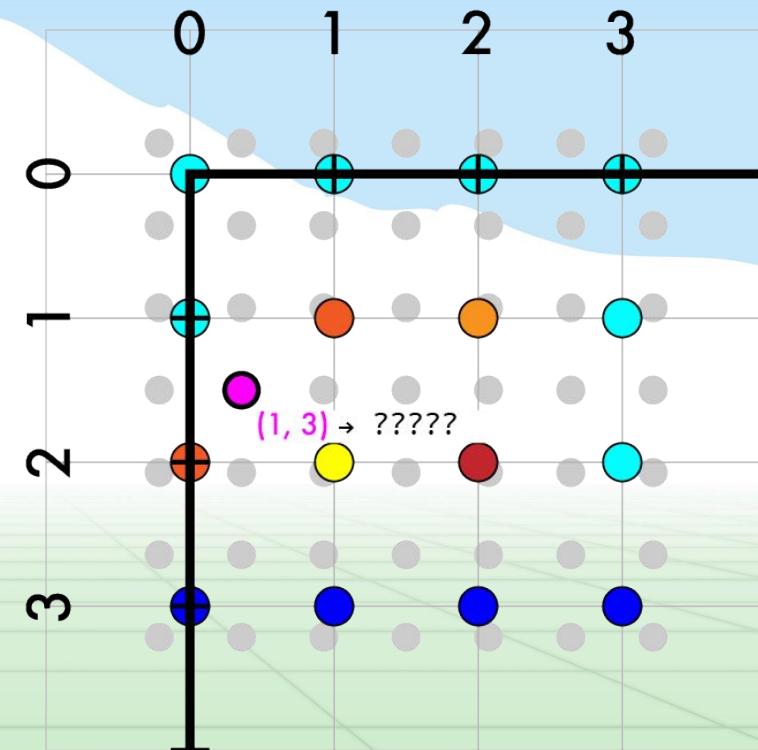
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3)$



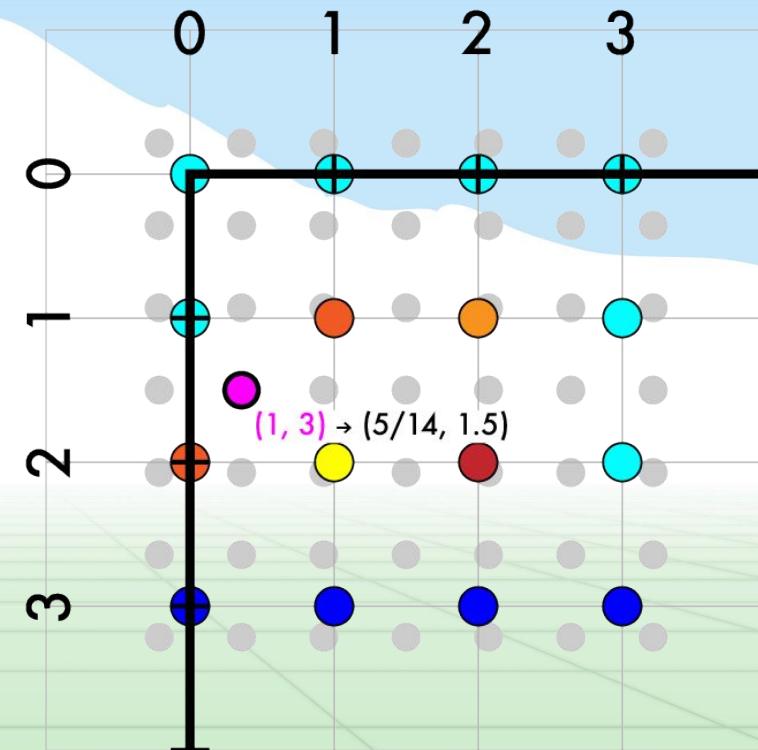
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3)$
  - $4/7 * 1 - 3/14$
  - $4/7 * 3 - 3/14$



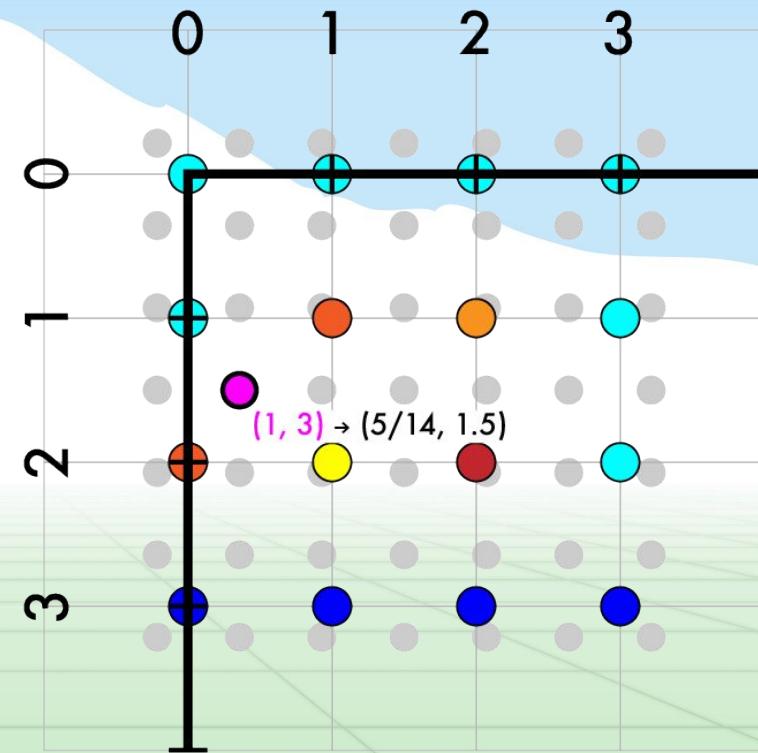
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
    - (1, 3)
    - $4/7 * 1 - 3/14$
    - $4/7 * 3 - 3/14$
    - $(5/14, 7/14)$



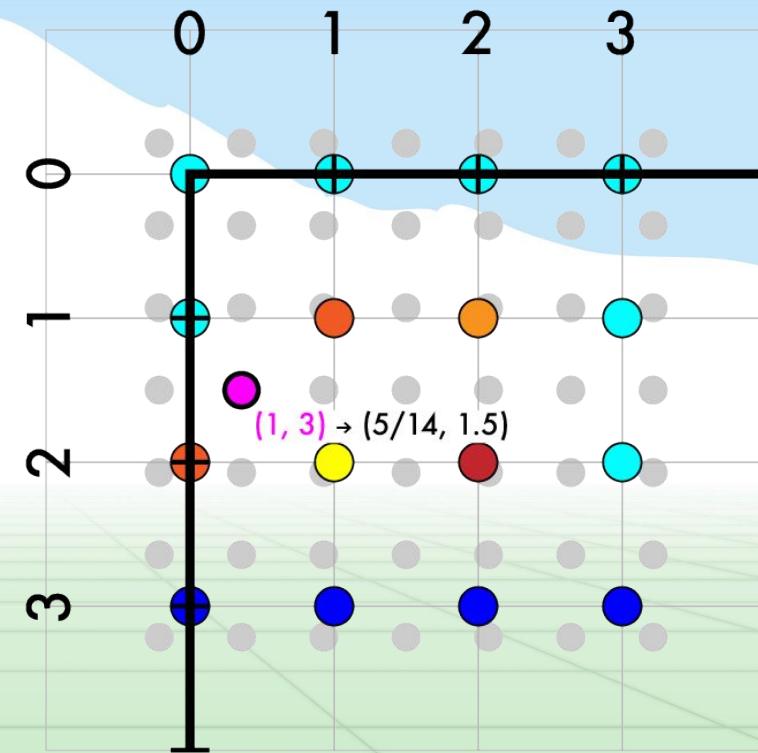
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$



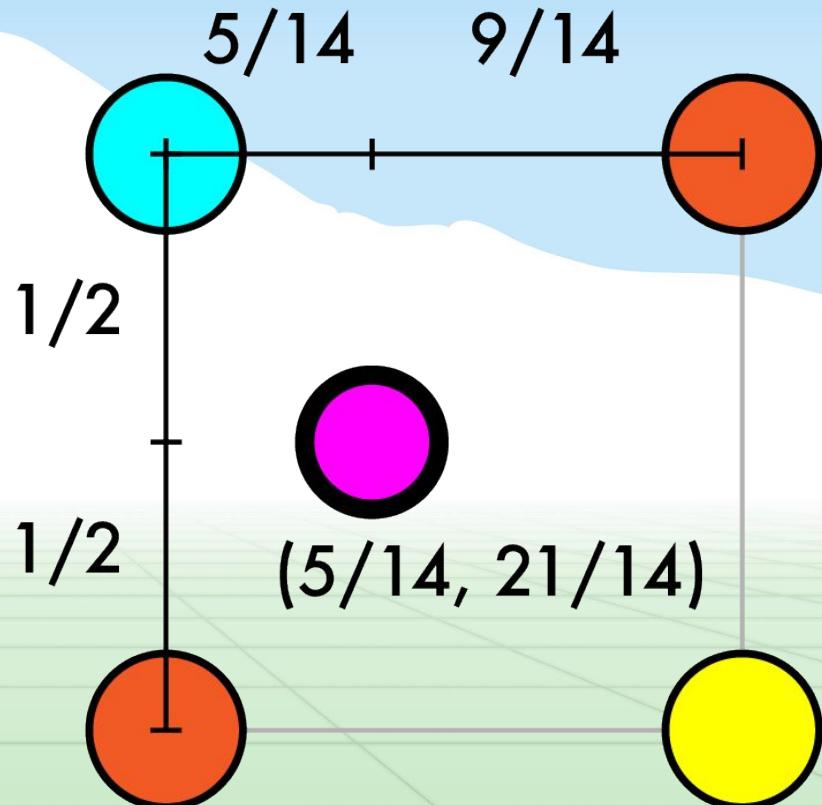
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values



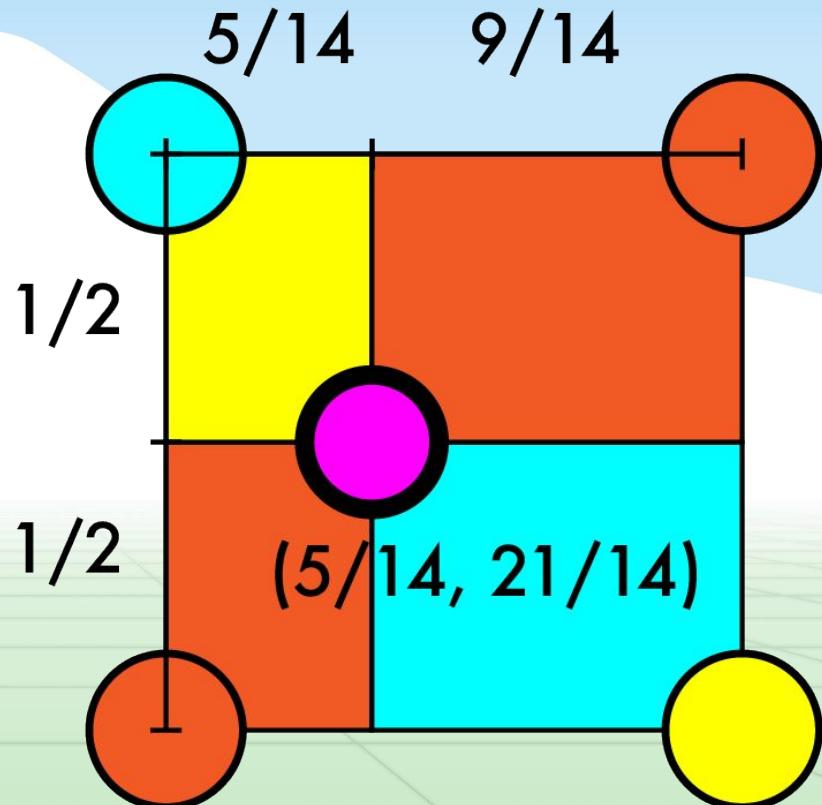
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
    - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values



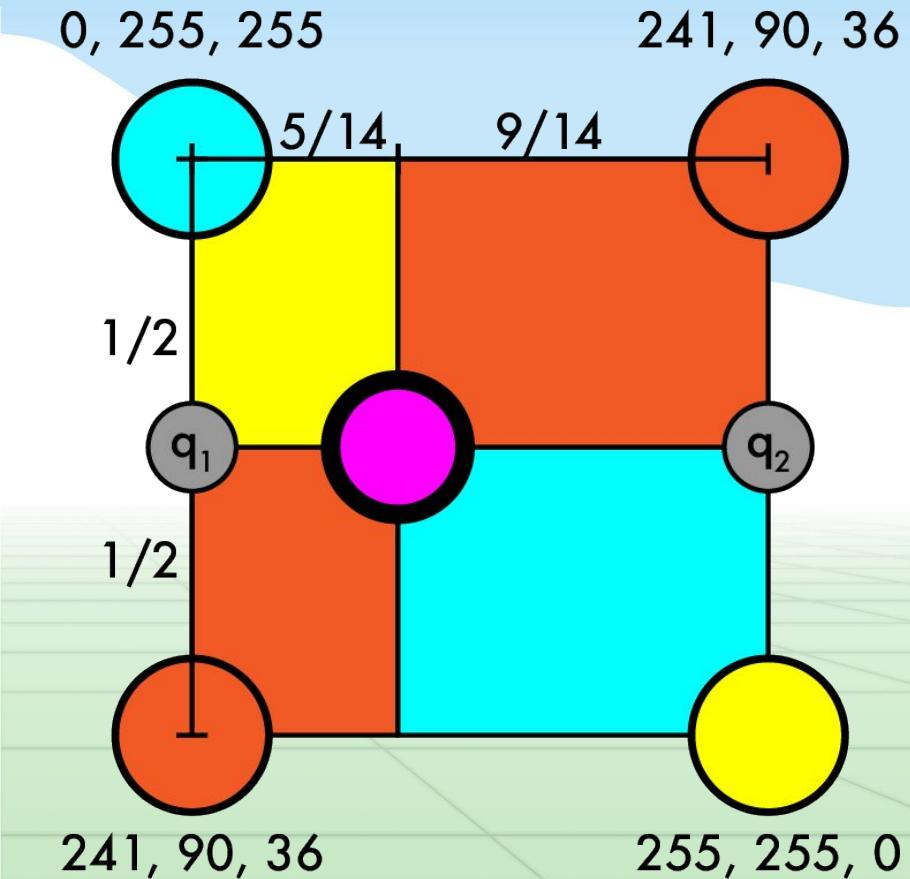
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - Size of opposite rects



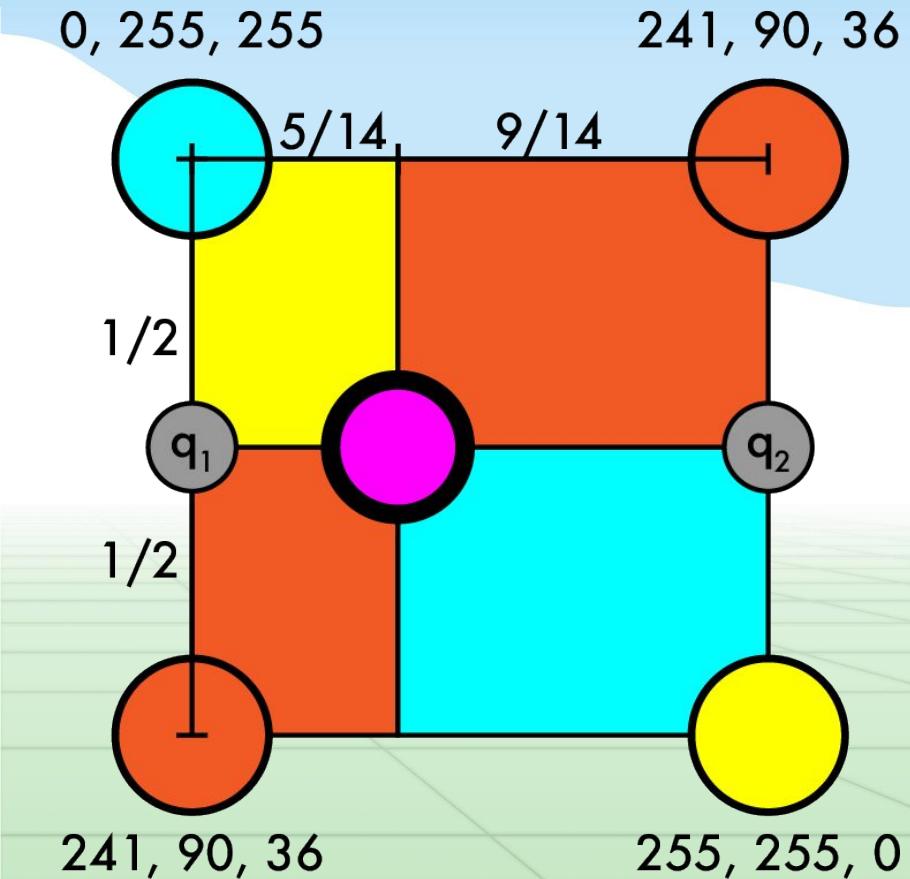
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - Size of opposite rects
    - OR find  $q_1$  and  $q_2$ , then interpolate between them



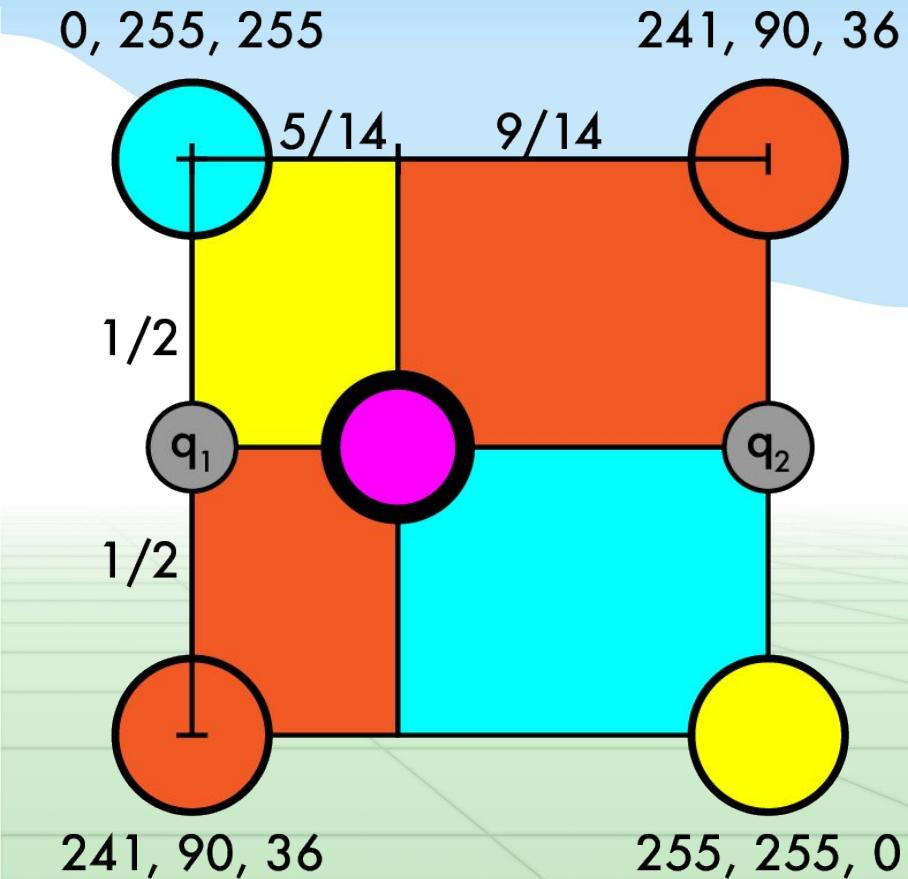
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q_1 = r_1, g_1, b_1$
    - $r_1 = .5*0 + .5*241$
    - $g_1 = .5*255 + .5*90$
    - $b_1 = .5*241 + .5*36$



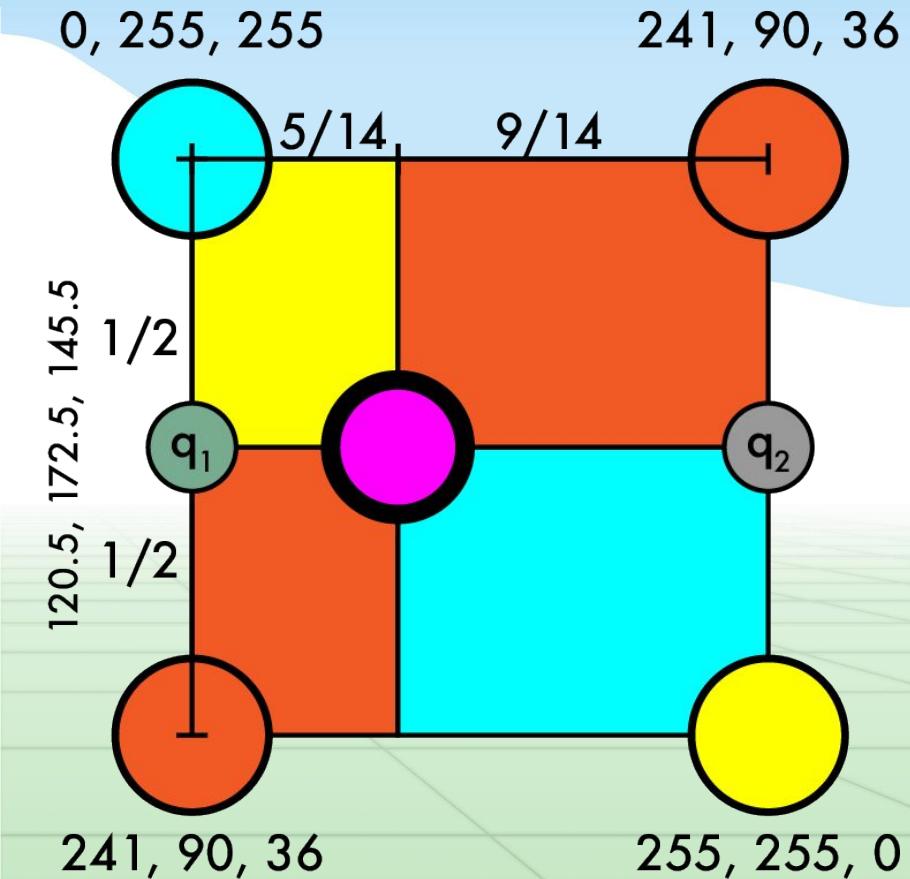
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q_1 = r_1, g_1, b_1$
    - $r_1 = .5*0 + .5*241$
    - $g_1 = .5*255 + .5*90$
    - $b_1 = .5*241 + .5*36$



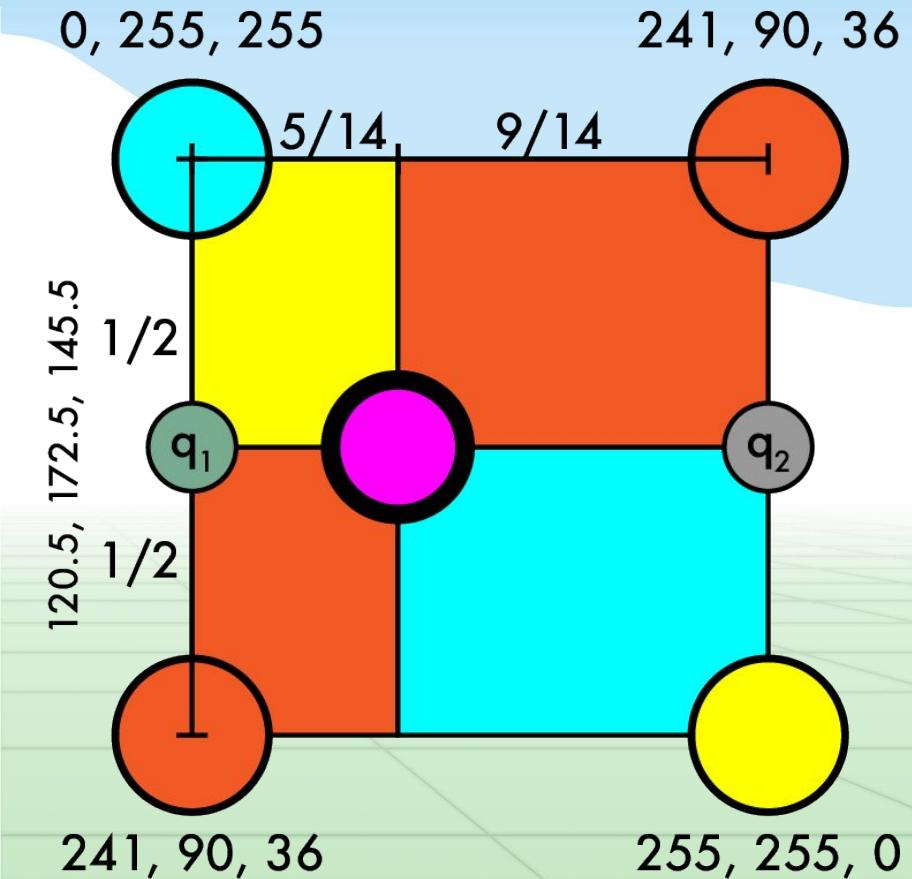
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q_1 = (120.5, 172.5, 145.5)$
    - $q_2 = r_2, g_2, b_2$



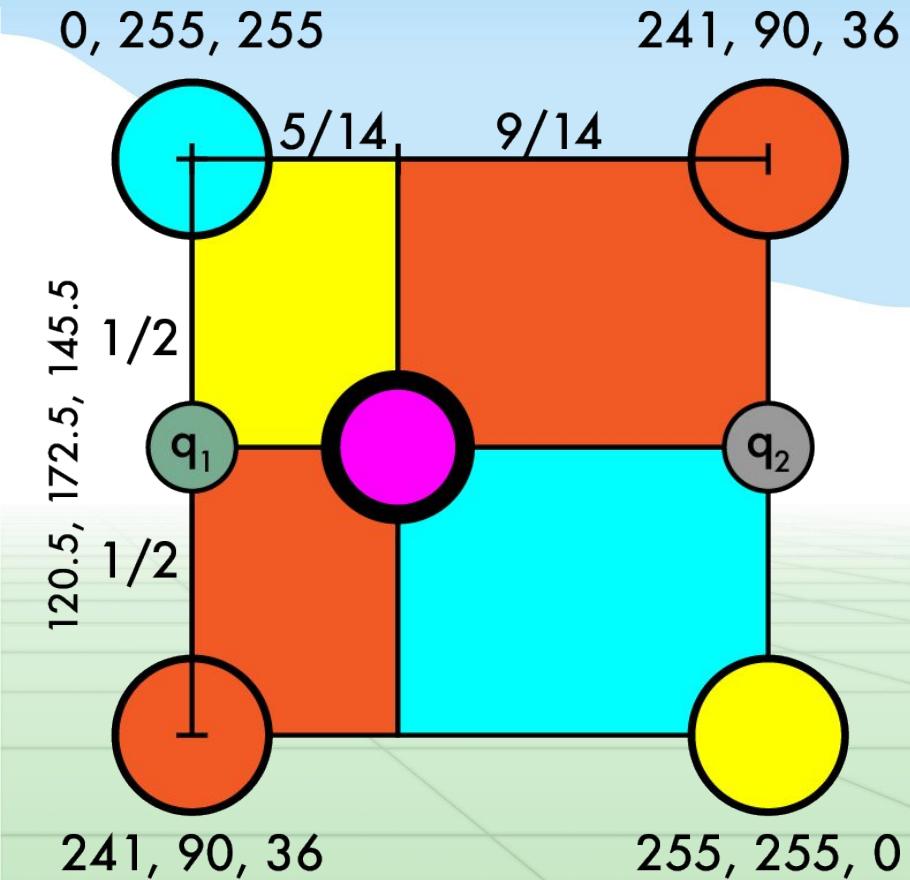
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q_1 = (120.5, 172.5, 145.5)$
    - $q_2 = r_2, g_2, b_2$



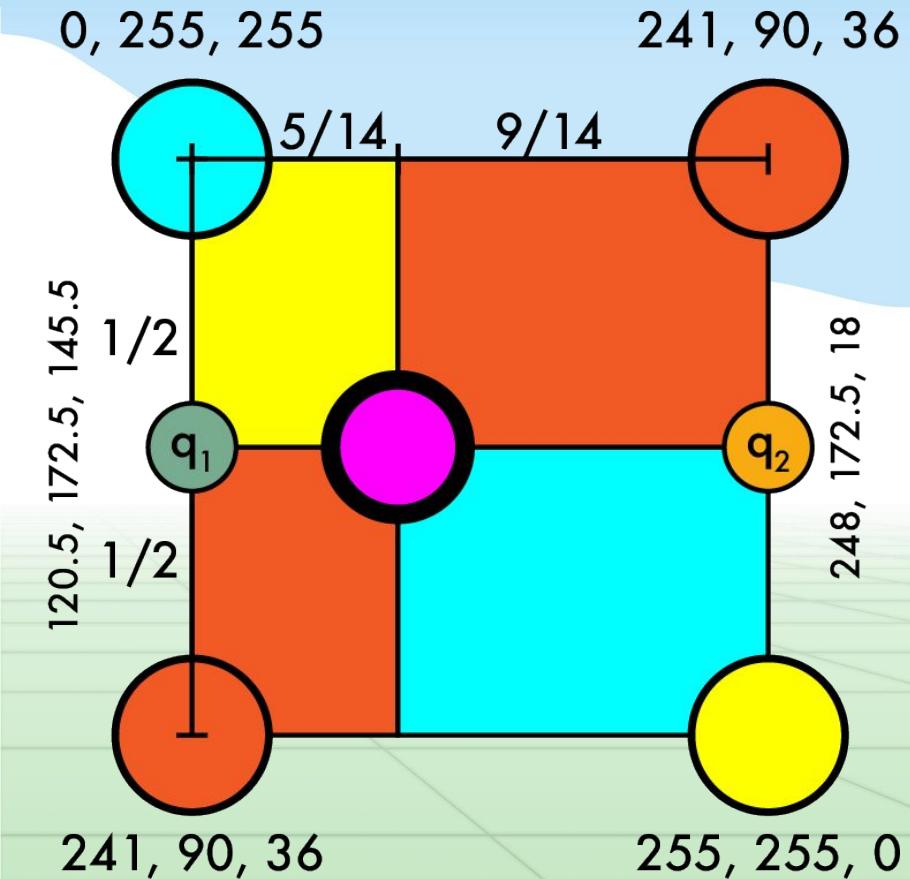
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q_1 = (120.5, 172.5, 145.5)$
    - $q_2 = r_2, g_2, b_2$
    - $r_2 = .5*241 + .5*255$
    - $g_2 = .5*90 + .5*255$
    - $b_2 = .5*36 + .5*0$



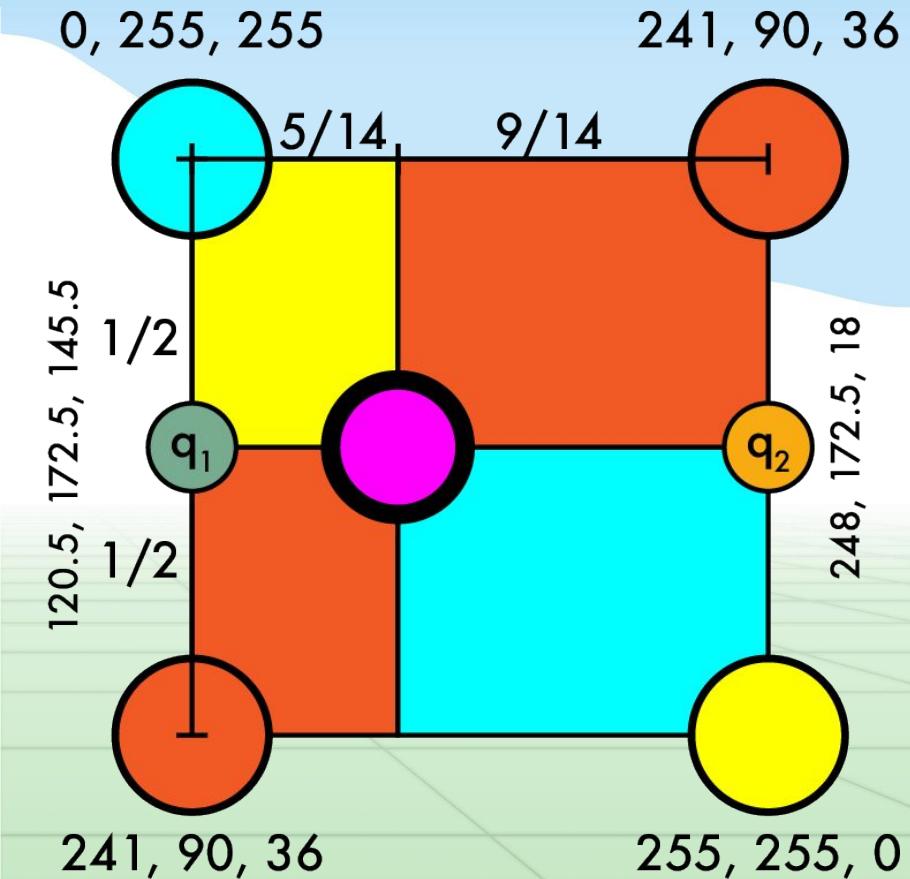
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q_1 = (120.5, 172.5, 145.5)$
    - $q_2 = (248, 172.5, 18)$



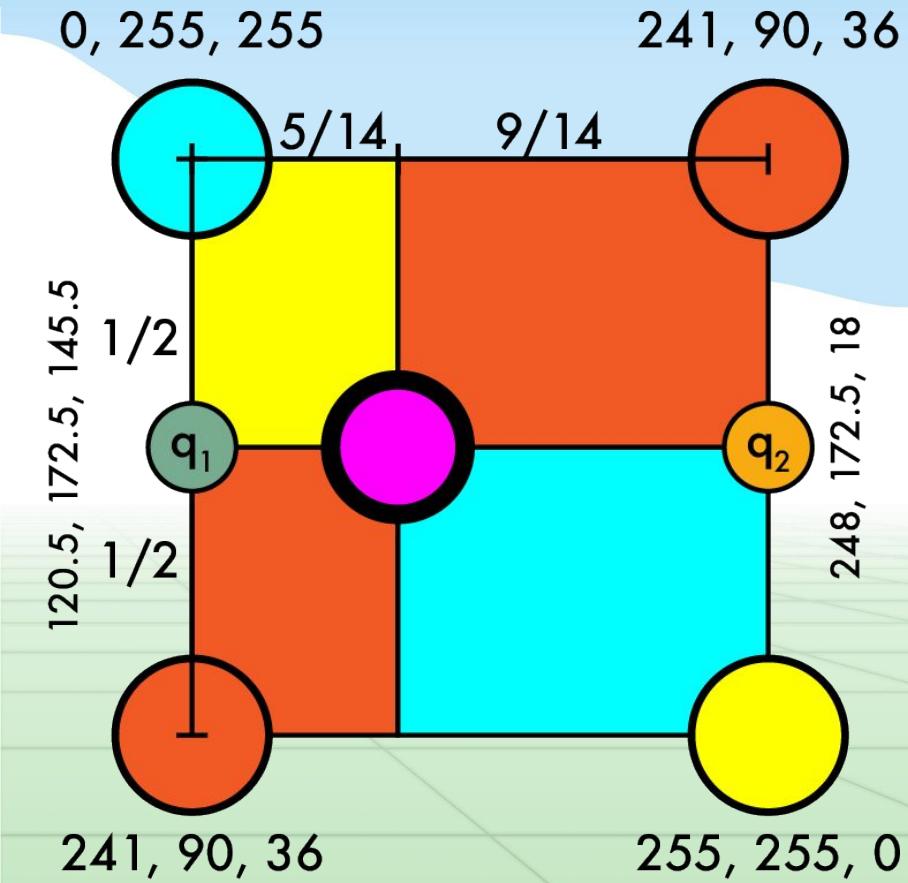
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 \times - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q_1 = (120.5, 172.5, 145.5)$
    - $q_2 = (248, 172.5, 18)$
    - $q = r, g, b$
    - $w = 9/14 * w_1 + 5/14 * w_2$



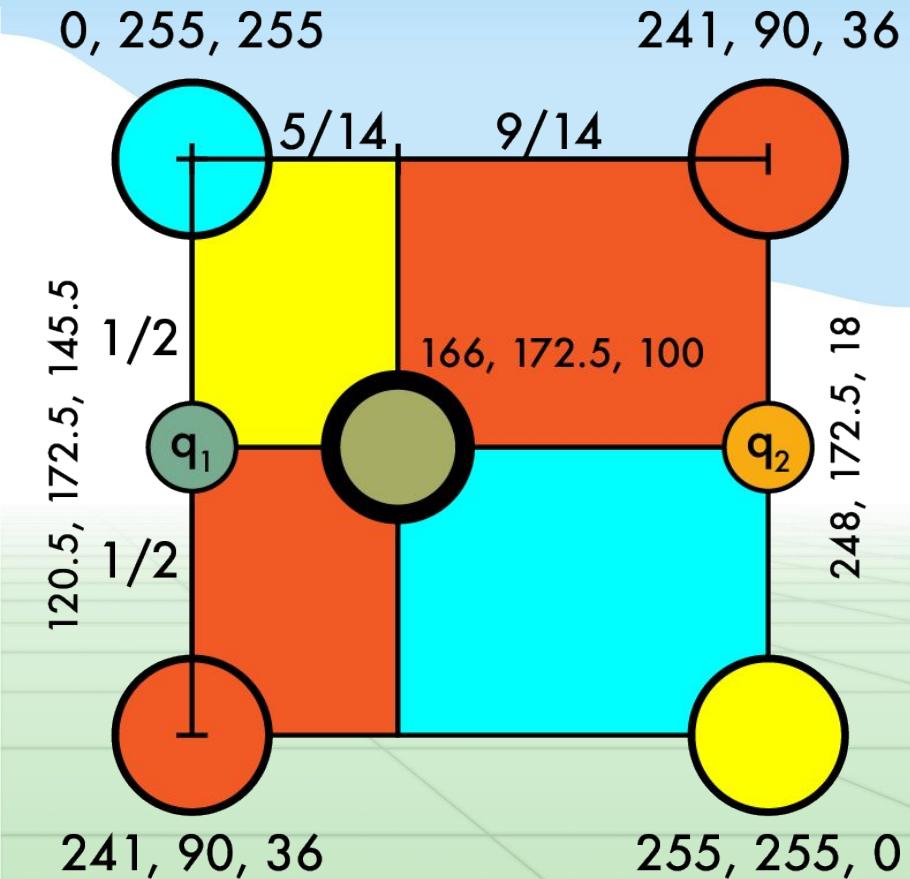
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q_1 = (120.5, 172.5, 145.5)$
    - $q_2 = (248, 172.5, 18)$
    - $q = r, g, b$
    - $w = 9/14 * w_1 + 5/14 * w_2$
    - $r = 9/14 * 120.5 + 5/14 * 248$
    - $g = 9/14 * 172.5 + 5/14 * 172.5$
    - $b = 9/14 * 145.5 + 5/14 * 18$



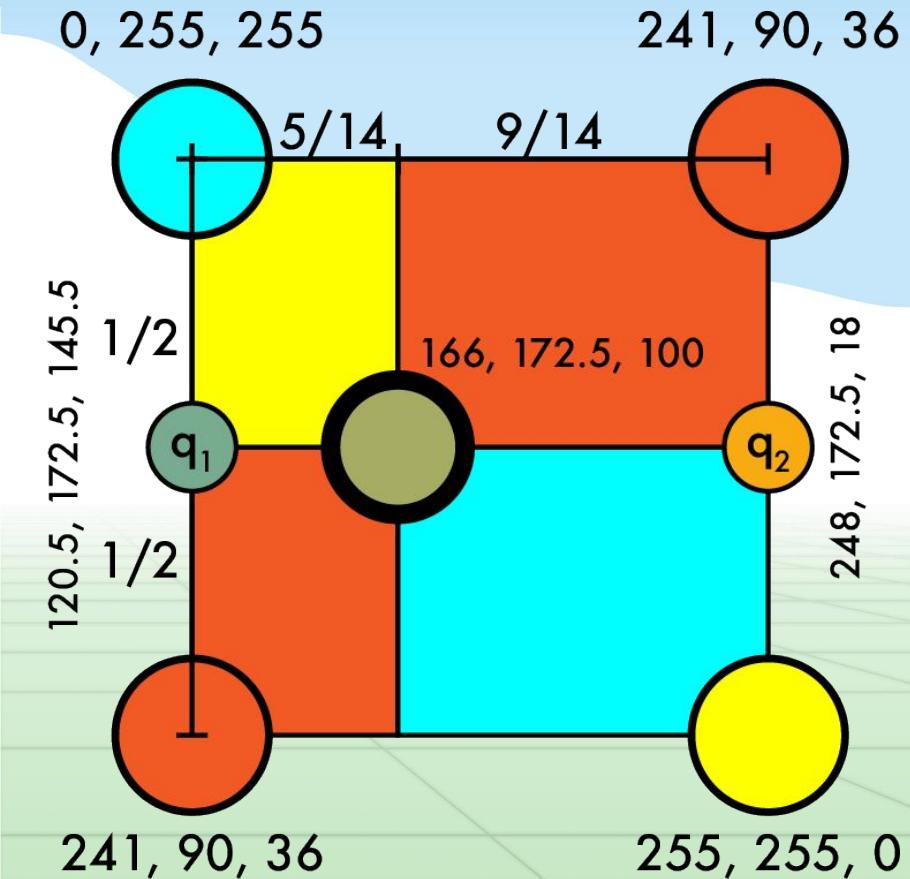
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q_1 = (120.5, 172.5, 145.5)$
    - $q_2 = (248, 172.5, 18)$
    - $q = (166, 172.5, 100)$



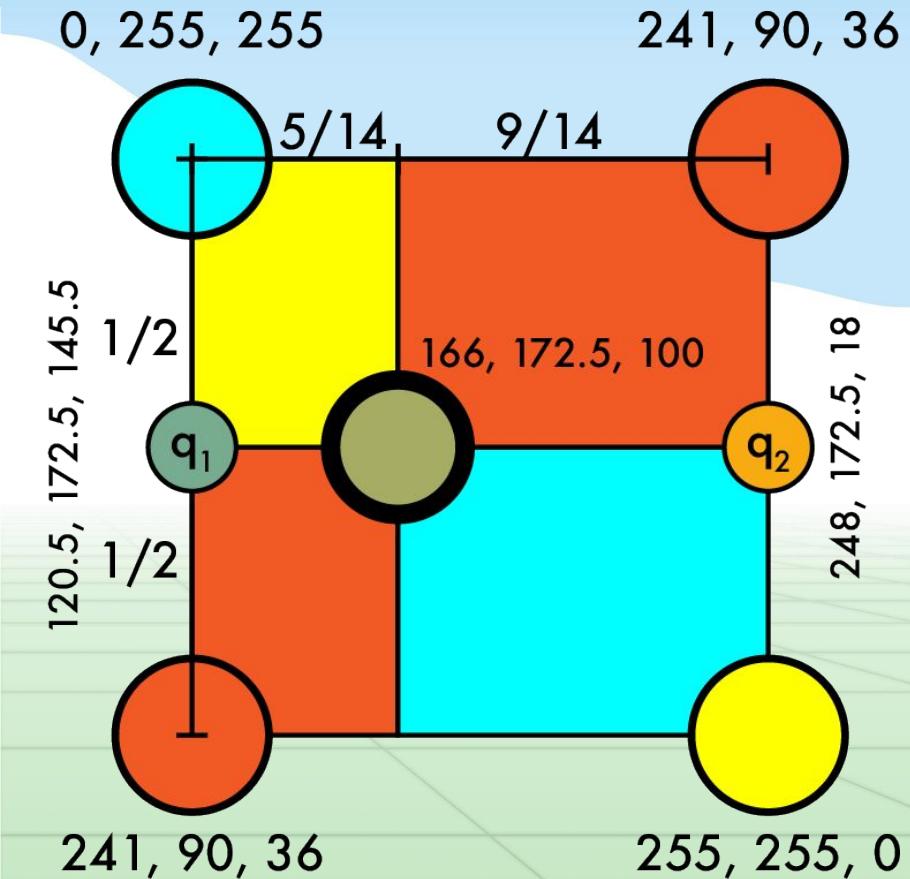
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q = (166, 172.5, 100)$



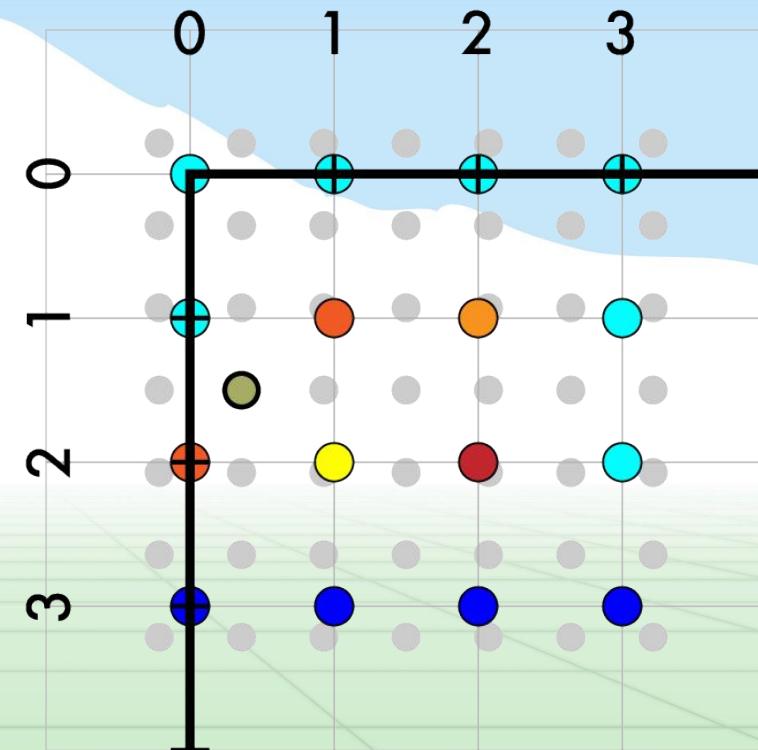
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q = (166, 172.5, 100)$



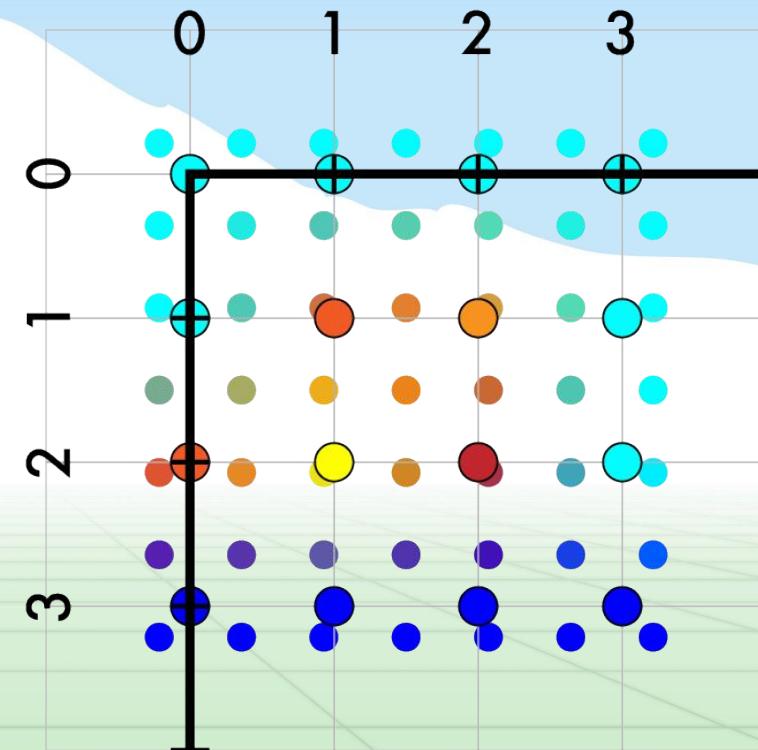
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q = (166, 172.5, 100)$



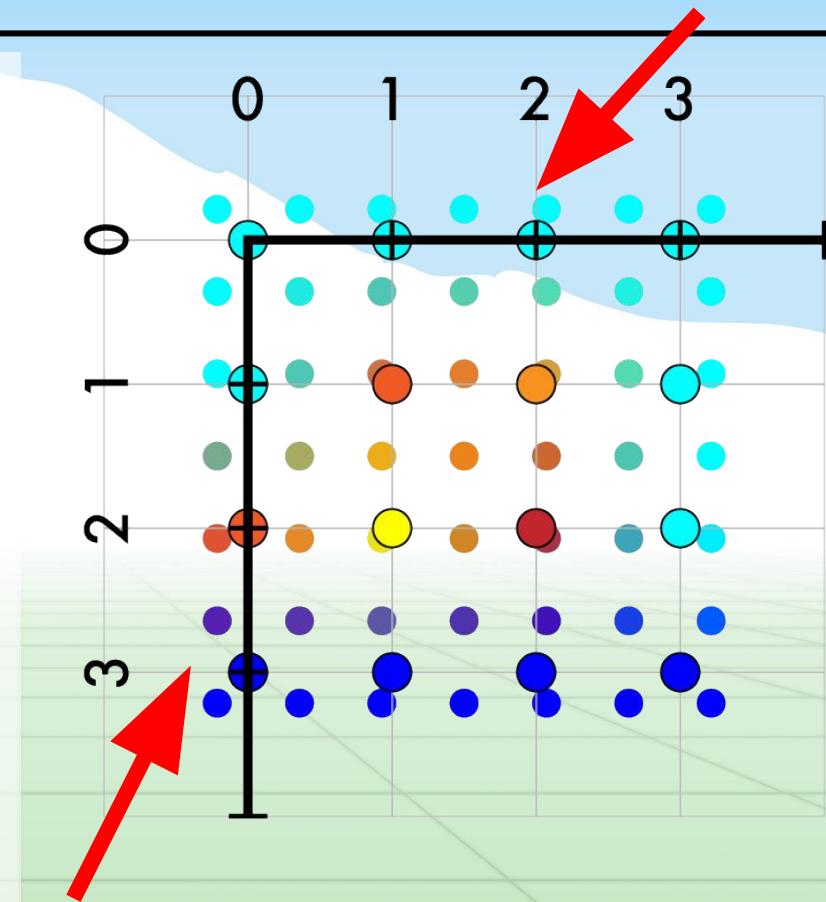
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q = (166, 172.5, 100)$
- Fill in the rest



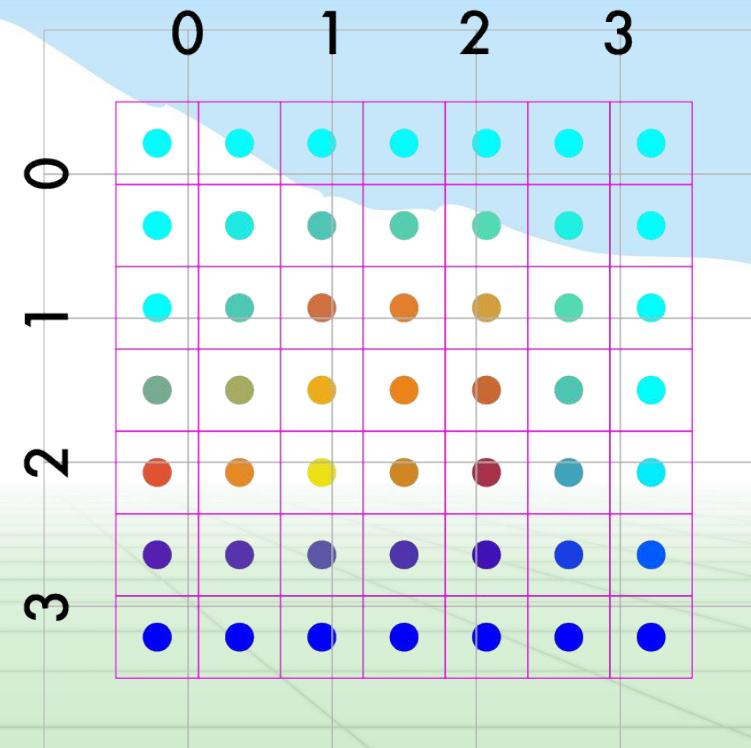
# Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
  - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q = (166, 172.5, 100)$
- Fill in the rest
  - On outer edges use padding!



# Resize 4x4 -> 7x7

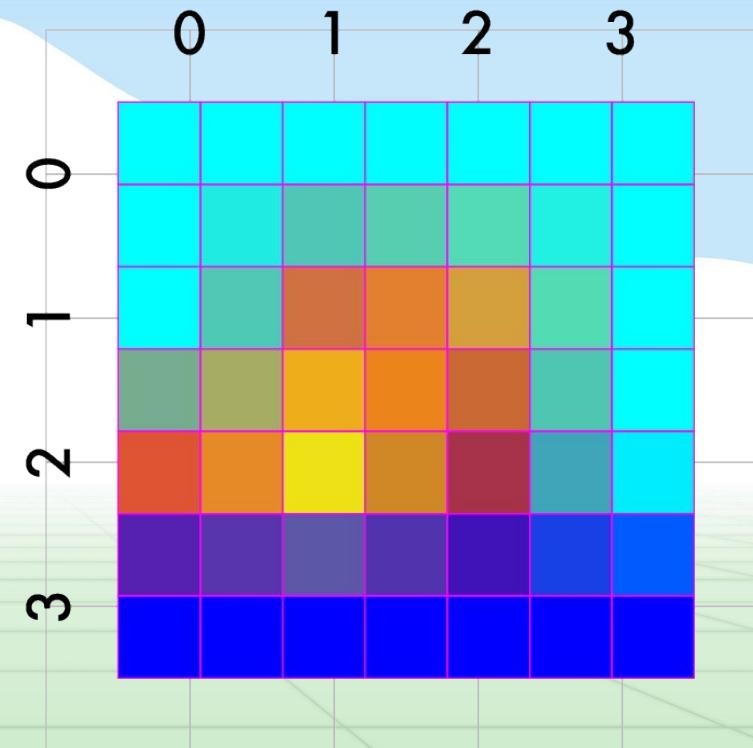
- Create our new image
  - Match up coordinates
    - $4/7 X - 3/14 = Y$
  - Iterate over new pts
    - Map to old coords
    - (1, 3)  $\rightarrow$  (5/14, 7/14)
    - Interpolate old values
      - $q = (166, 172.5, 100)$
  - Fill in the rest



# Resize 4x4 -> 7x7

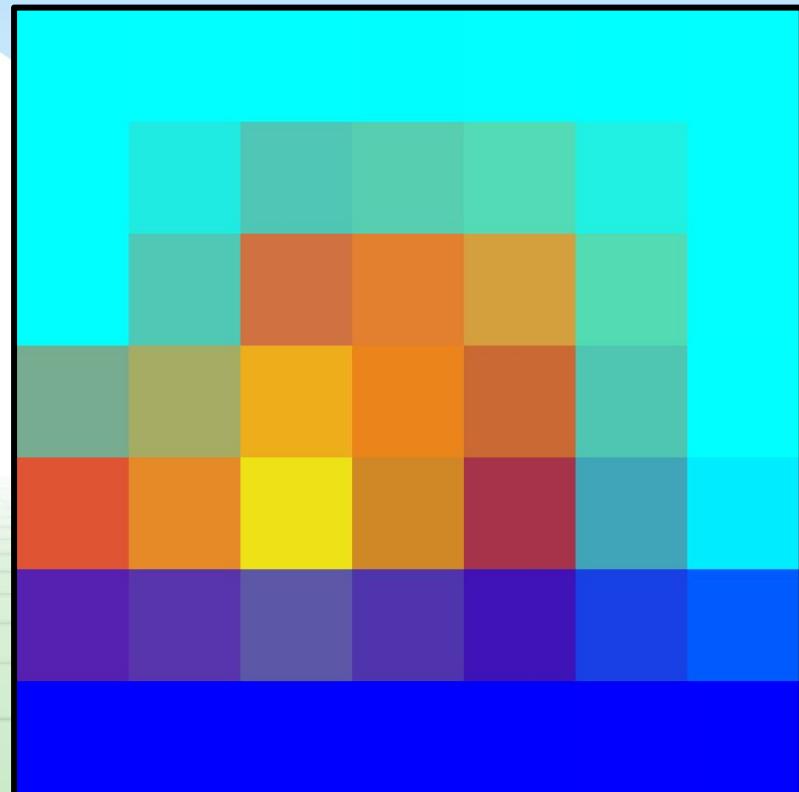
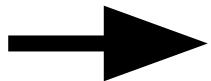
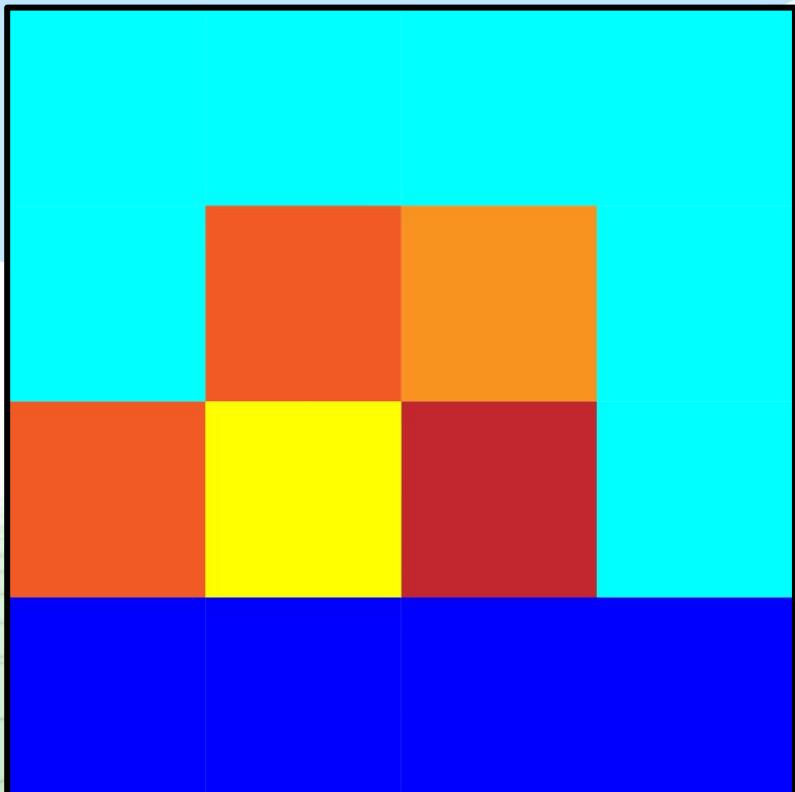
---

- Create our new image
- Match up coordinates
  - $4/7 X - 3/14 = Y$
- Iterate over new pts
  - Map to old coords
    - $(1, 3) \rightarrow (5/14, 7/14)$
  - Interpolate old values
    - $q = (166, 172.5, 100)$
- Fill in the rest



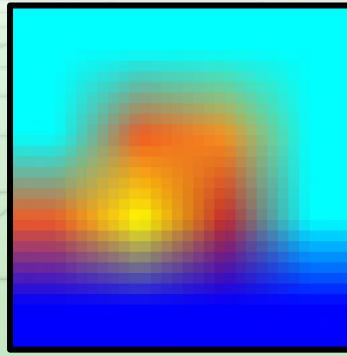
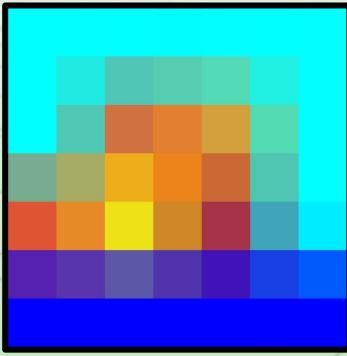
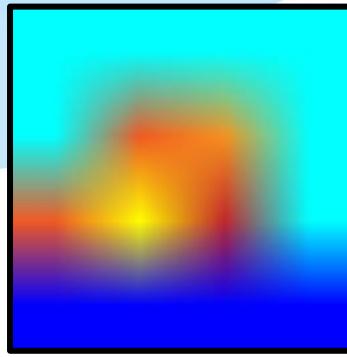
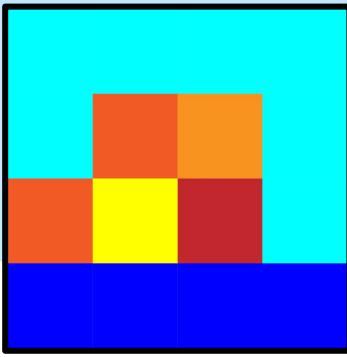
# We did it!

---



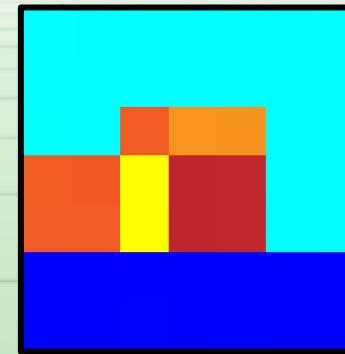
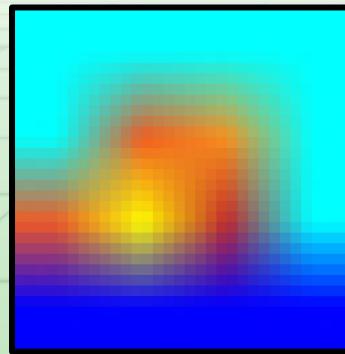
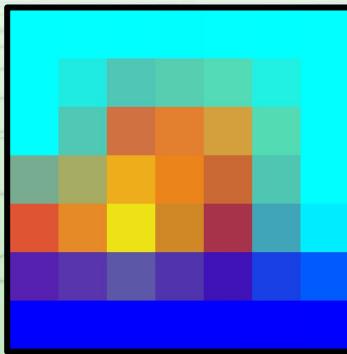
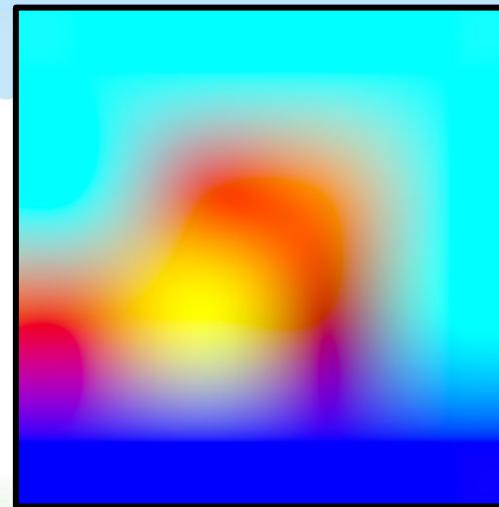
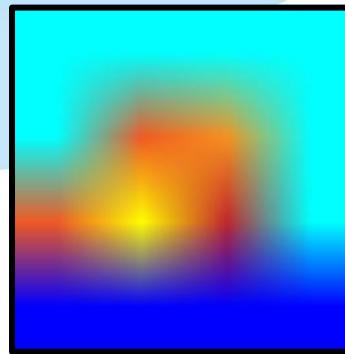
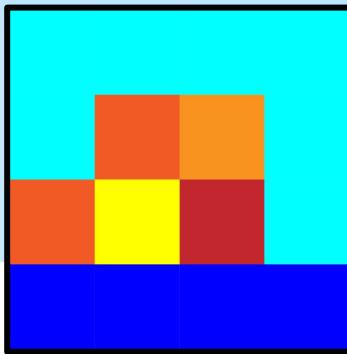
# Different scales

---



# Different methods

---





Let's do something interesting  
already!!

# Want to make image smaller



448x448 -> 64x64

---



448x448 -> 64x64

---



448x448 -> 64x64

---



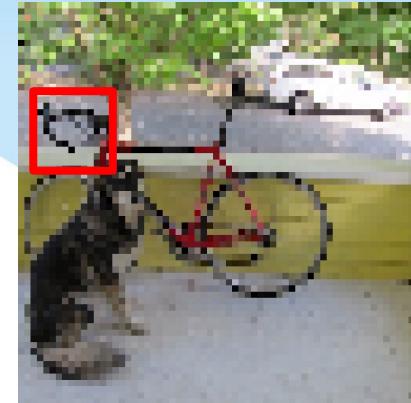
448x448 -> 64x64

---



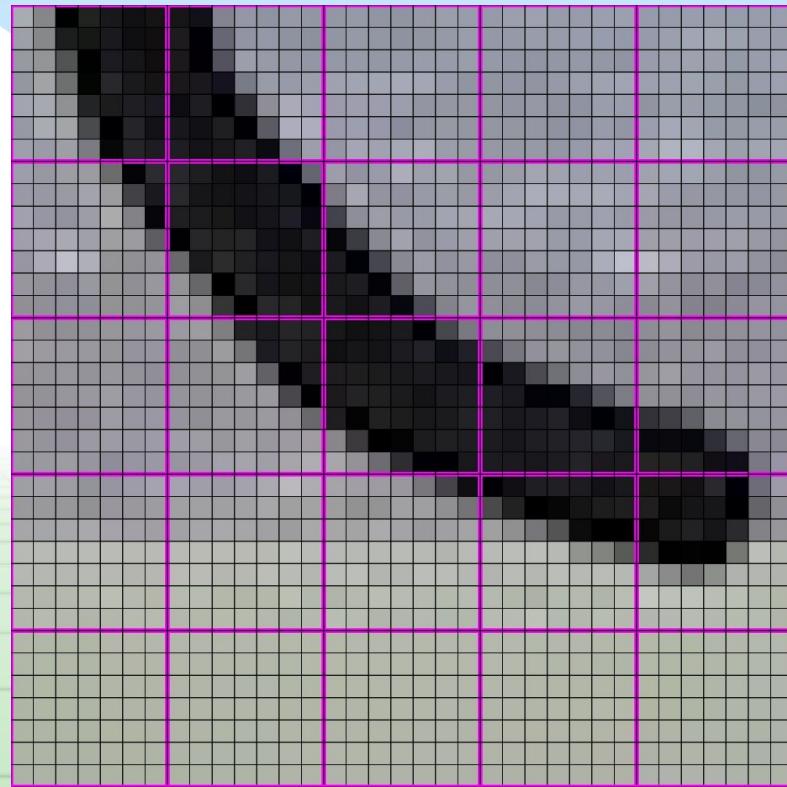
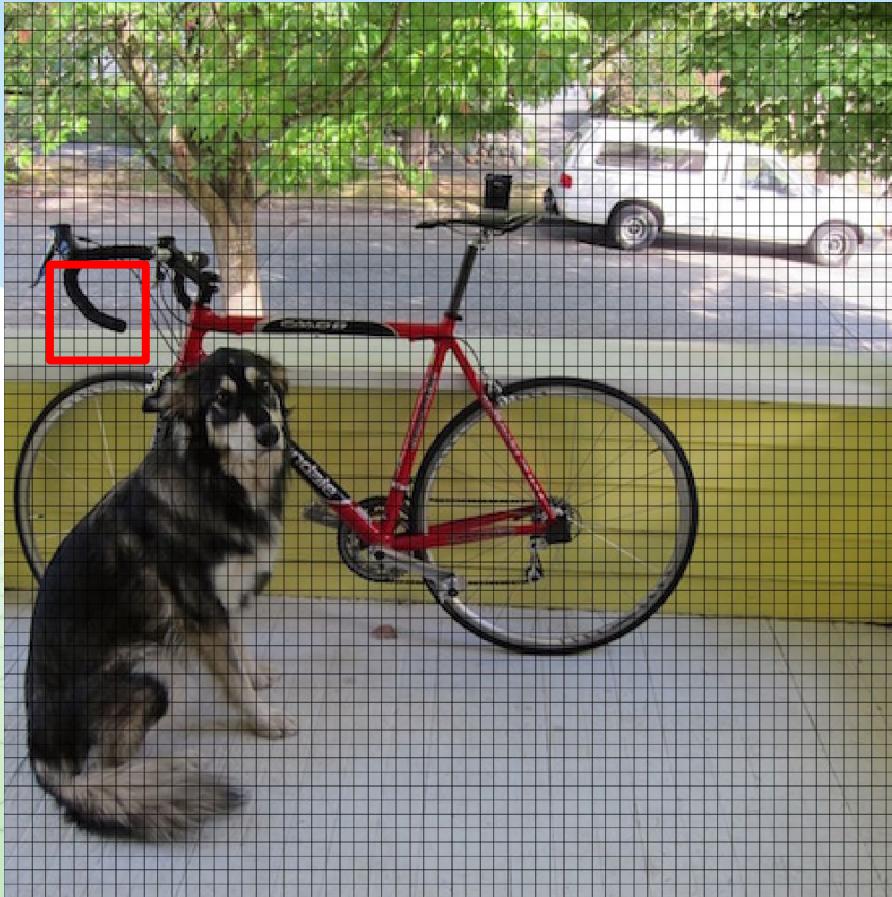
448x448 -> 64x64

---



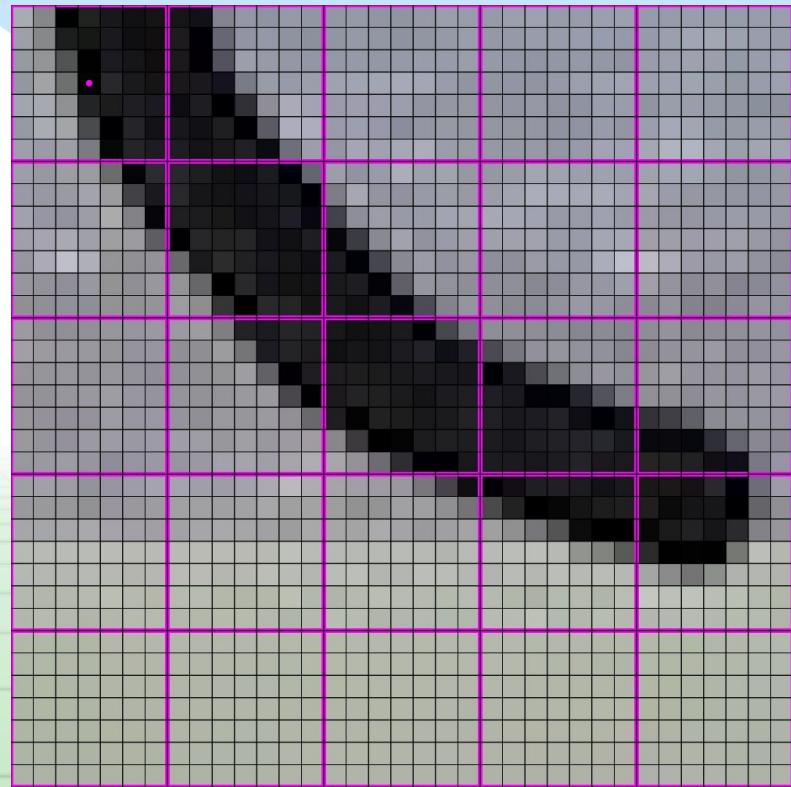
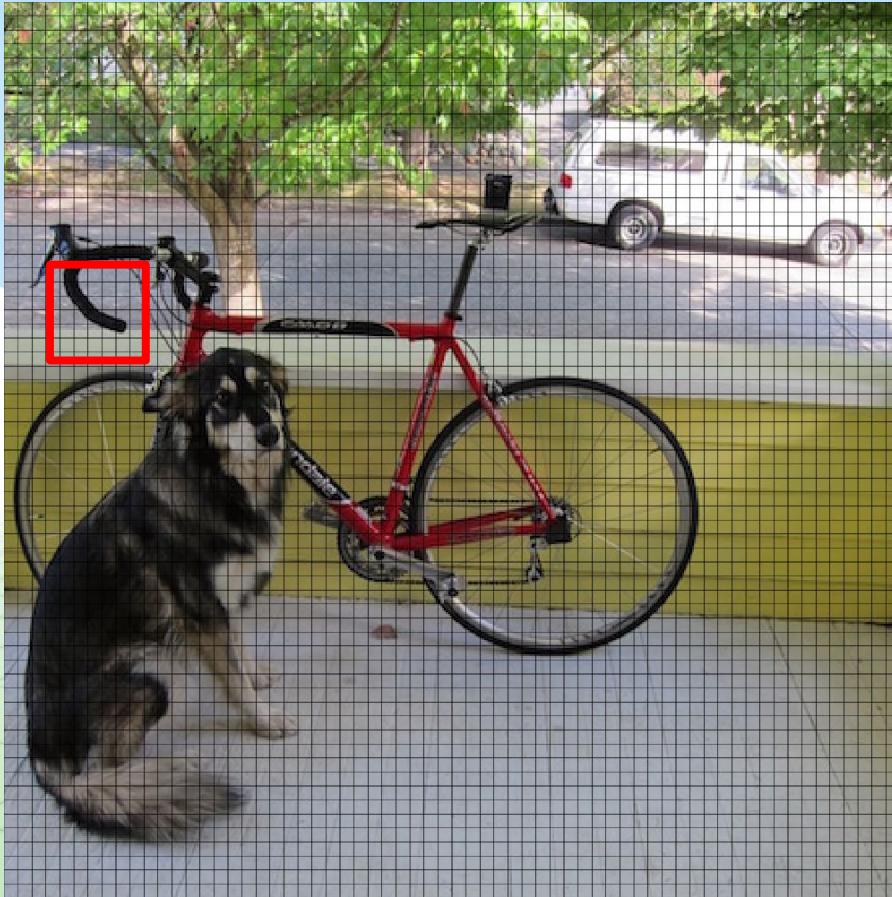
448x448 -> 64x64

---



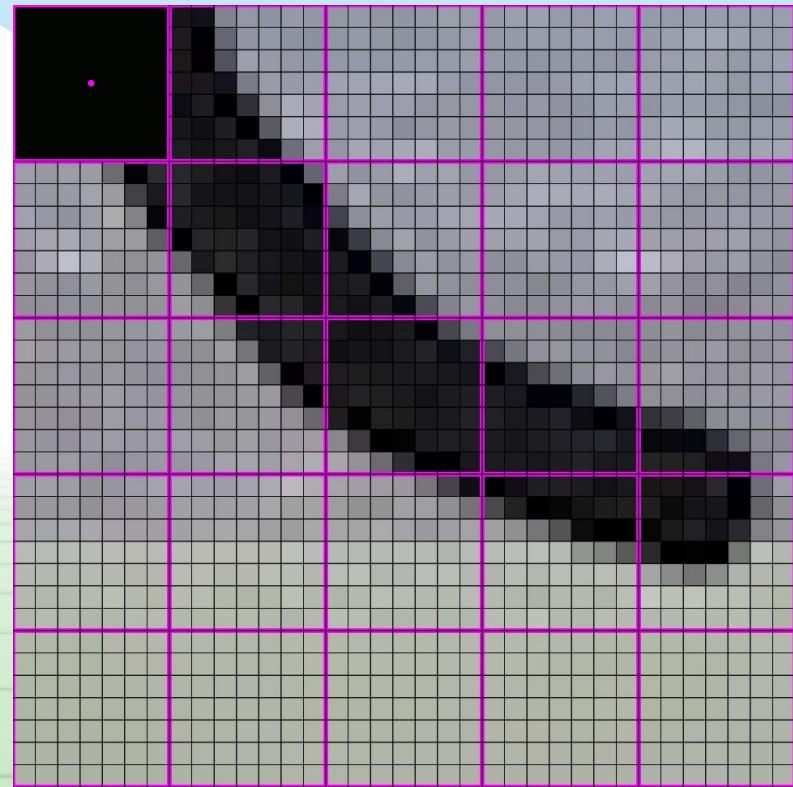
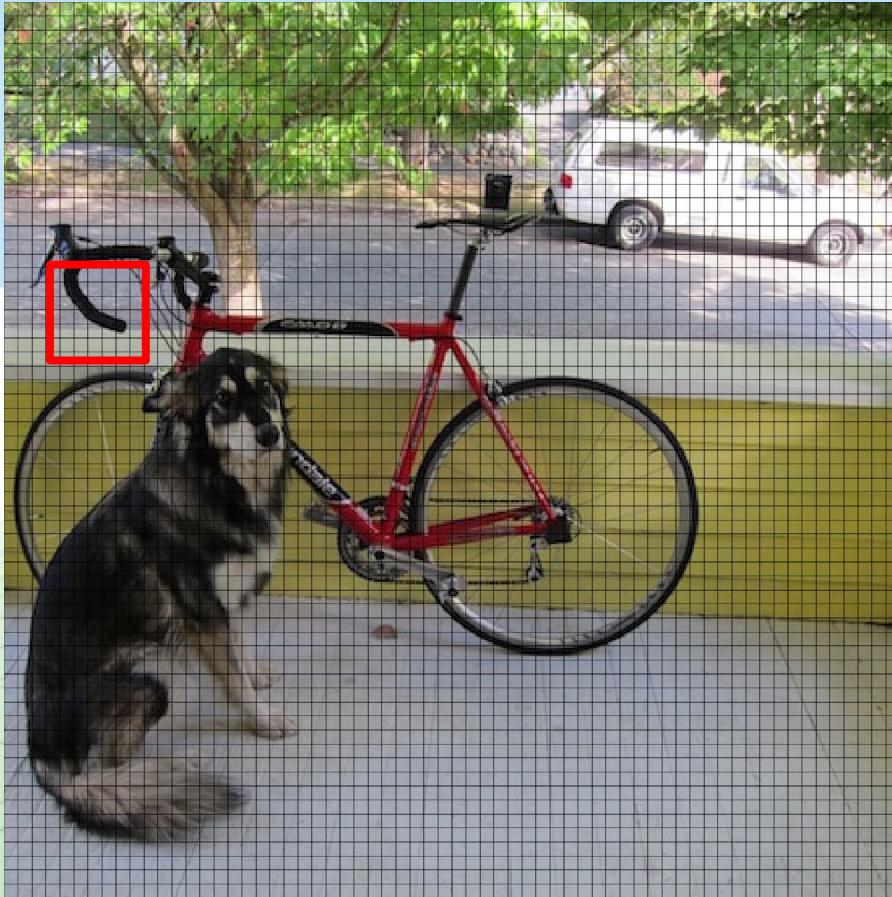
448x448 -> 64x64

---



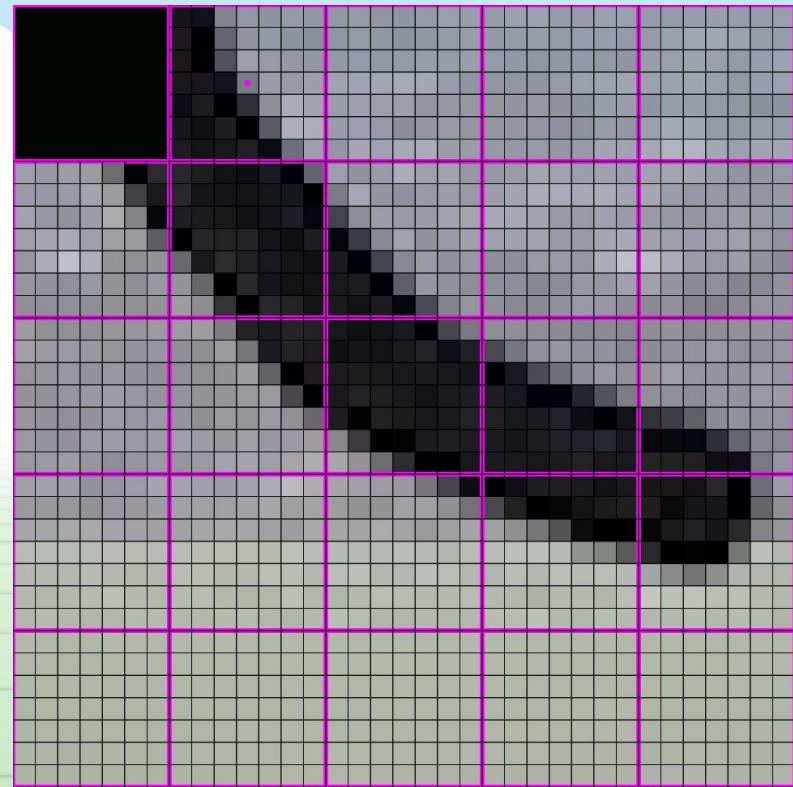
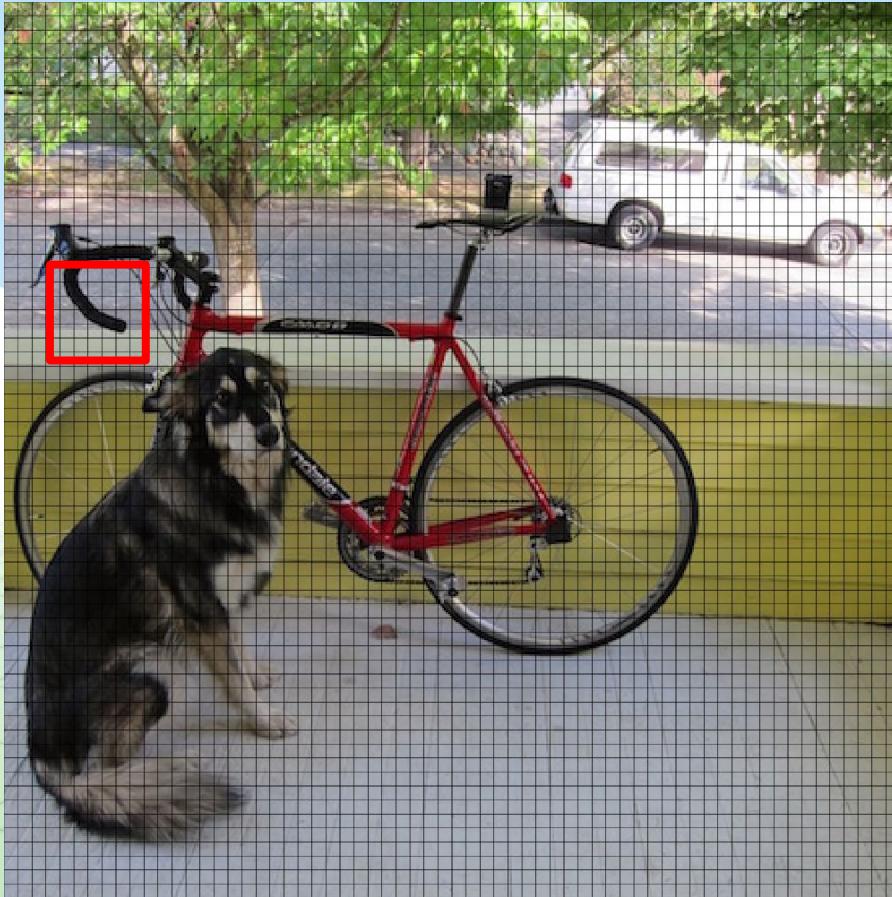
448x448 -> 64x64

---



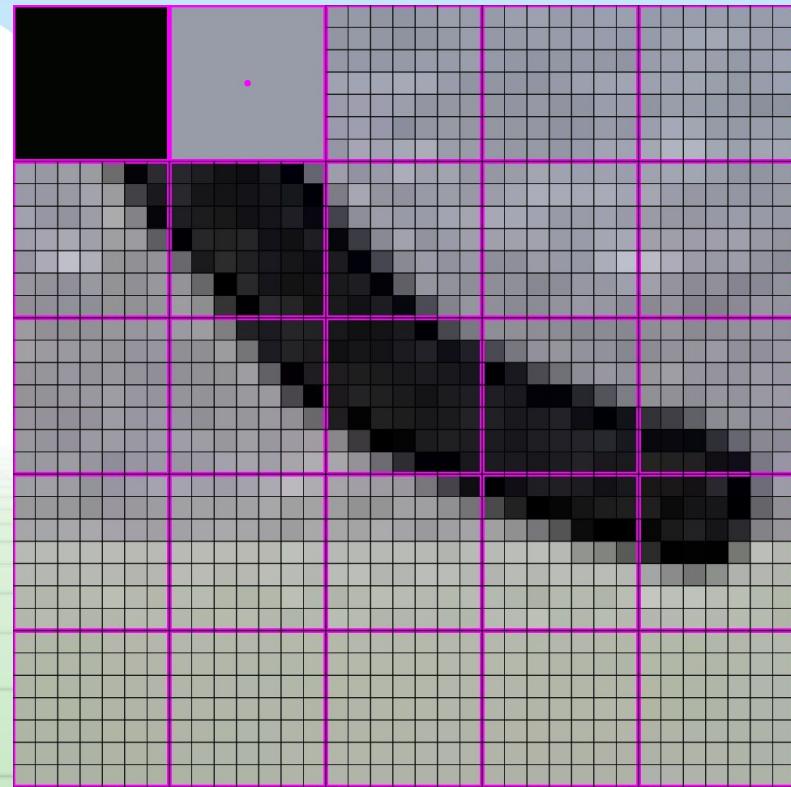
448x448 -> 64x64

---



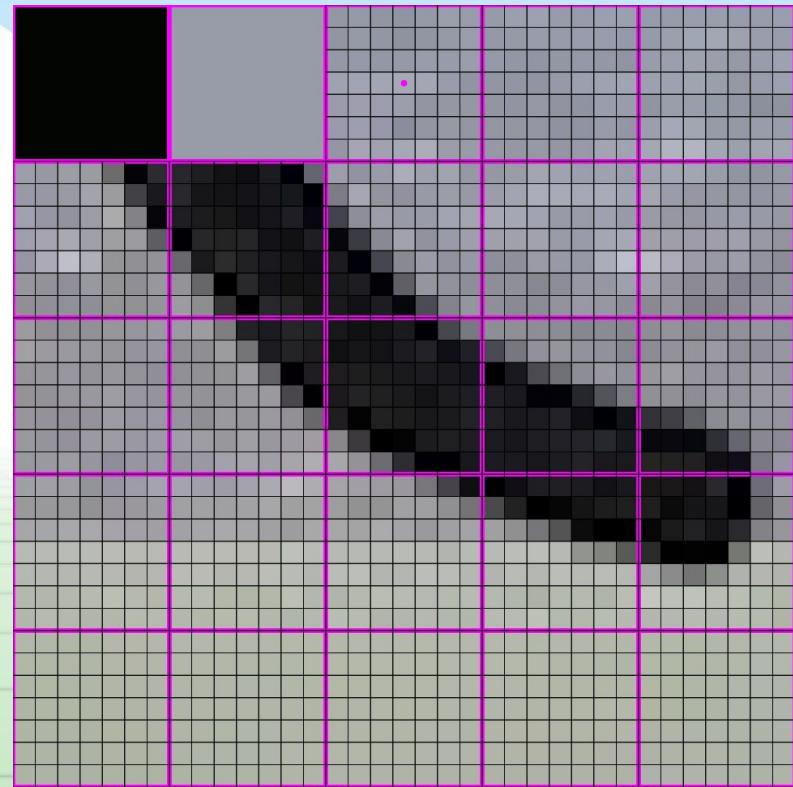
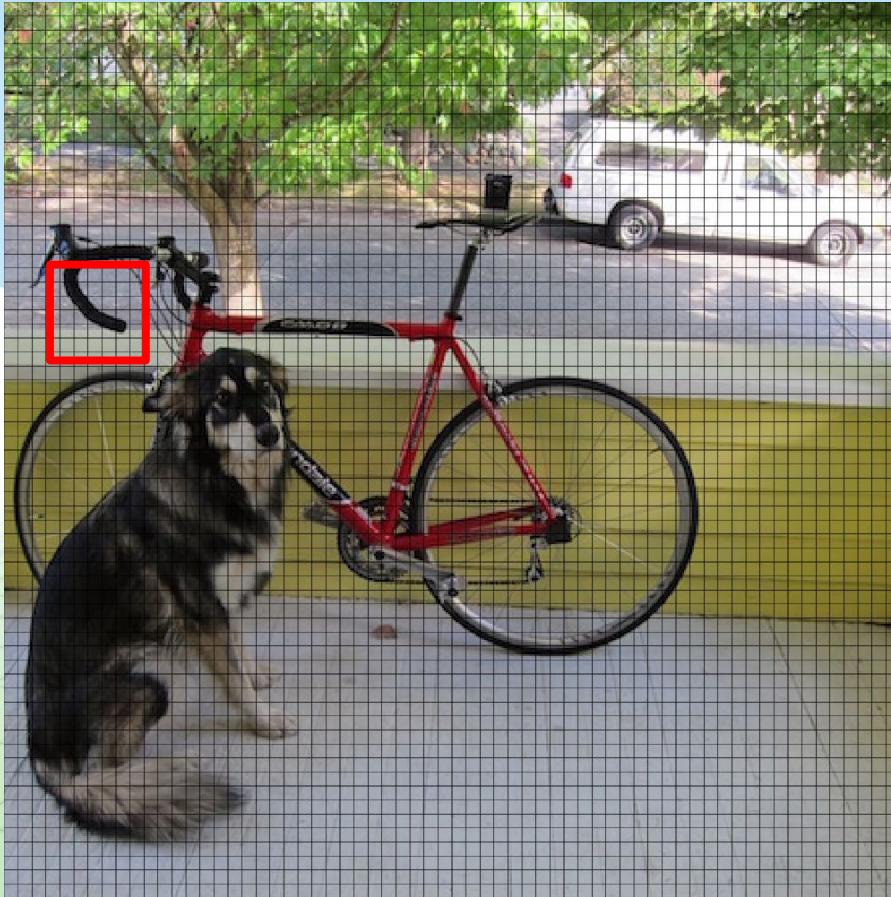
448x448 -> 64x64

---



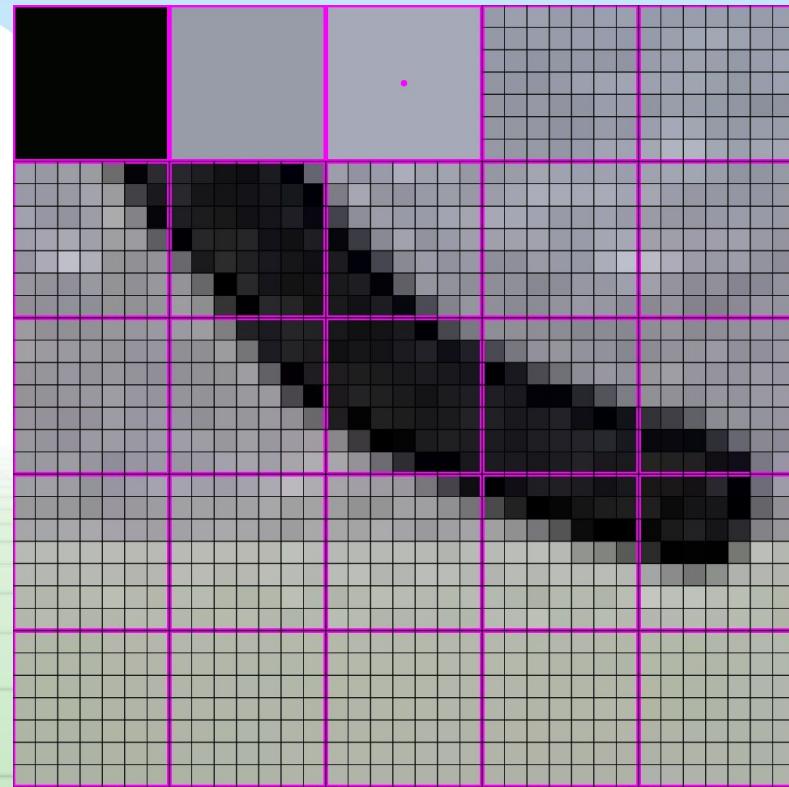
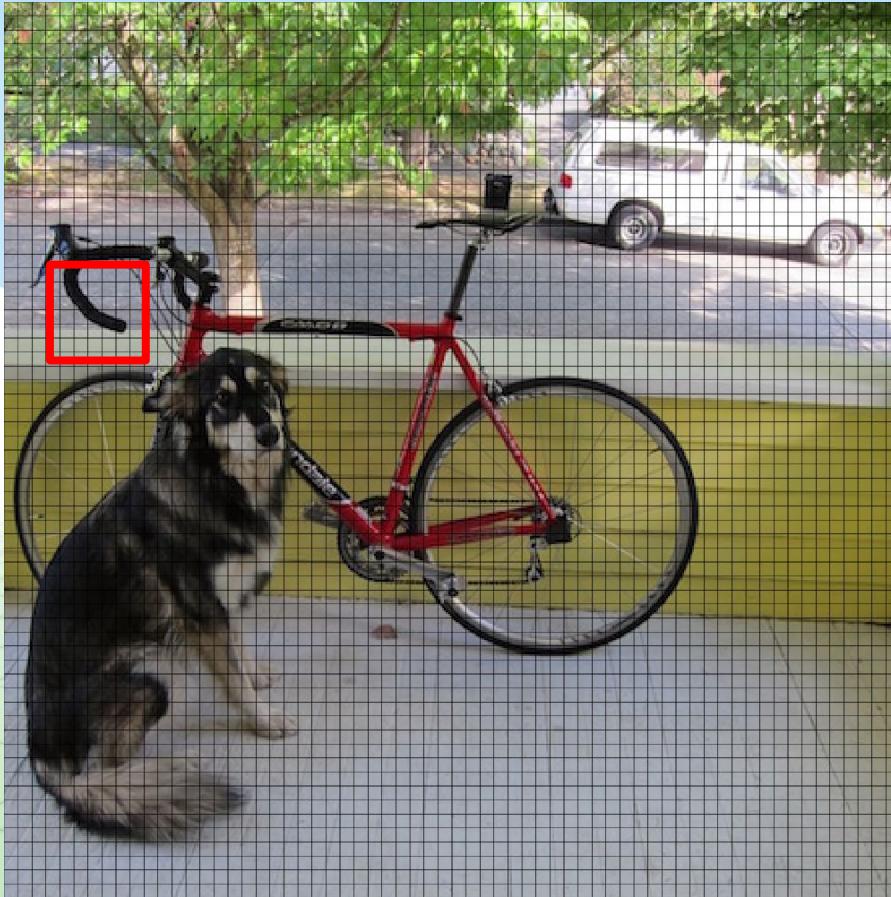
448x448 -> 64x64

---



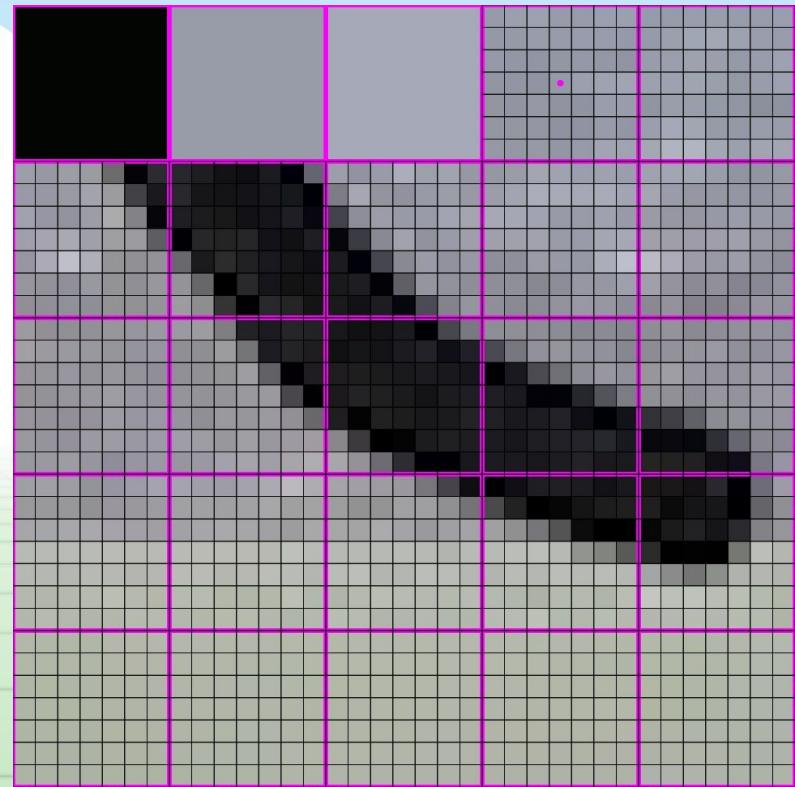
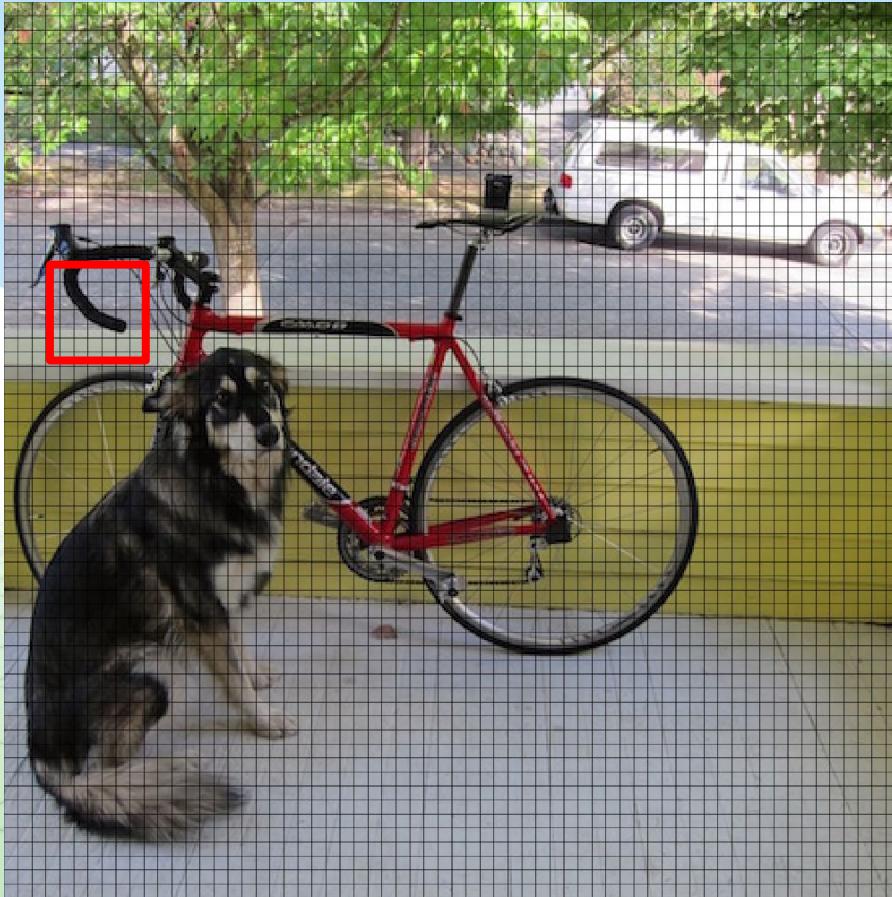
448x448 -> 64x64

---



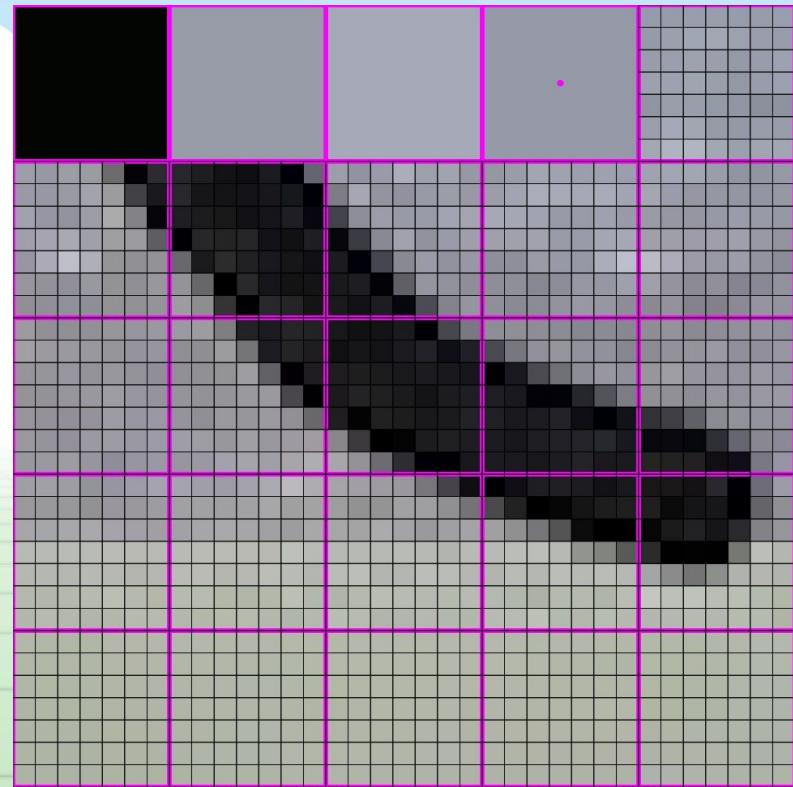
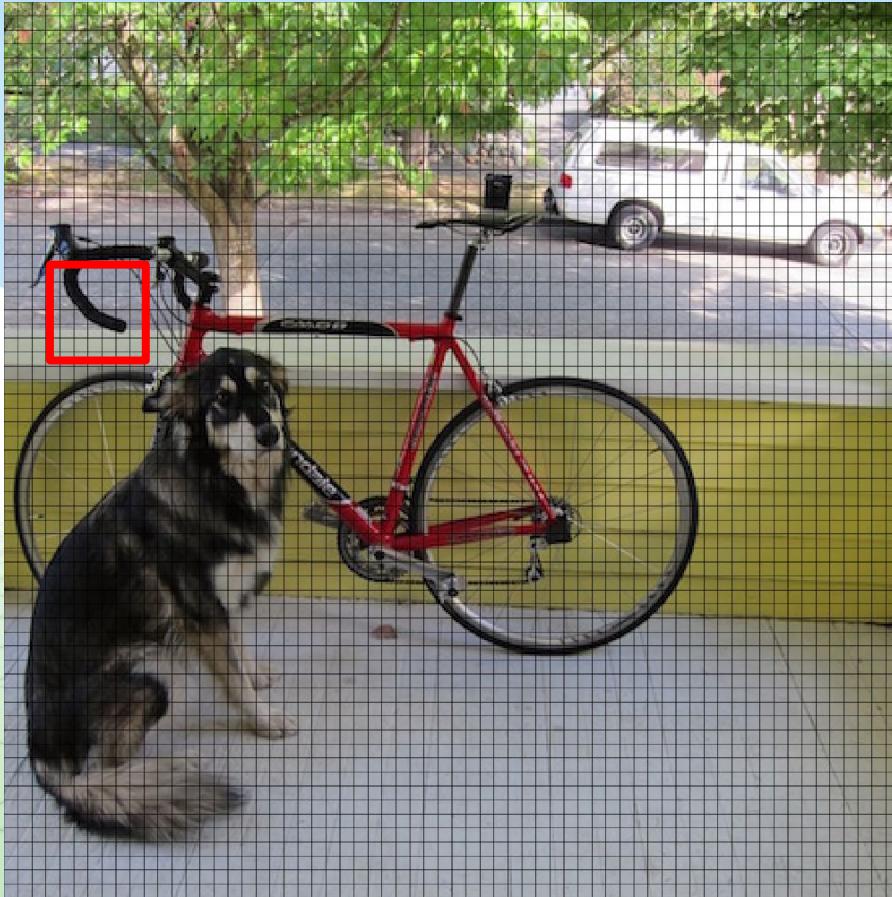
448x448 -> 64x64

---



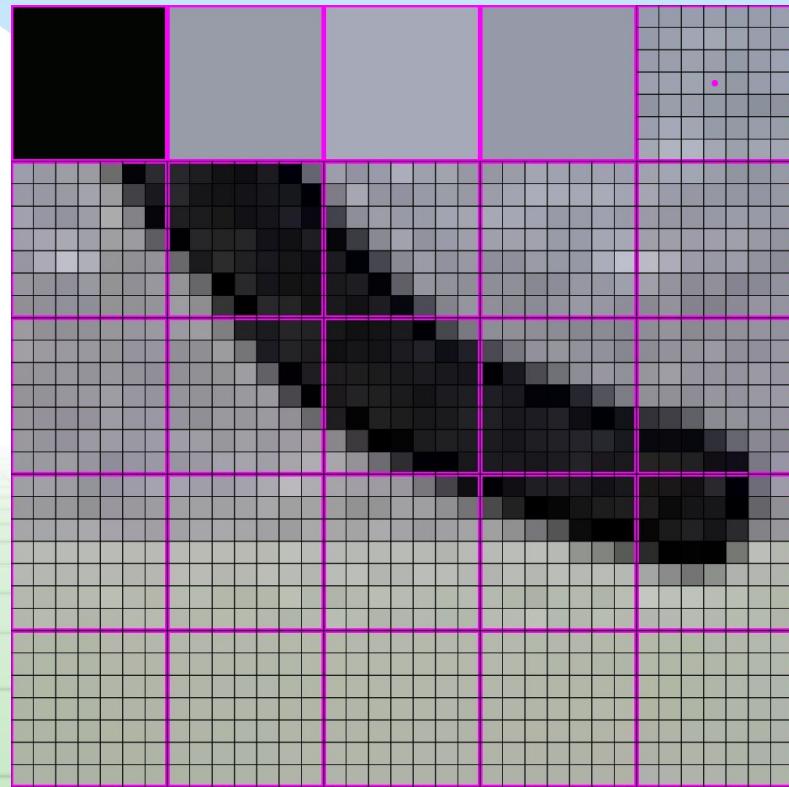
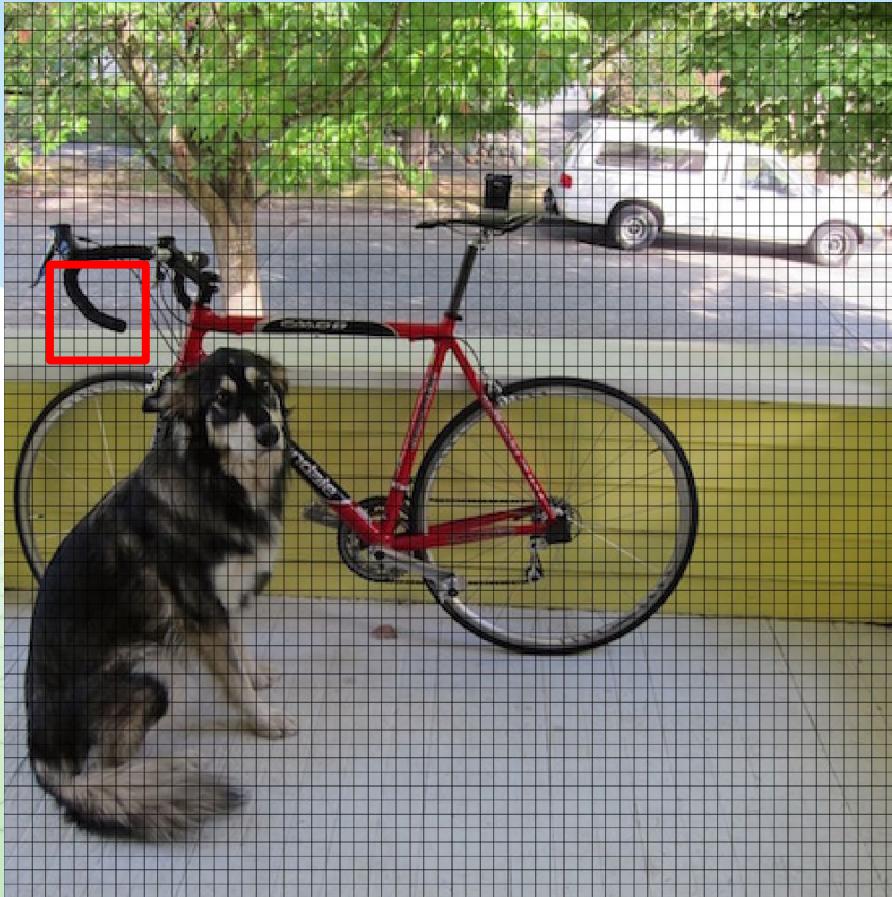
448x448 -> 64x64

---



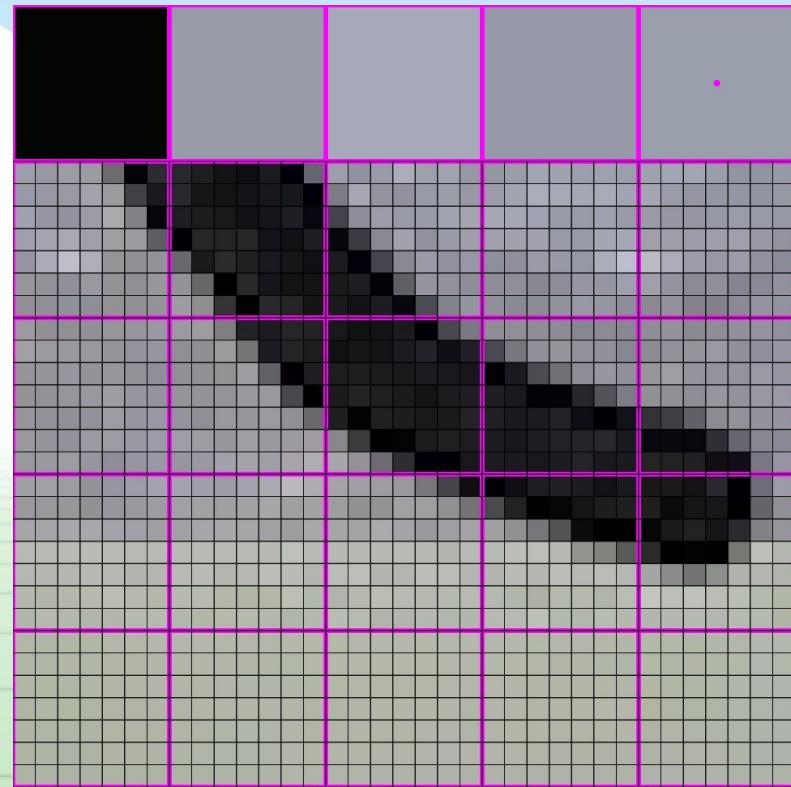
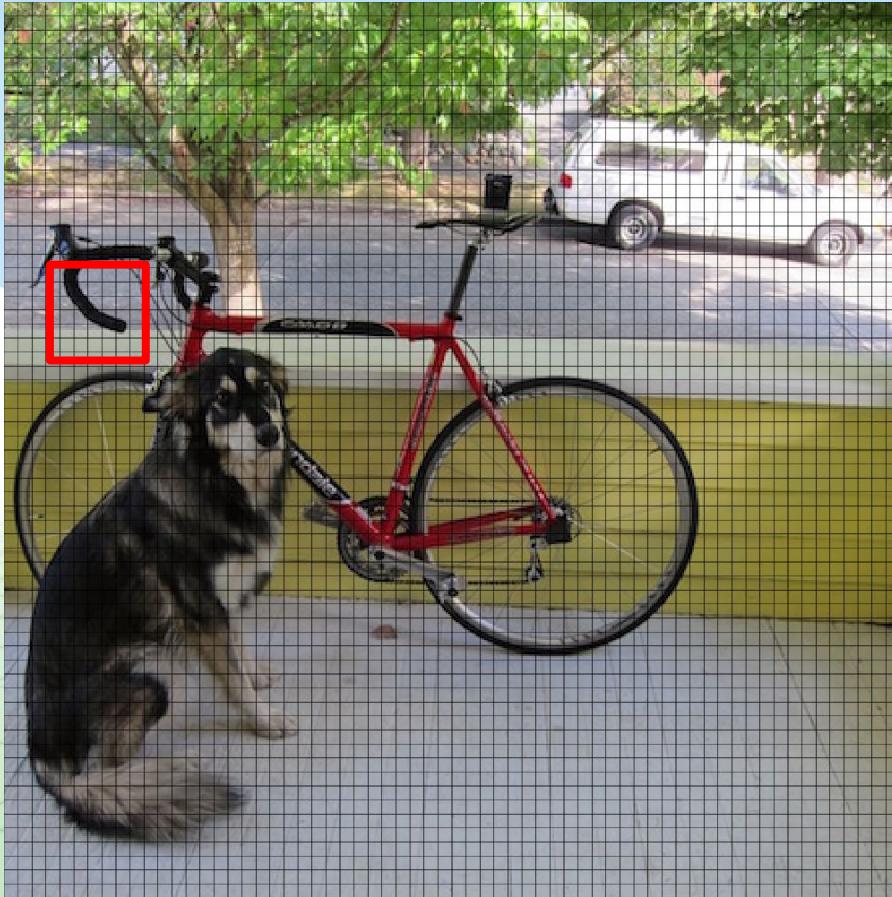
448x448 -> 64x64

---



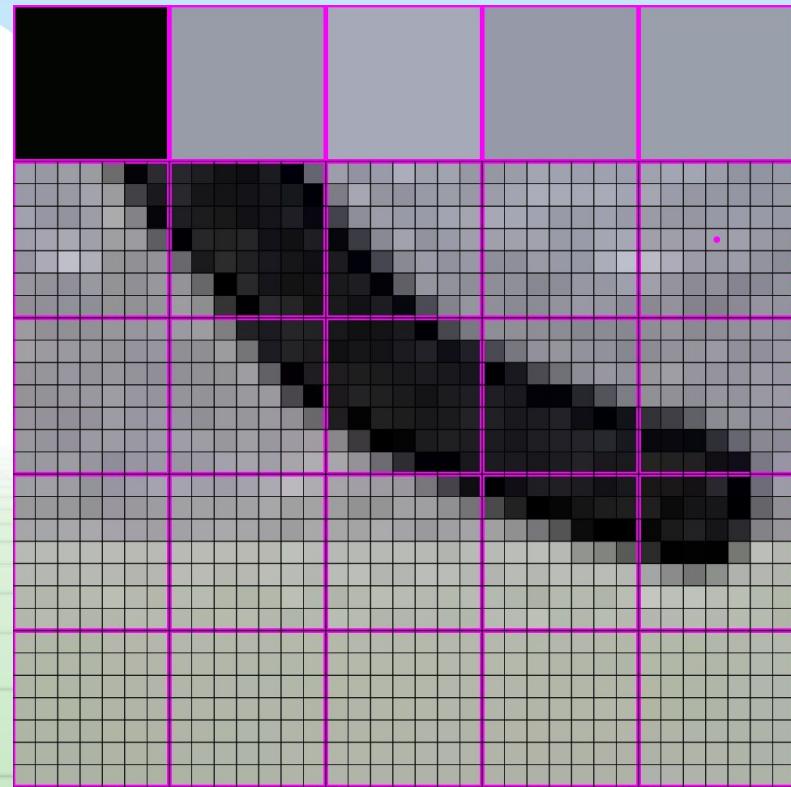
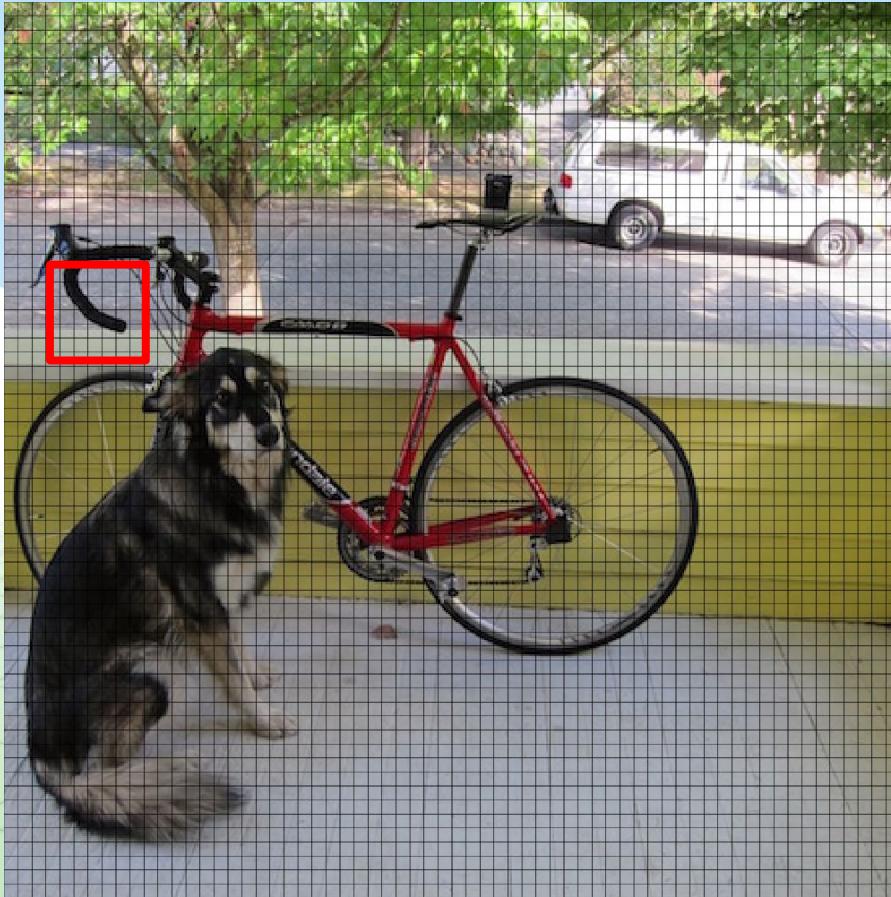
448x448 -> 64x64

---



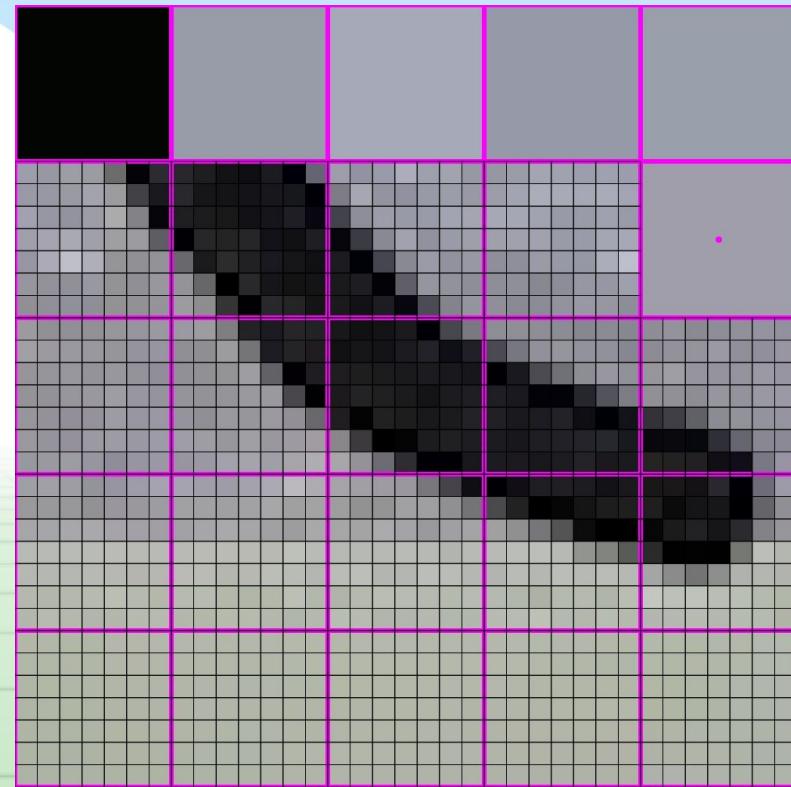
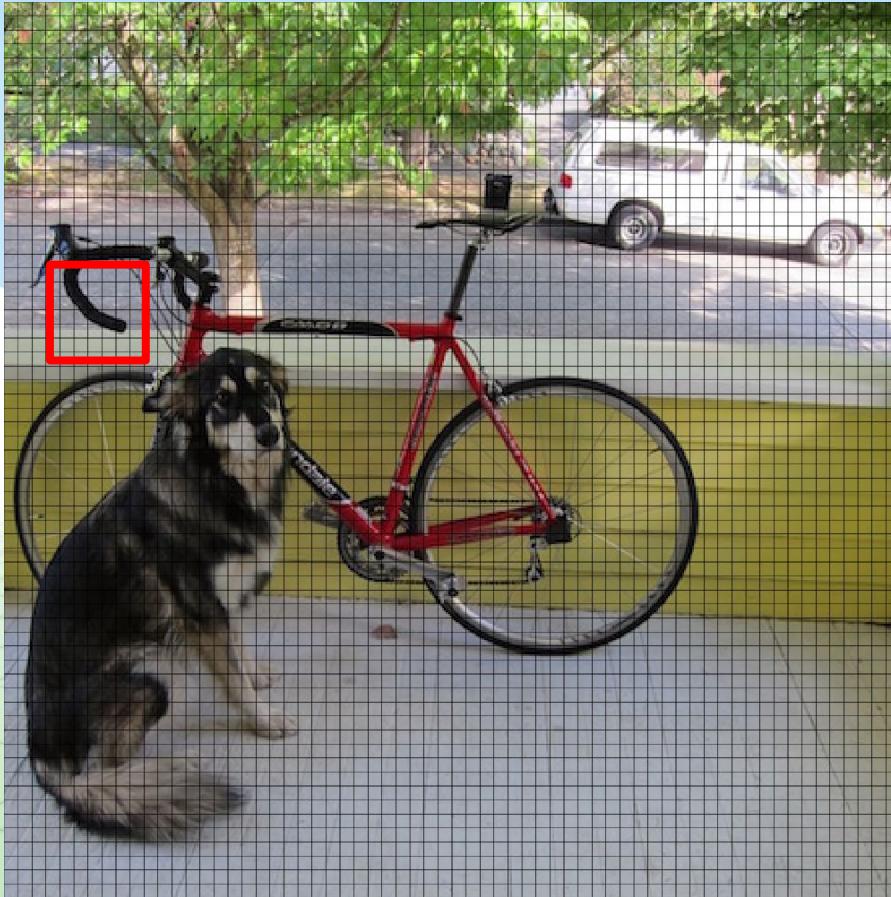
448x448 -> 64x64

---



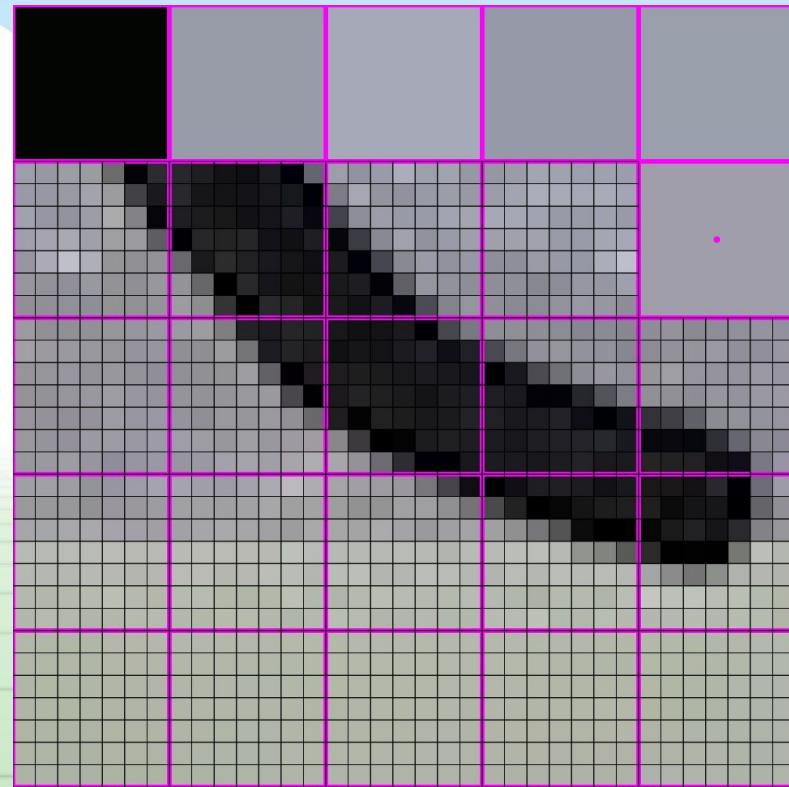
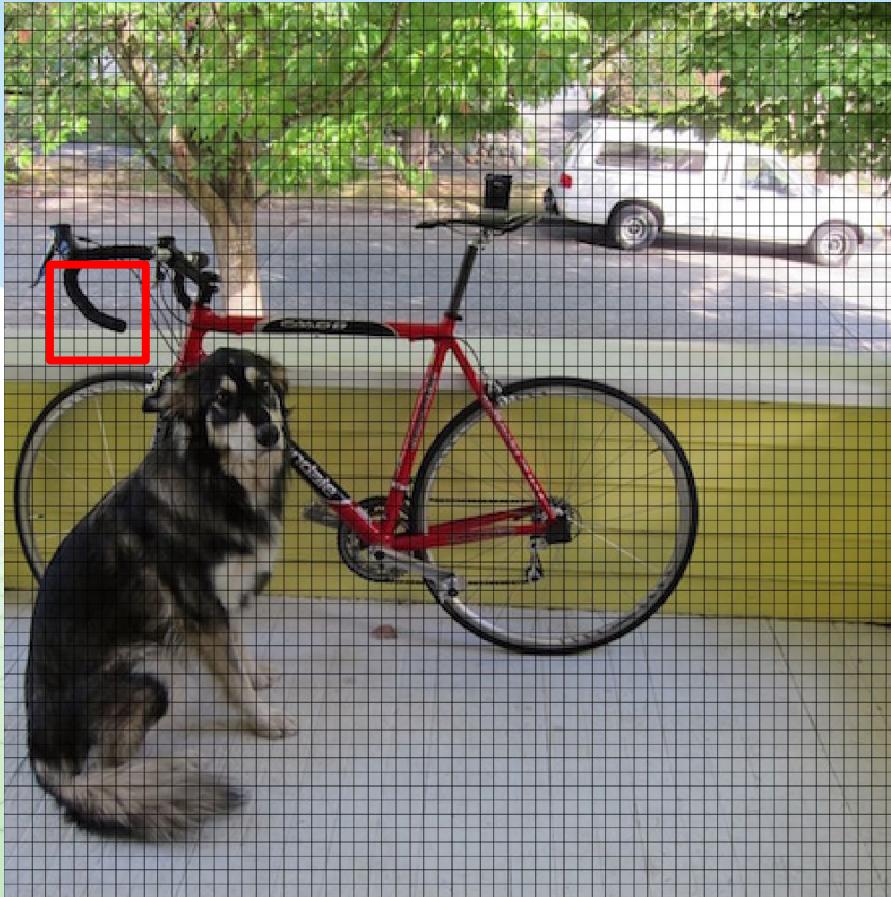
448x448 -> 64x64

---



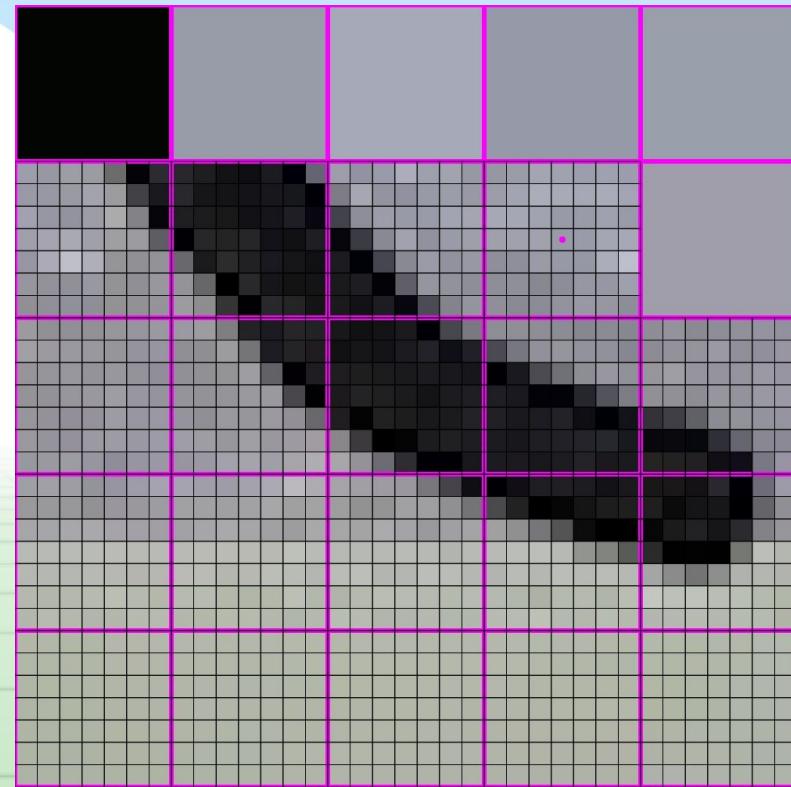
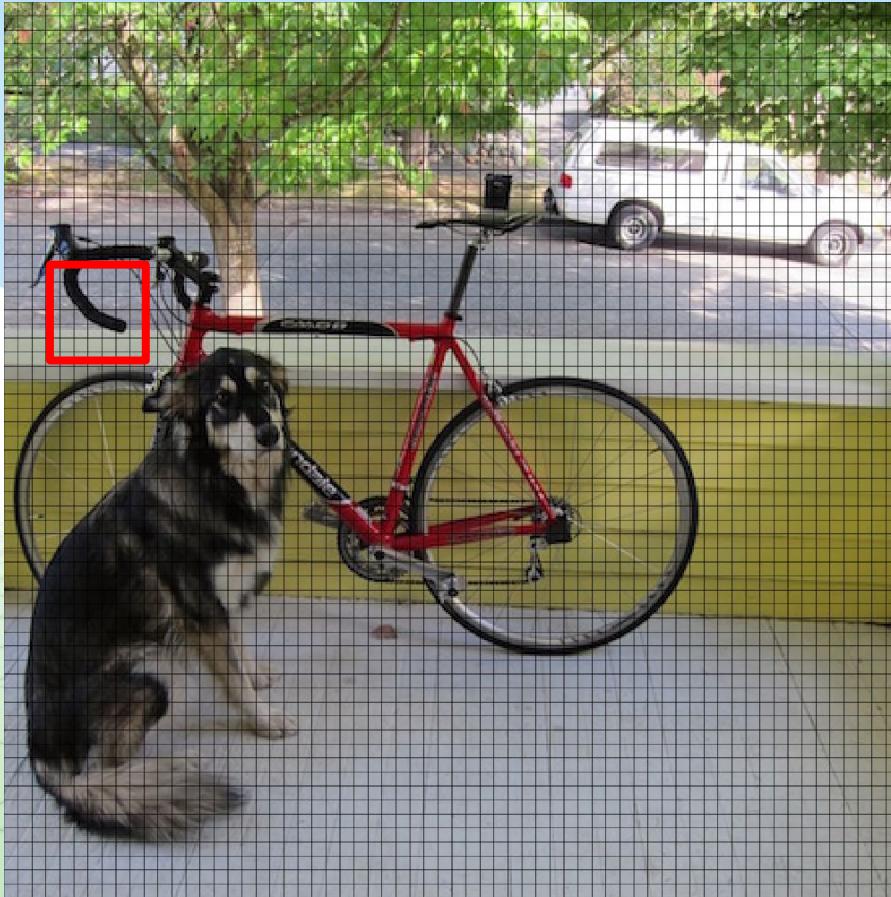
448x448 -> 64x64

---



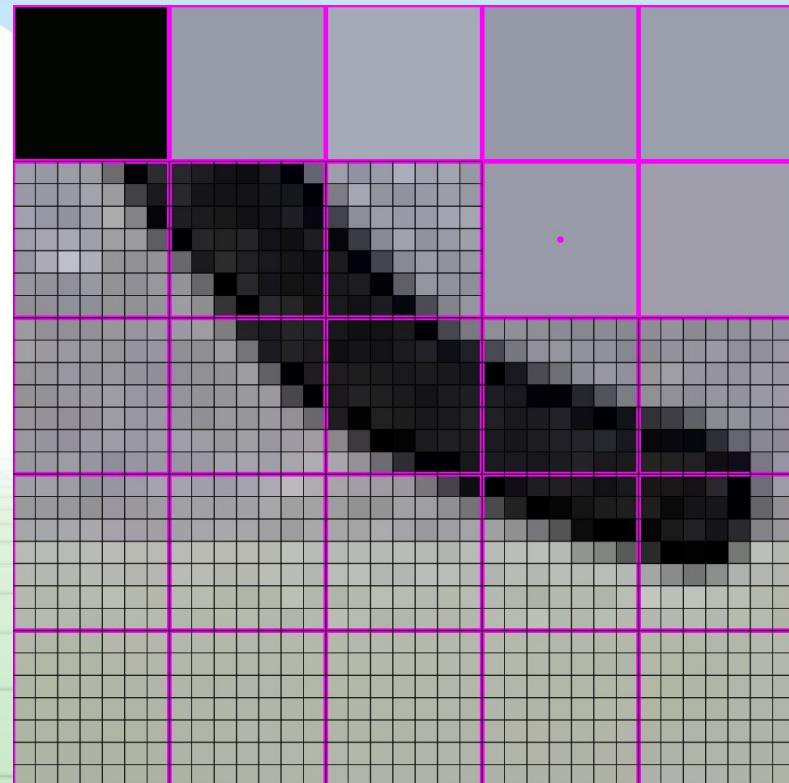
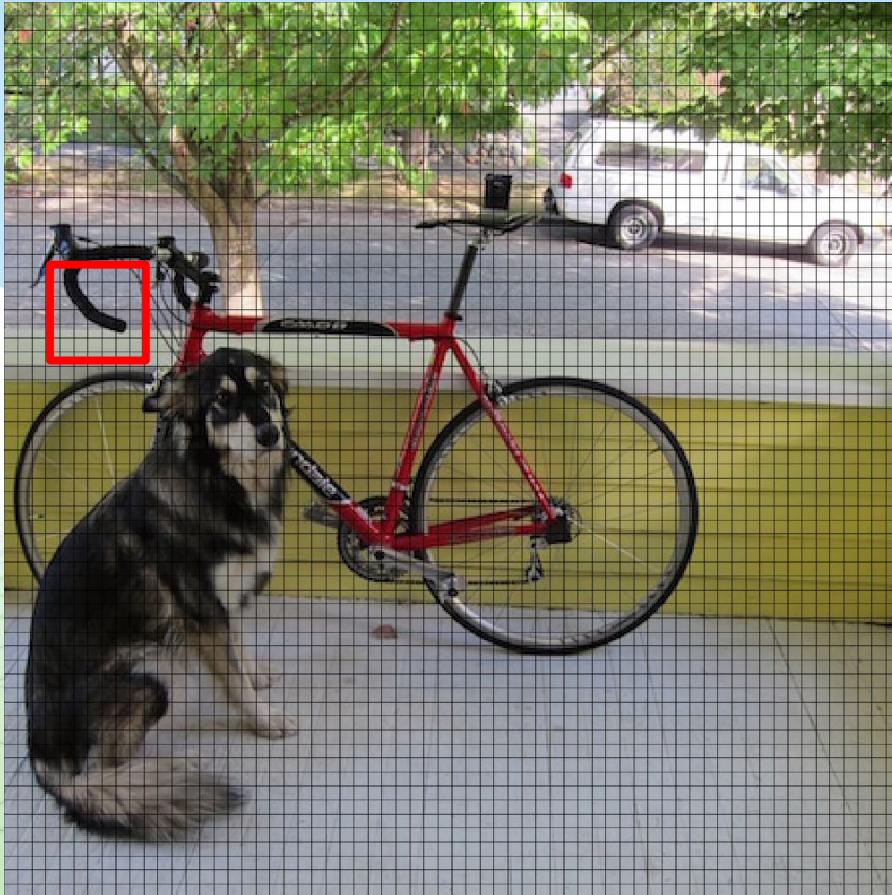
448x448 -> 64x64

---



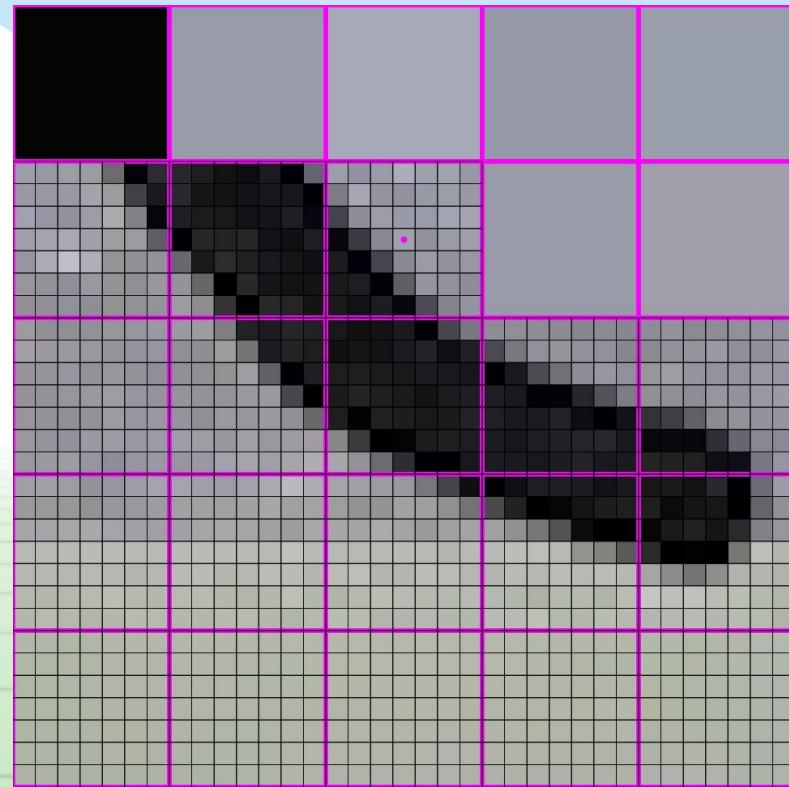
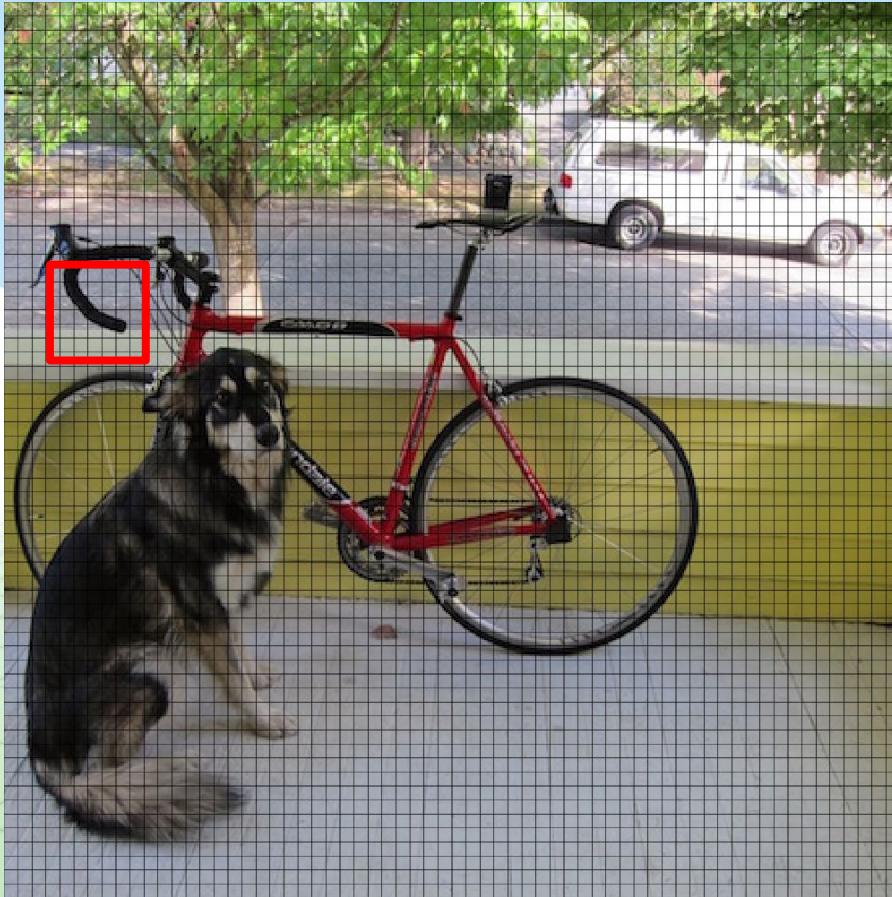
448x448 -> 64x64

---



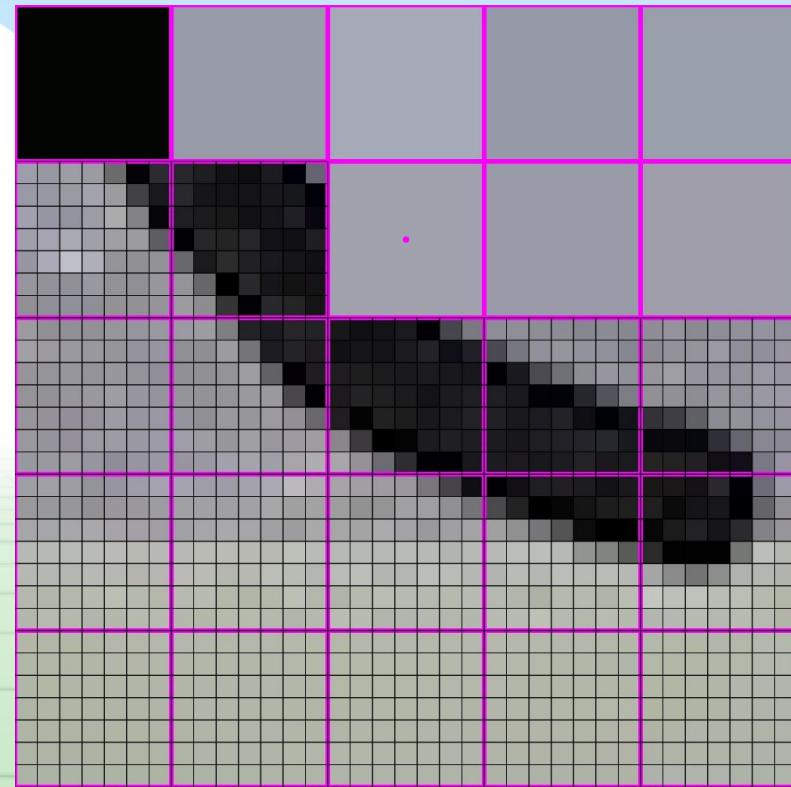
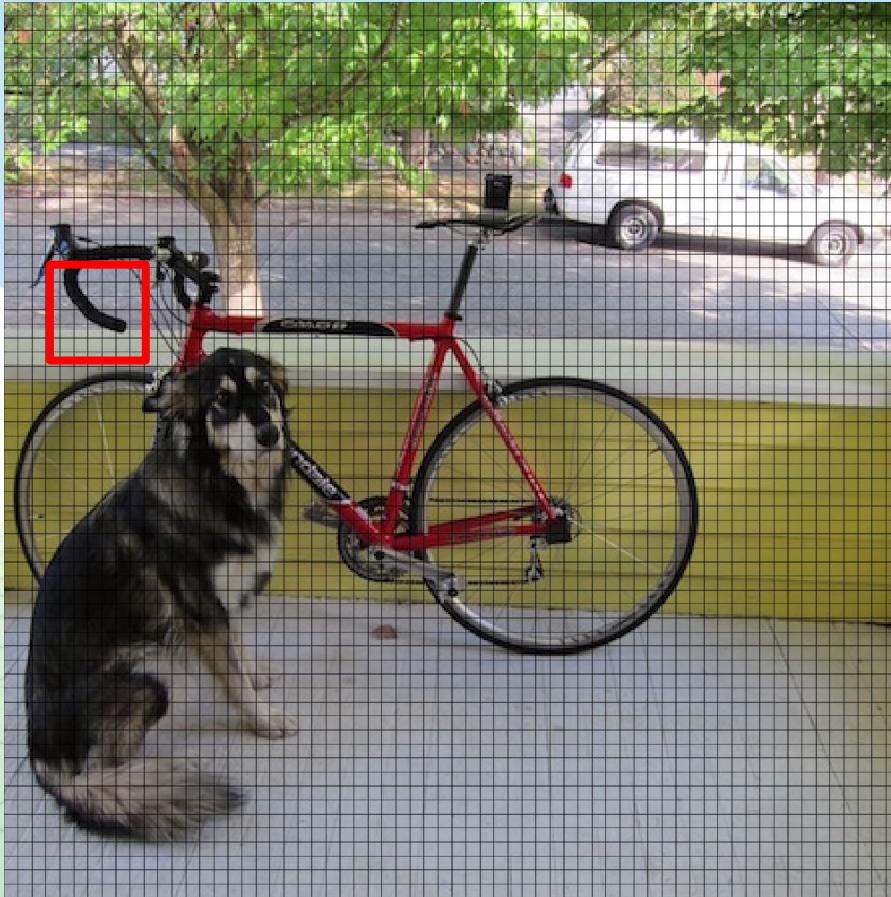
448x448 -> 64x64

---



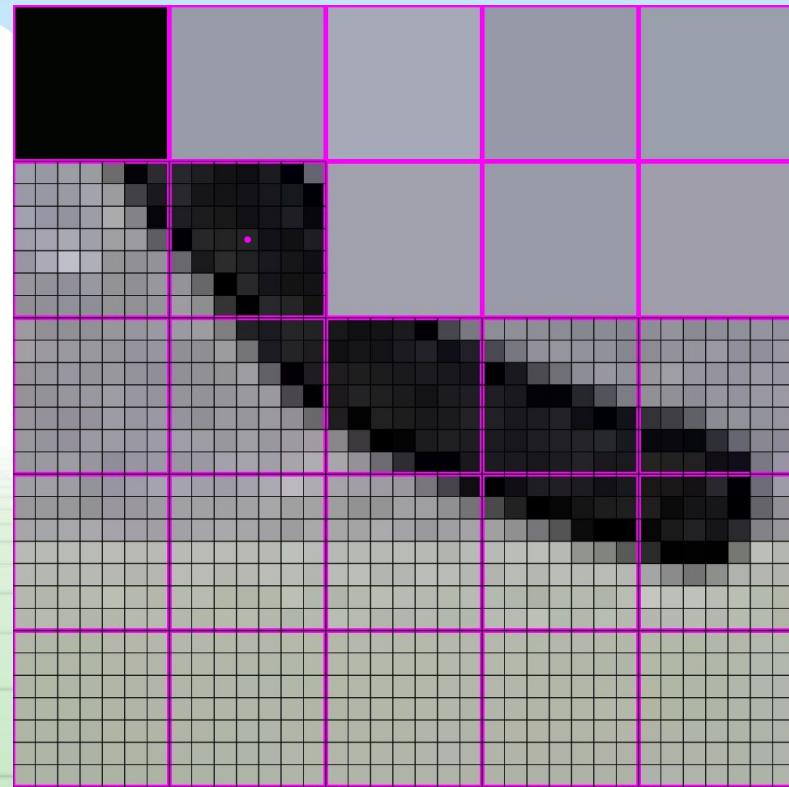
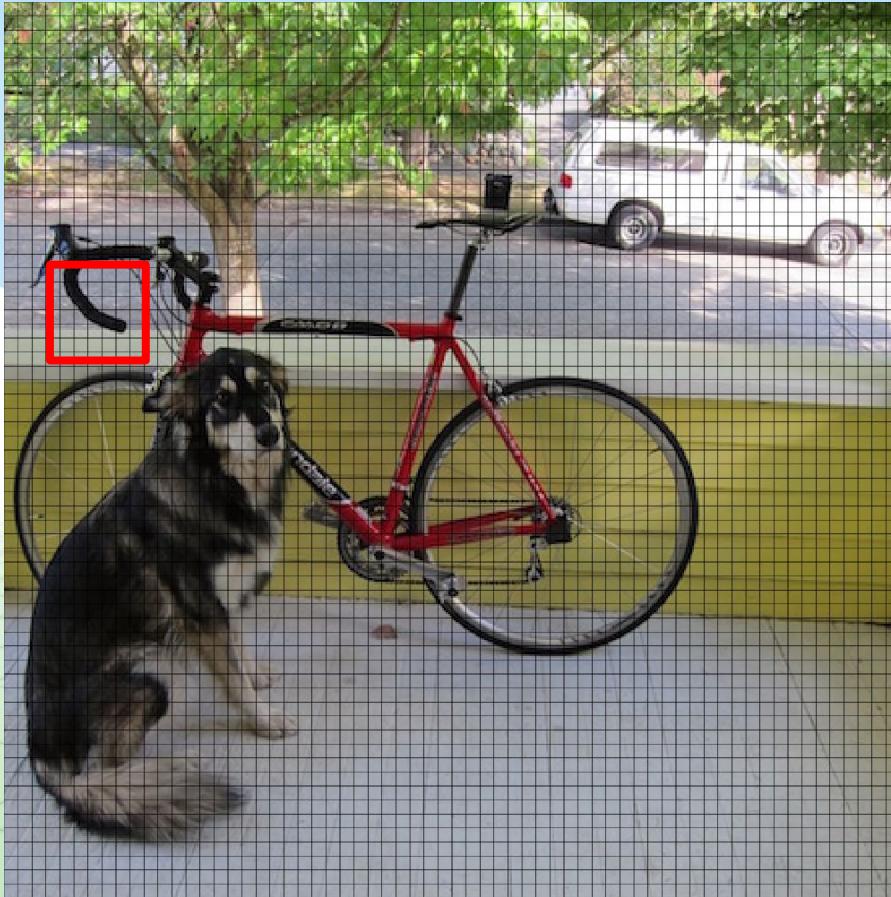
448x448 -> 64x64

---



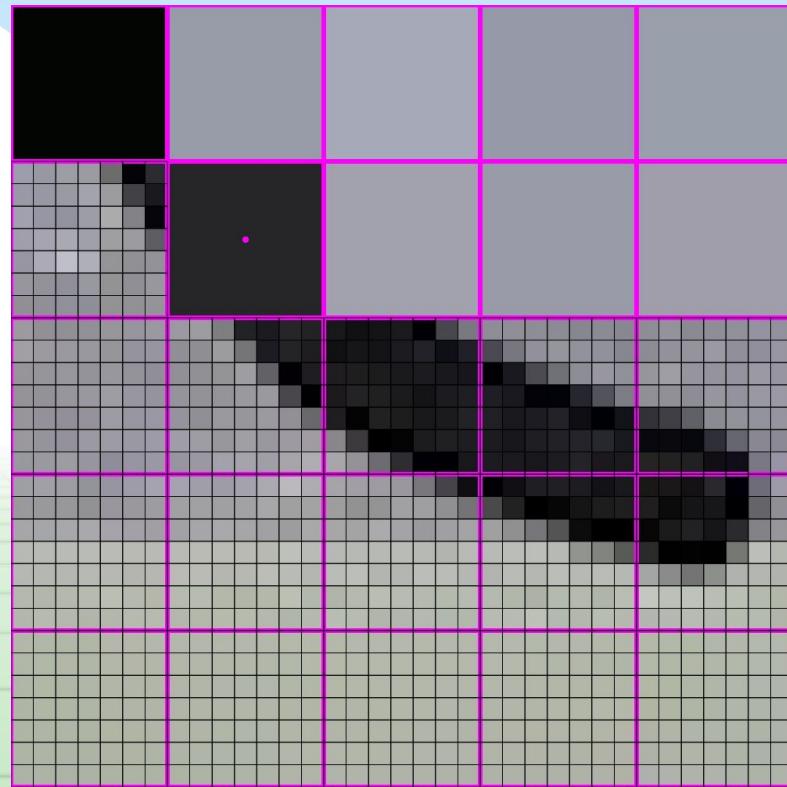
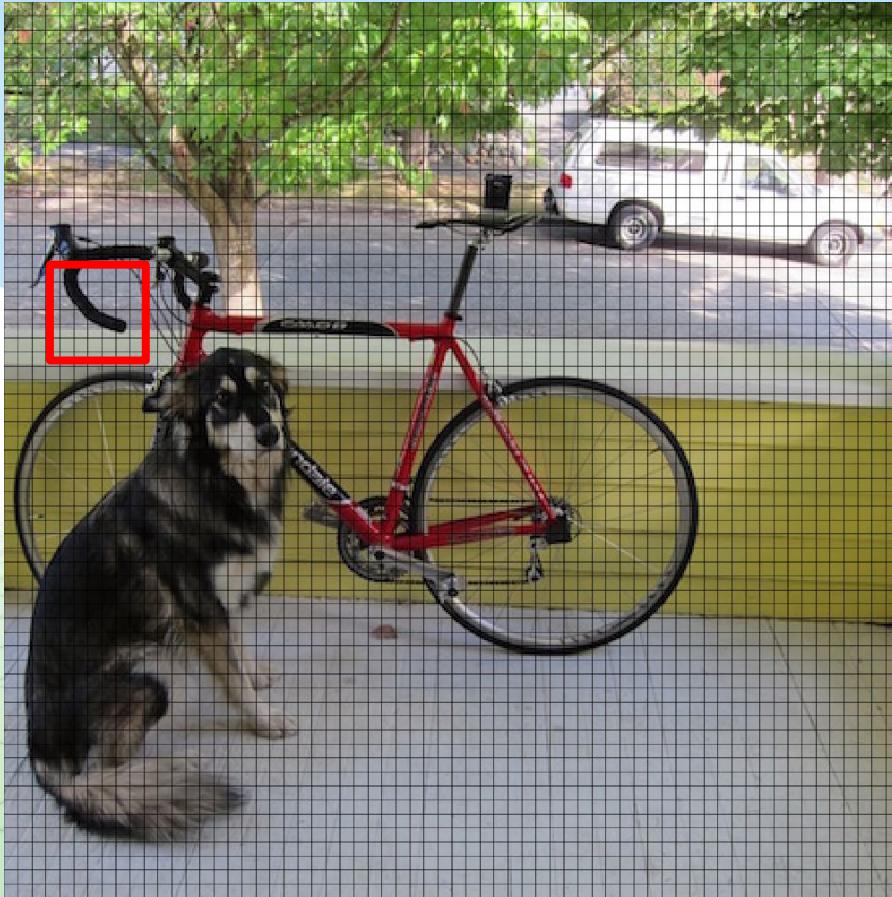
448x448 -> 64x64

---



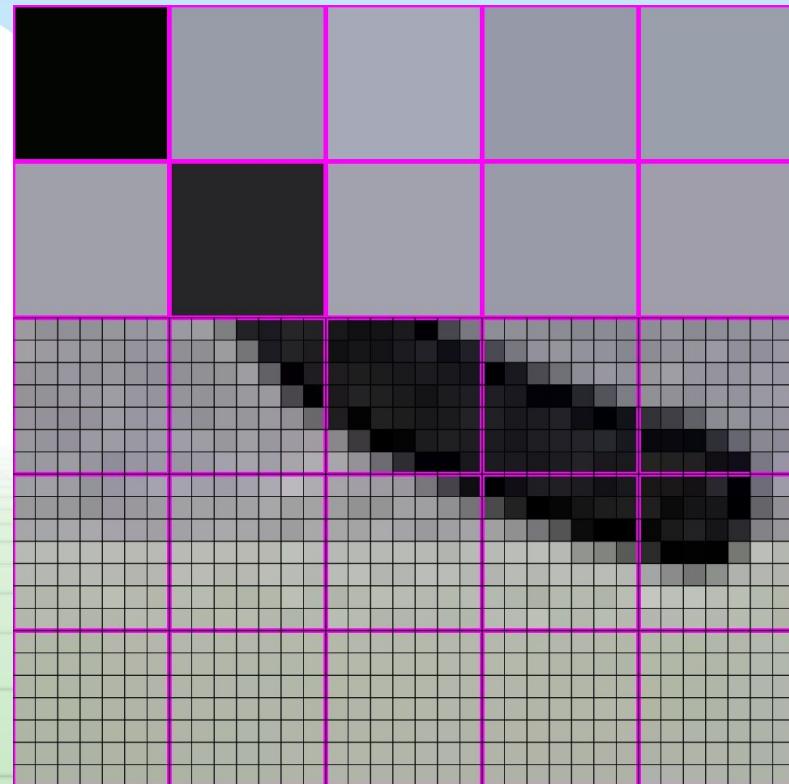
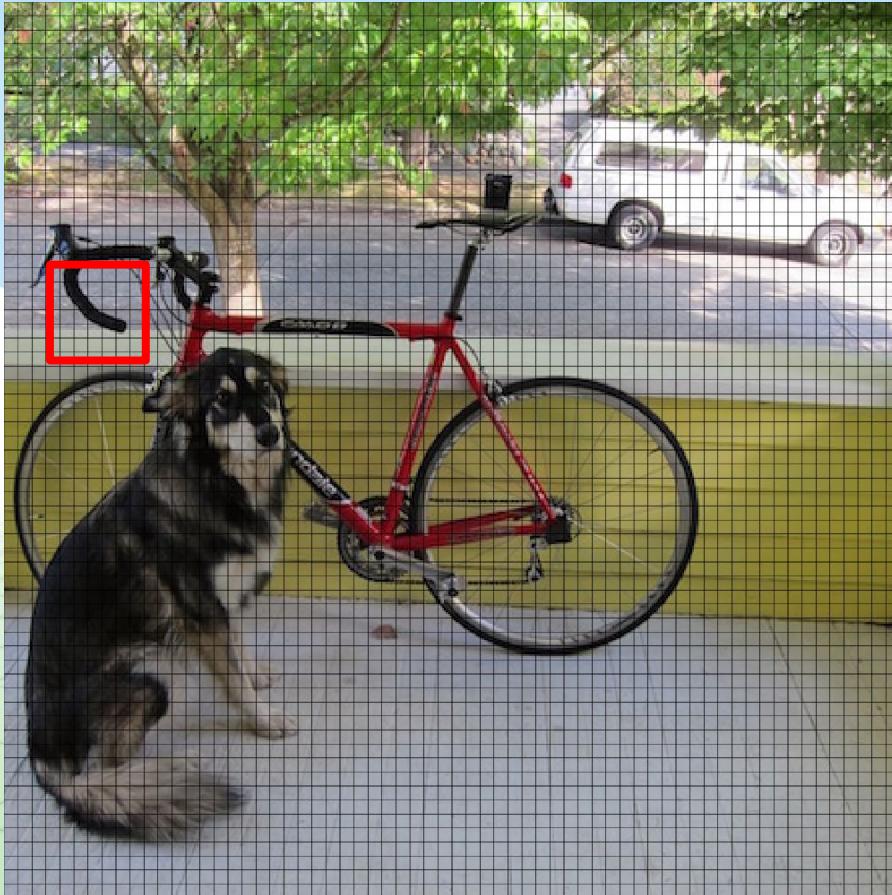
448x448 -> 64x64

---



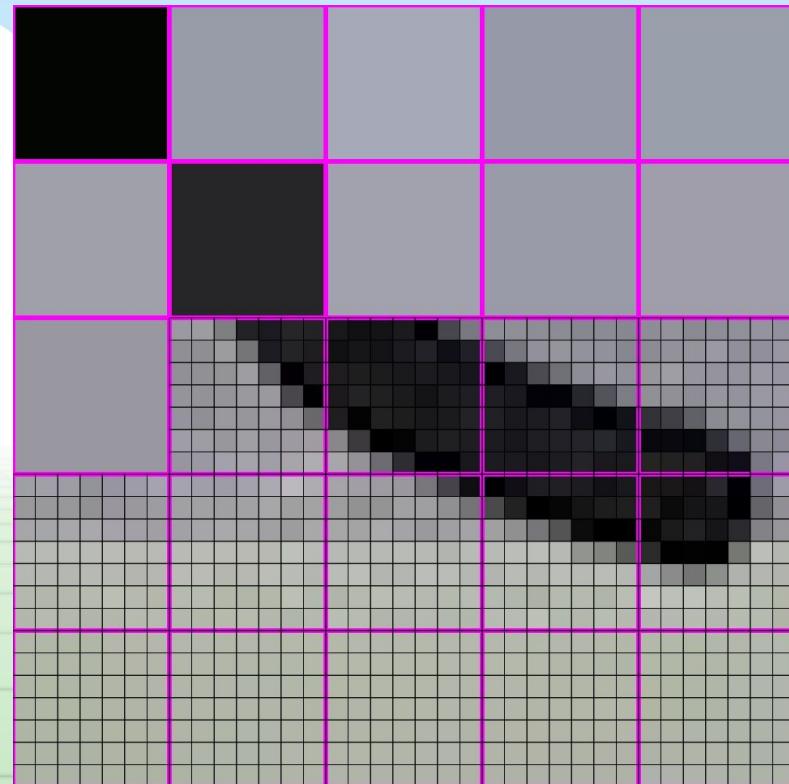
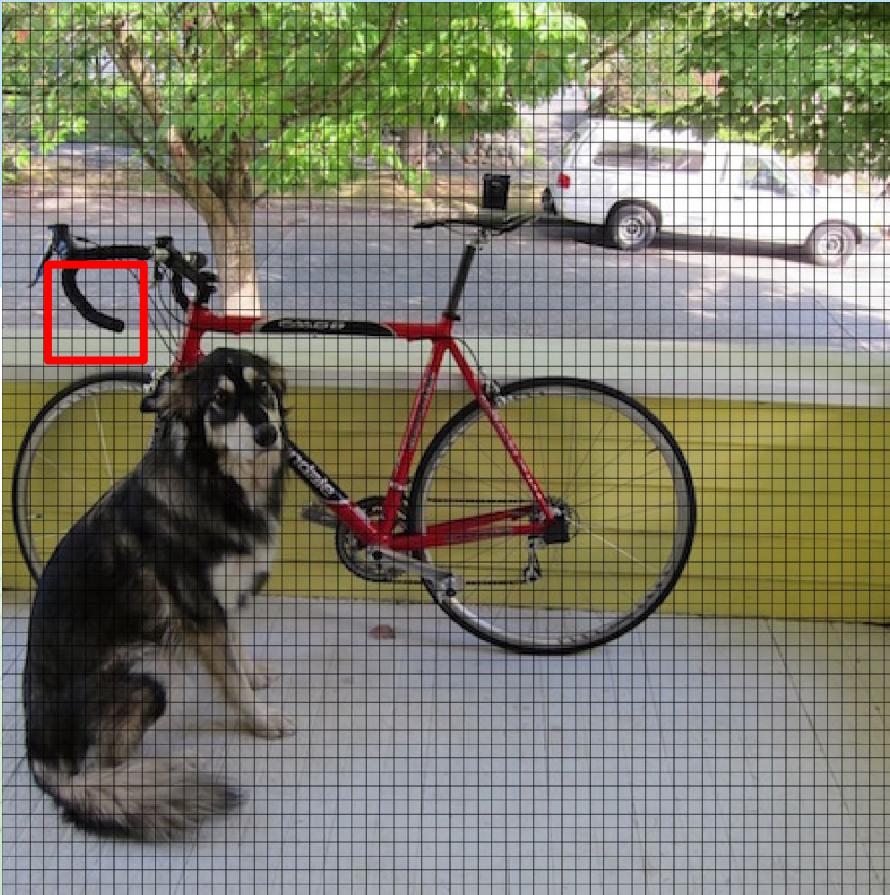
448x448 -> 64x64

---



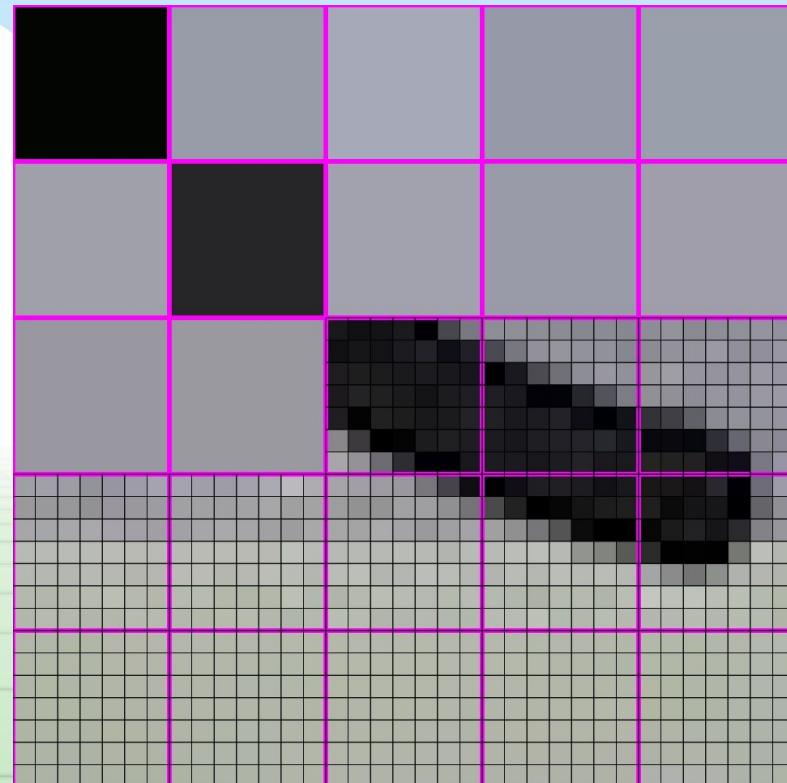
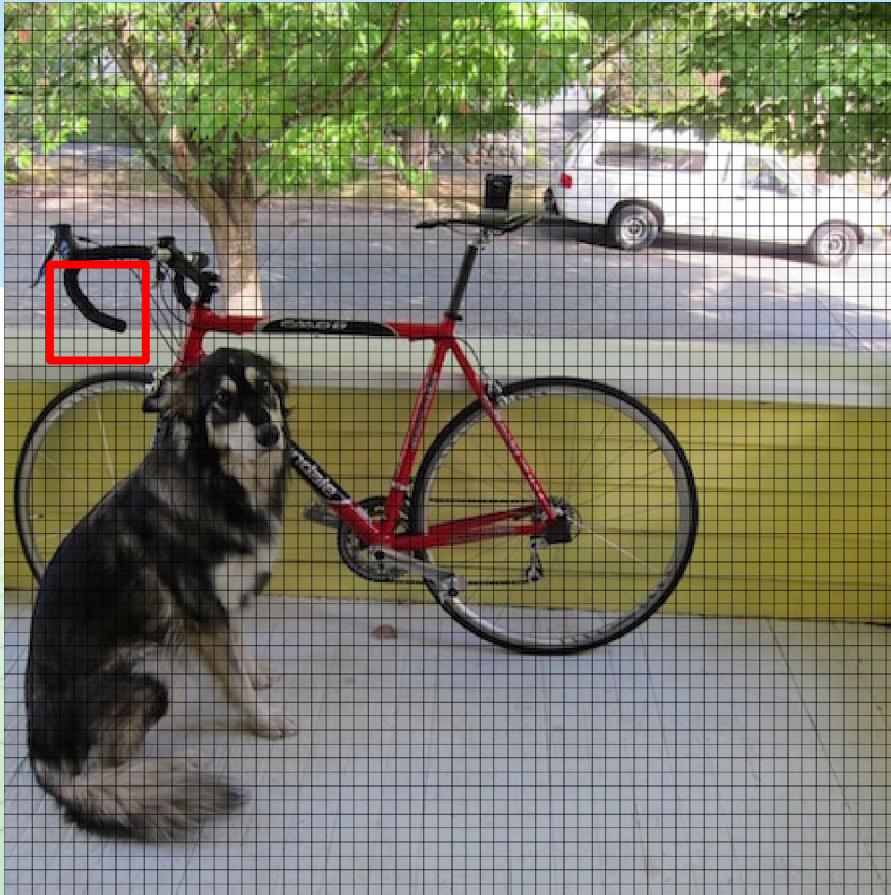
448x448 -> 64x64

---



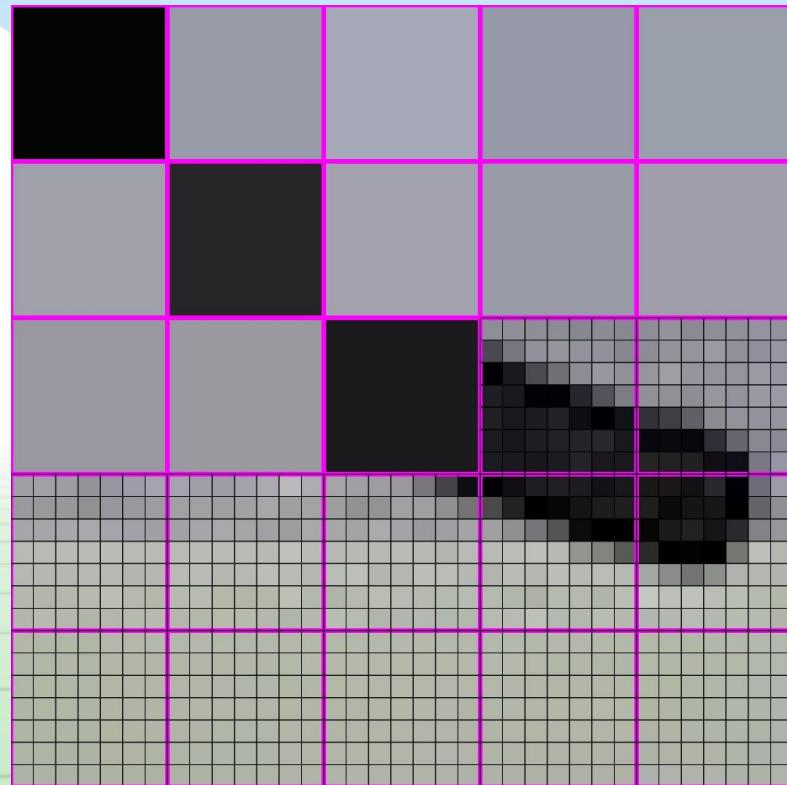
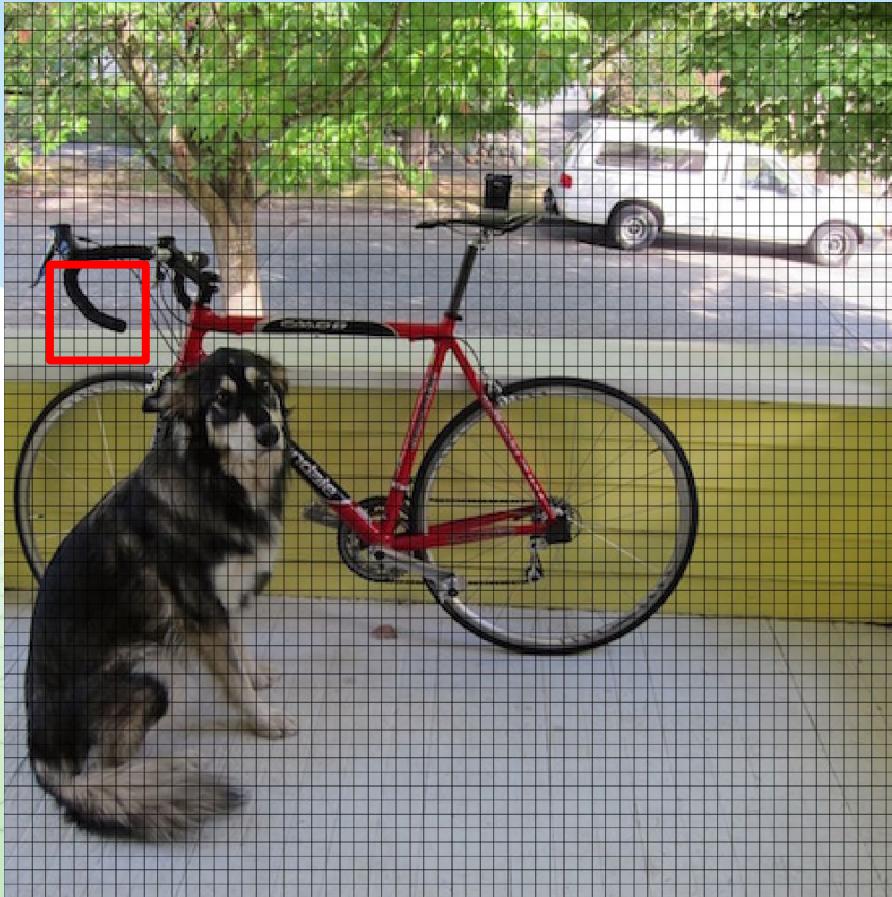
448x448 -> 64x64

---



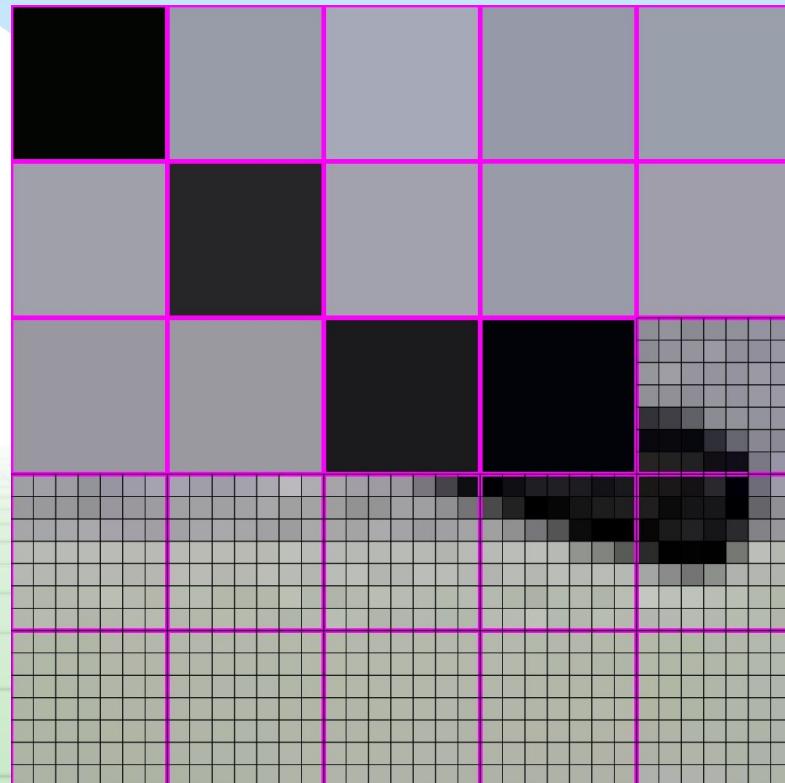
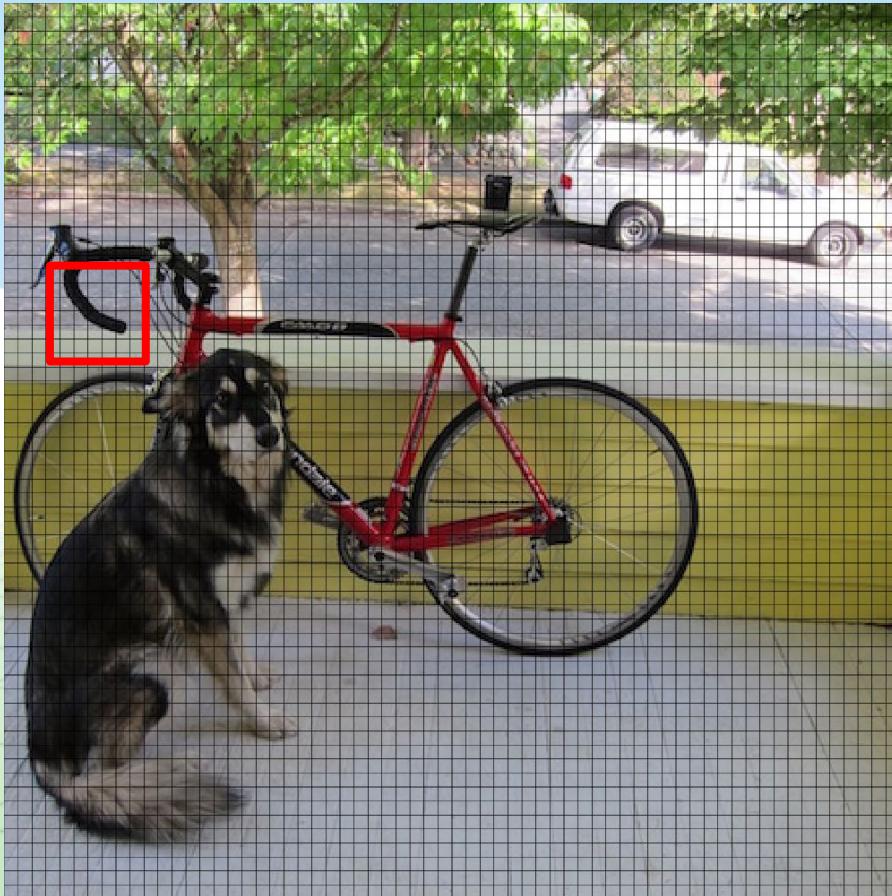
448x448 -> 64x64

---



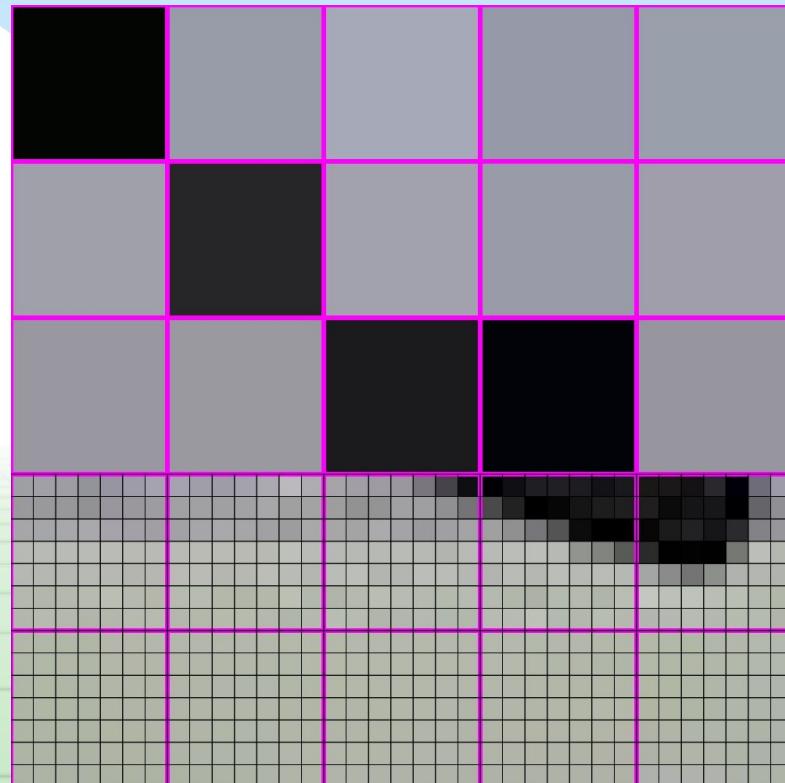
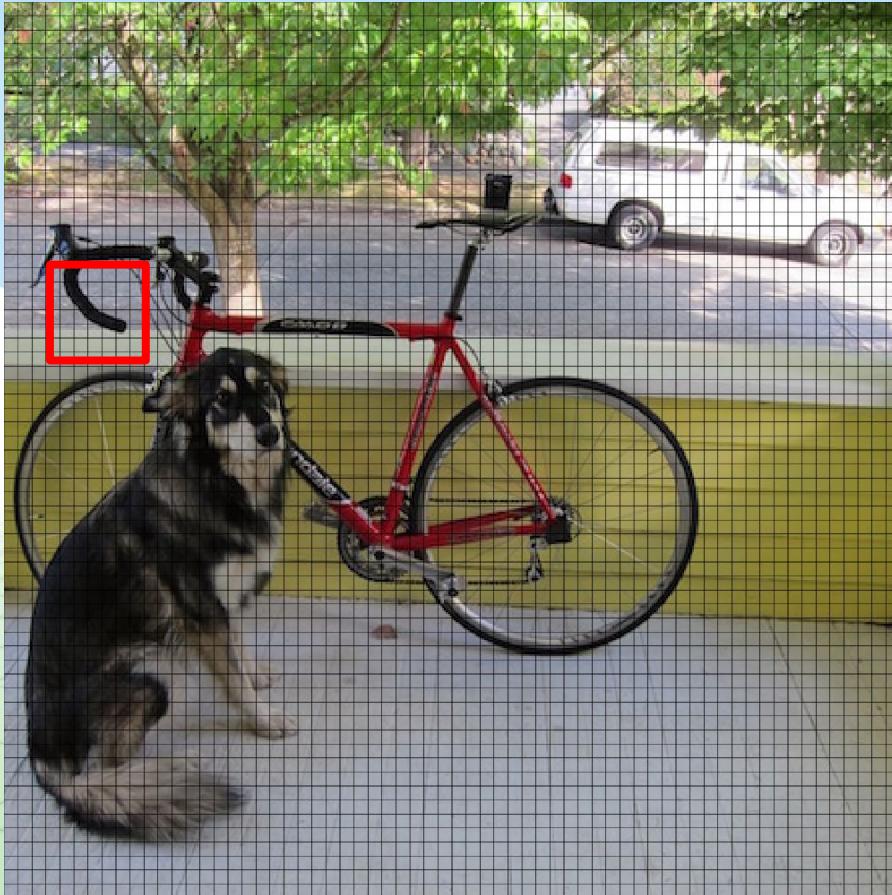
448x448 -> 64x64

---



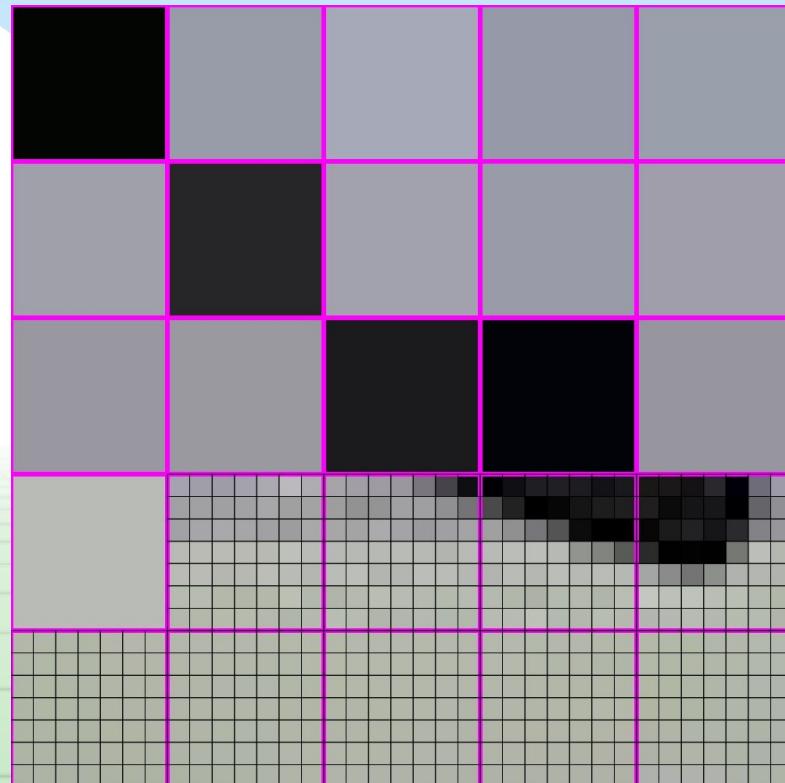
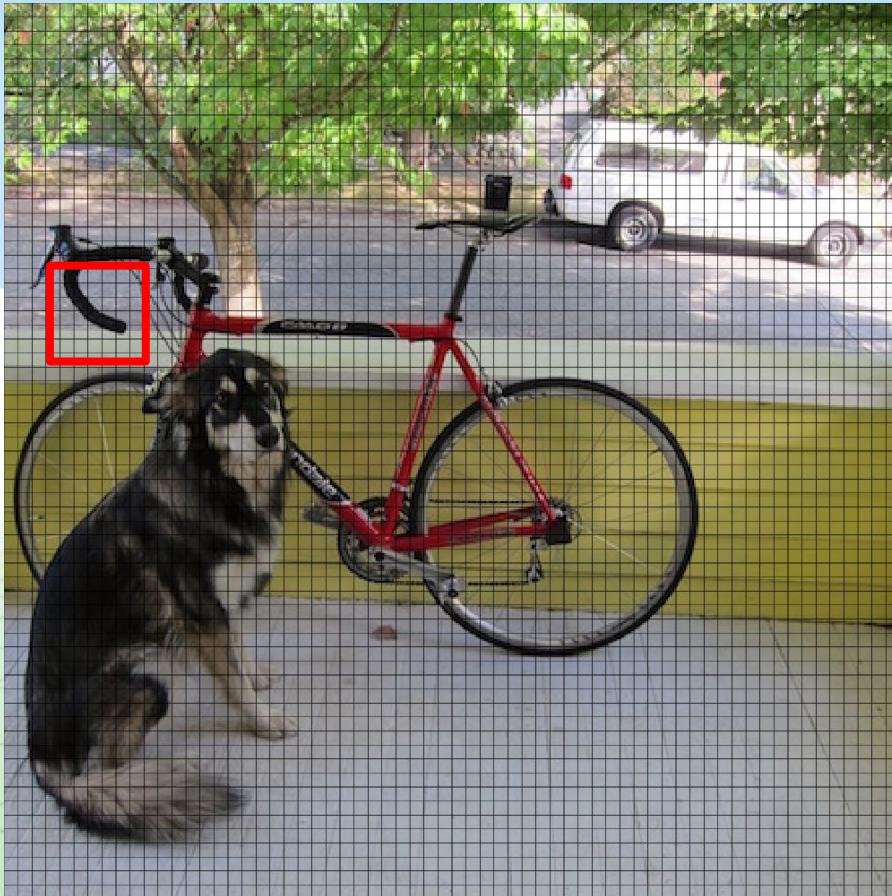
448x448 -> 64x64

---



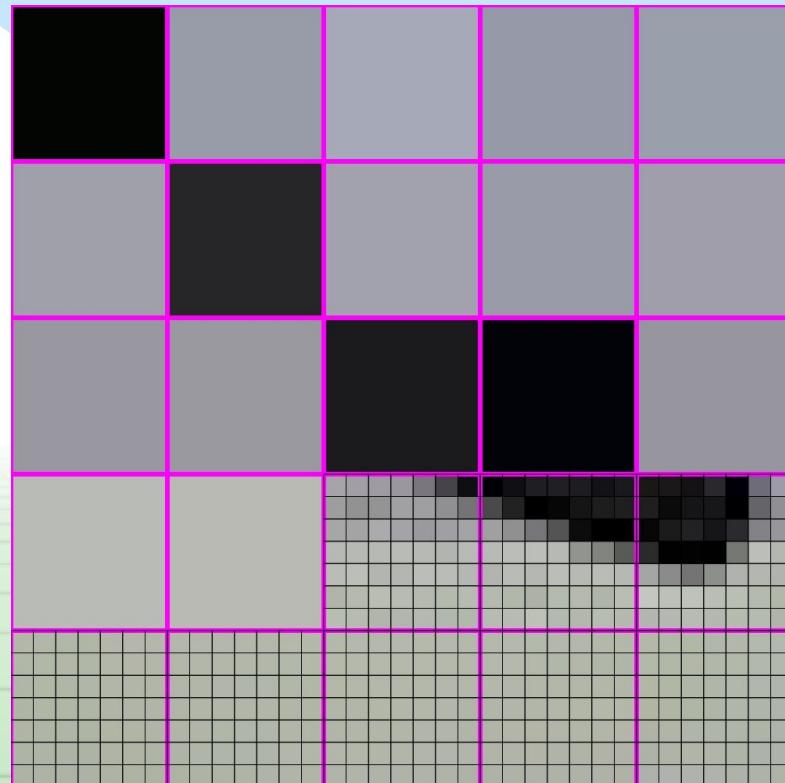
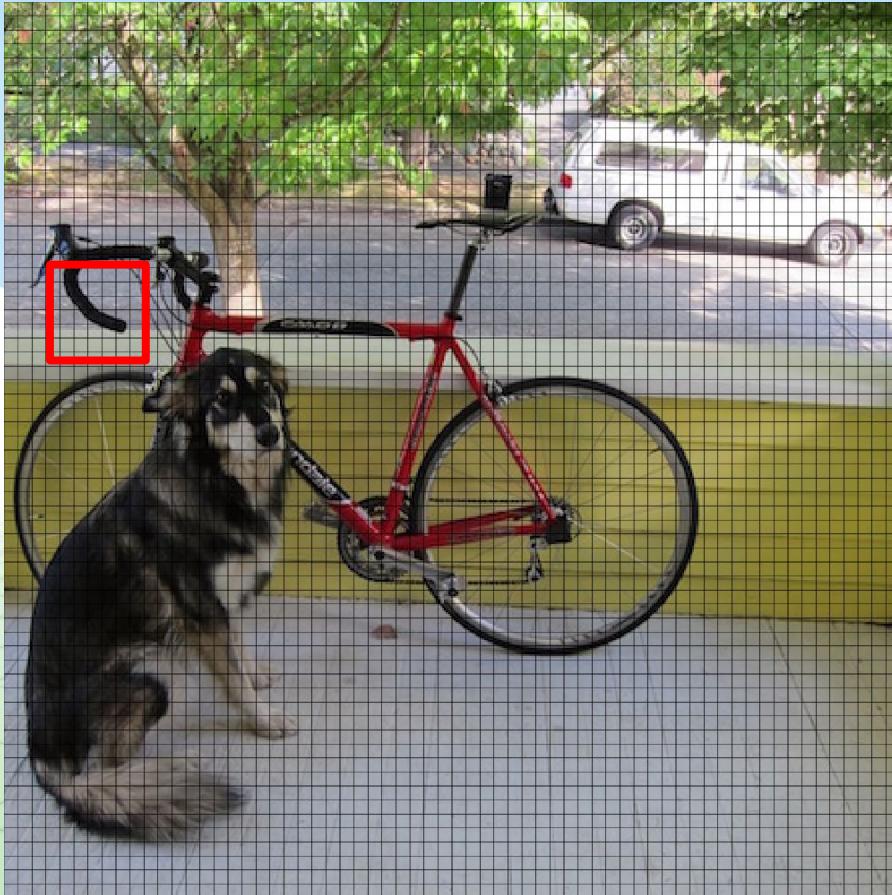
448x448 -> 64x64

---



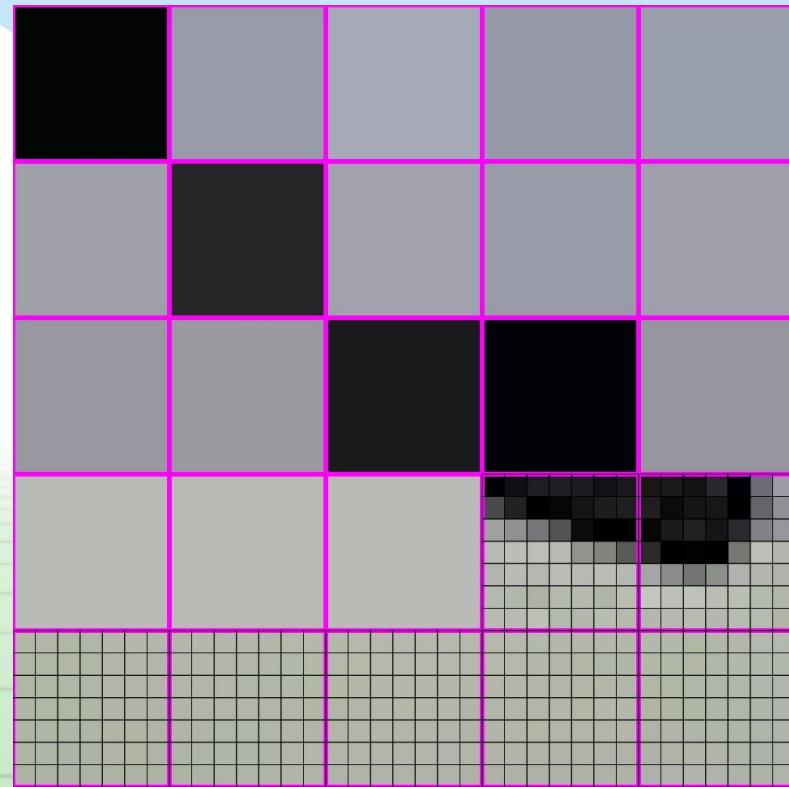
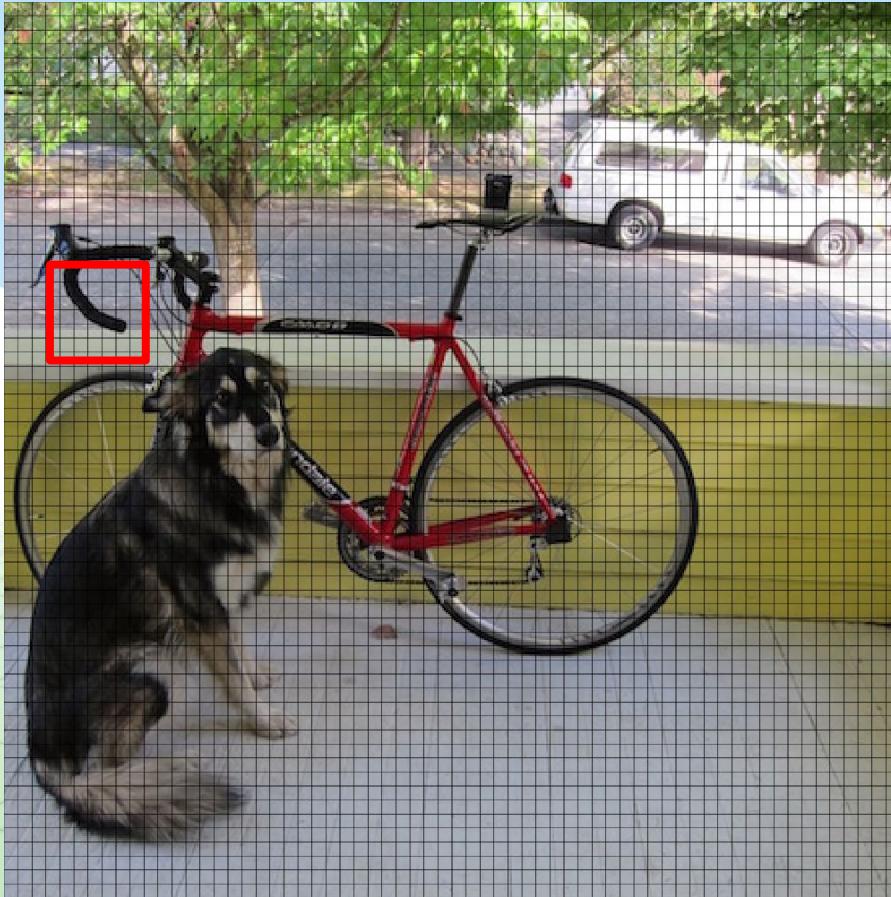
448x448 -> 64x64

---



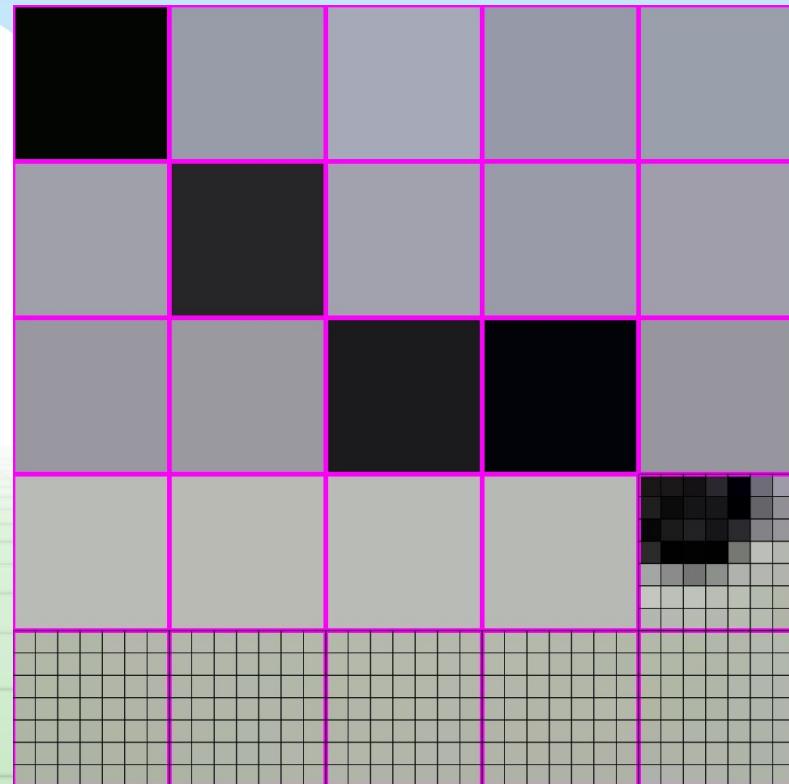
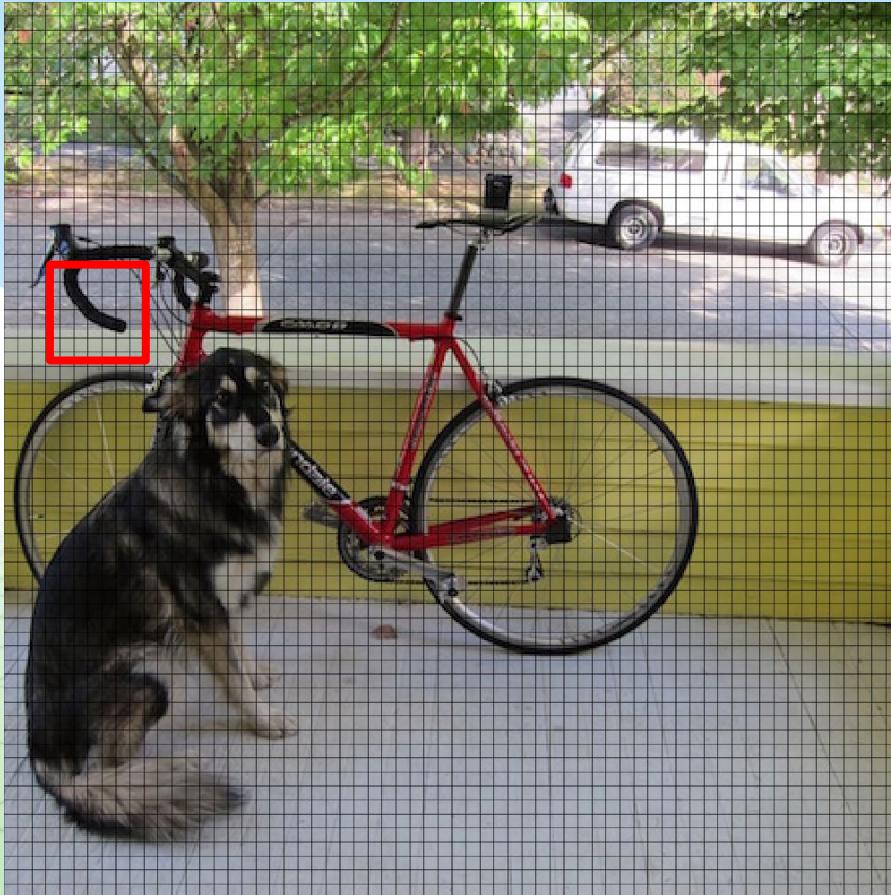
448x448 -> 64x64

---



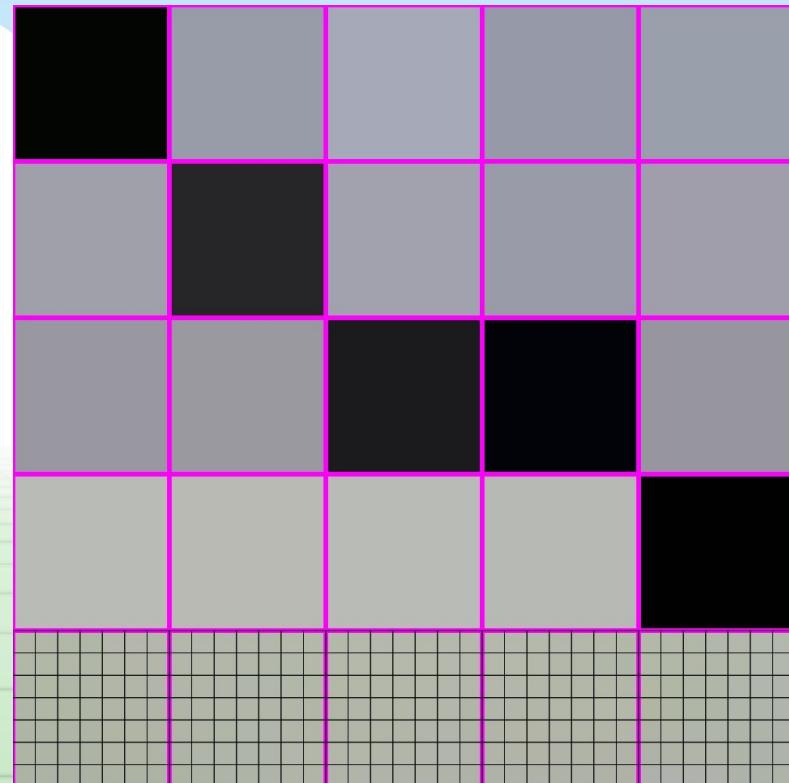
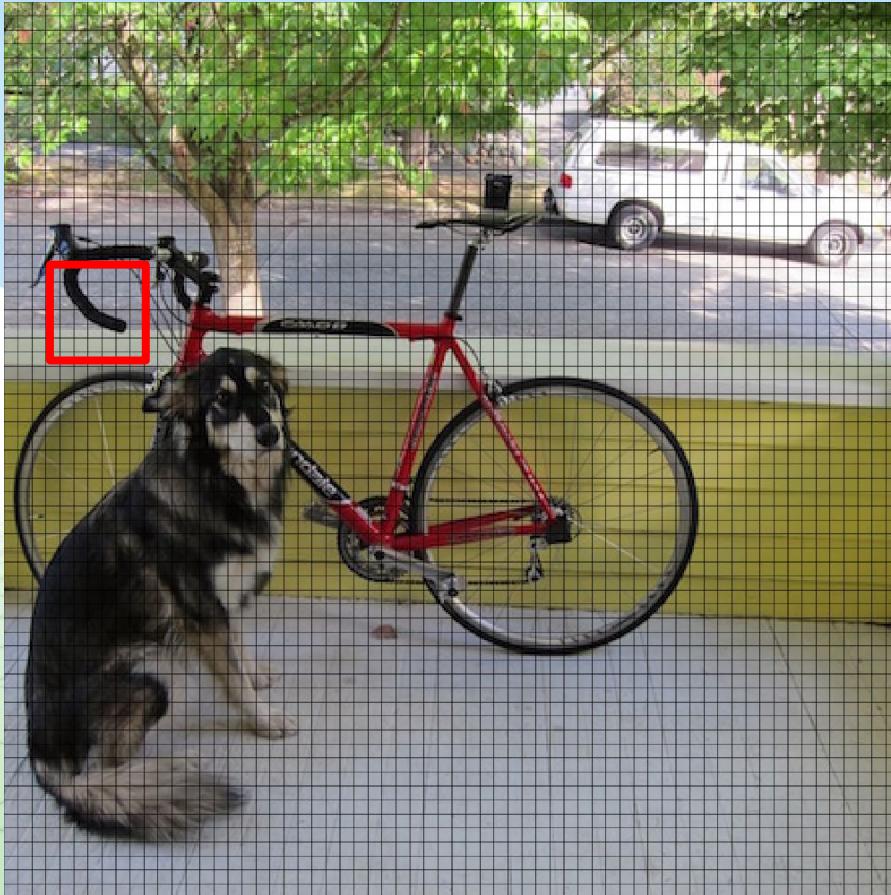
448x448 -> 64x64

---



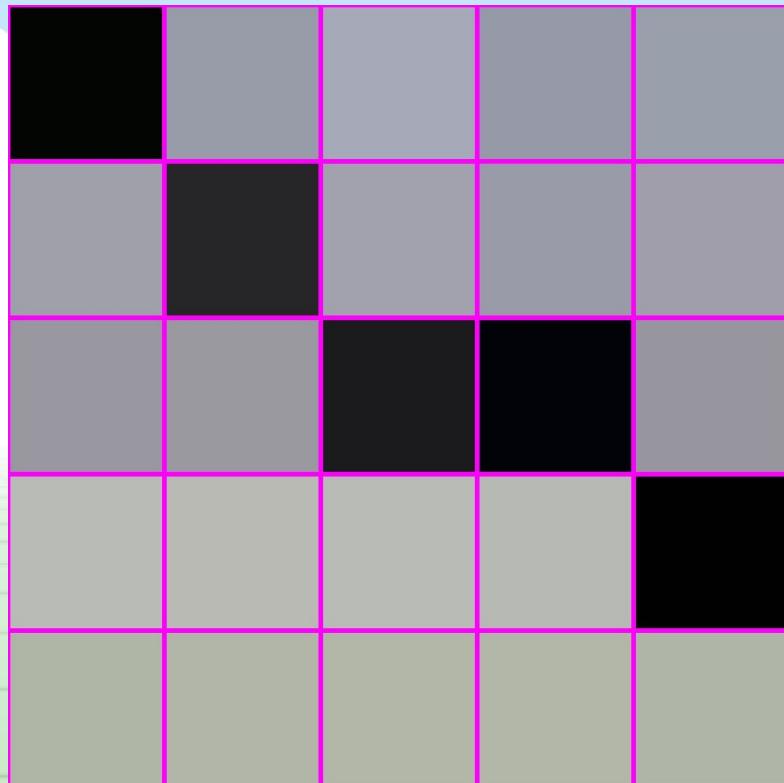
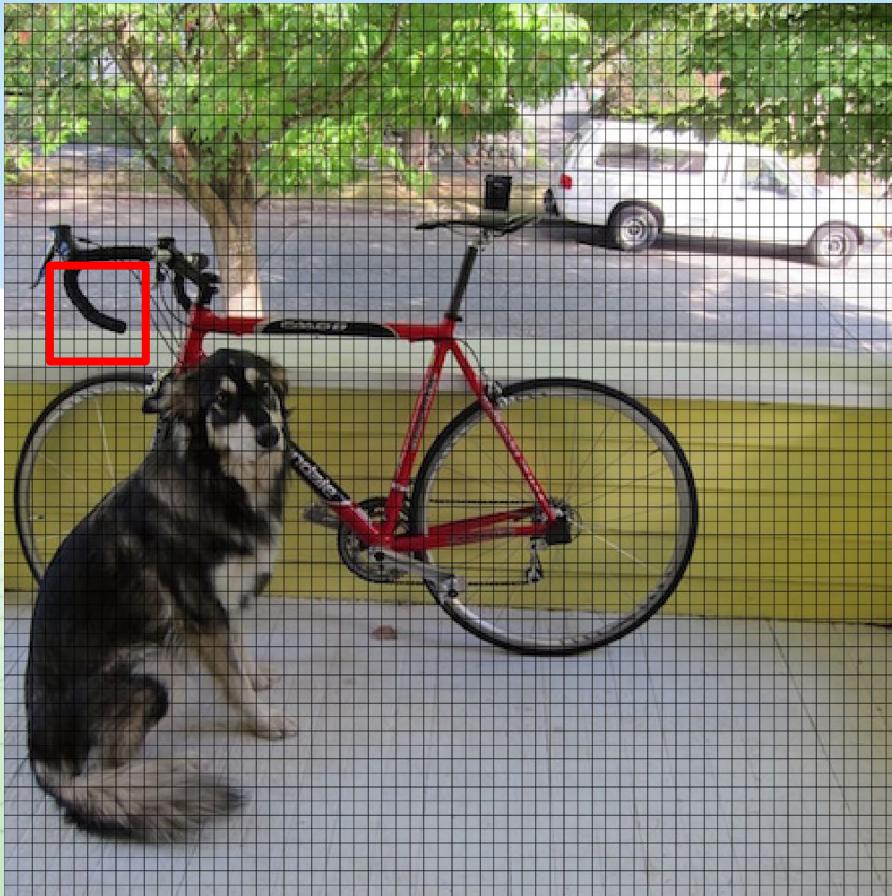
448x448 -> 64x64

---



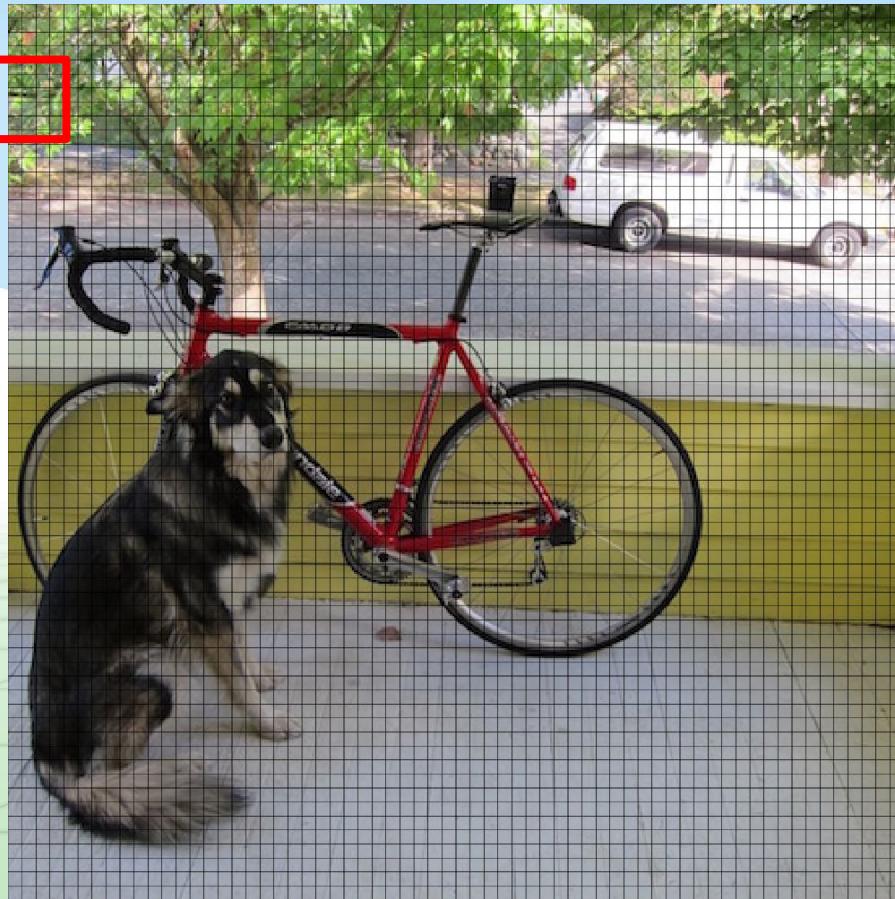
448x448 -> 64x64

---



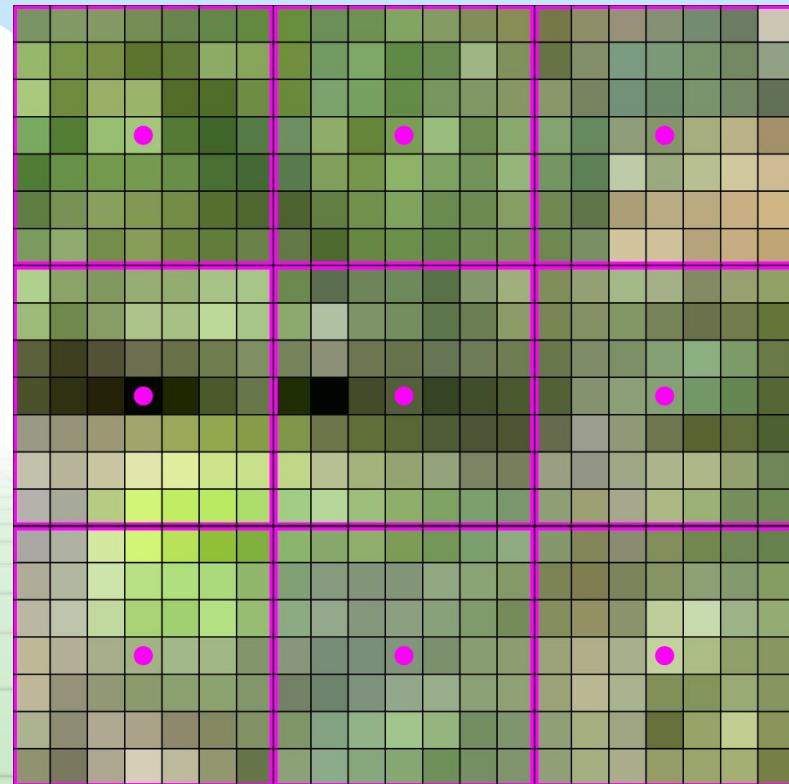
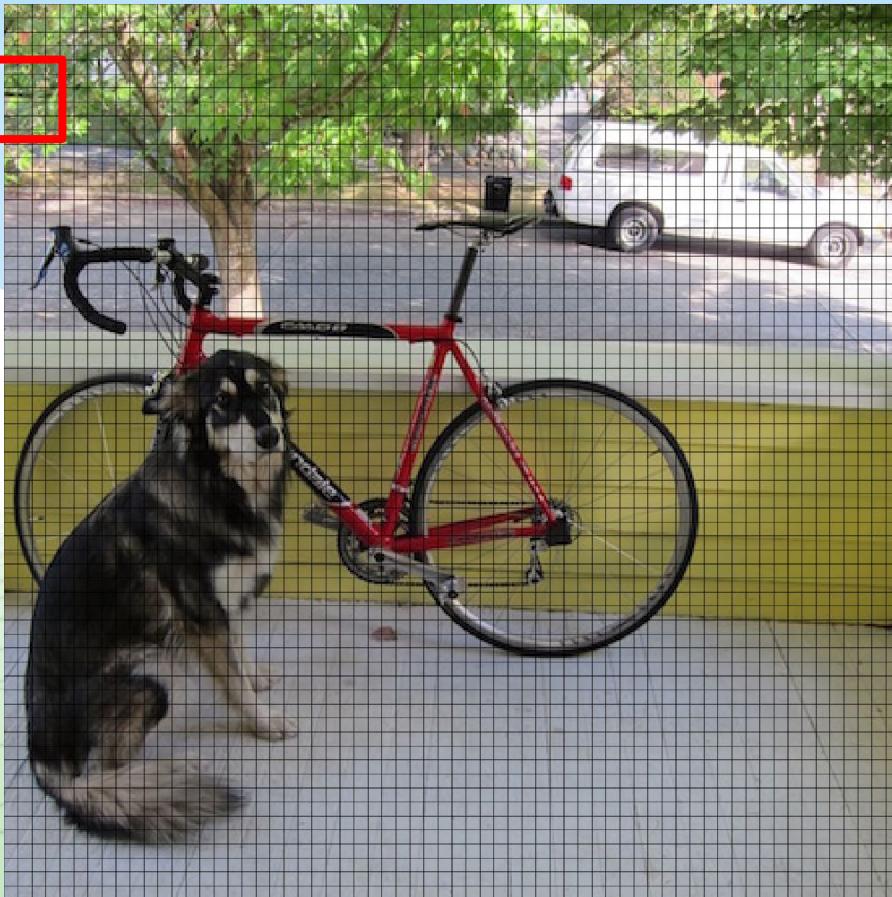
448x448 -> 64x64

---



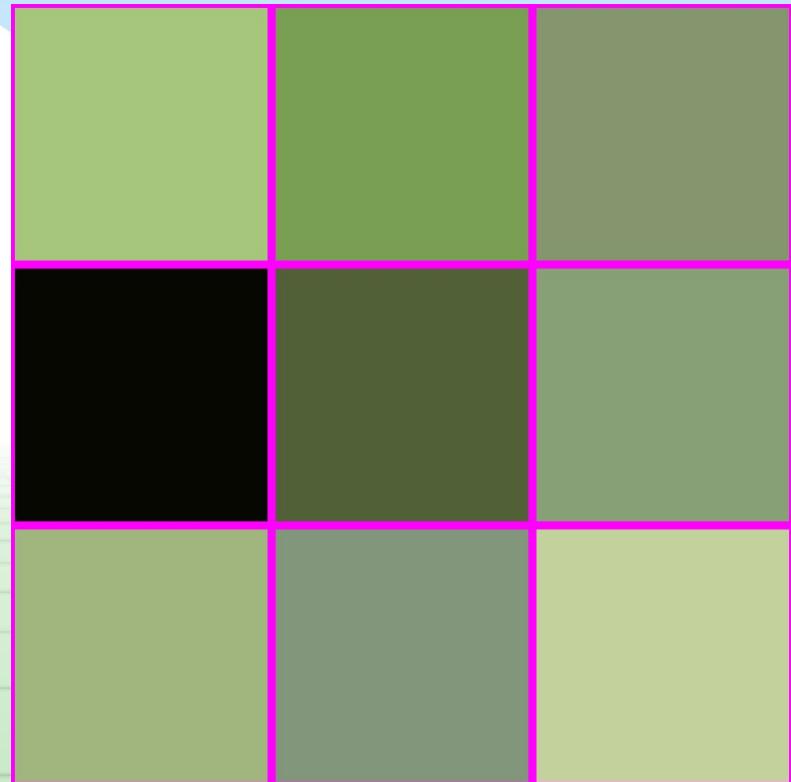
448x448 -> 64x64

---



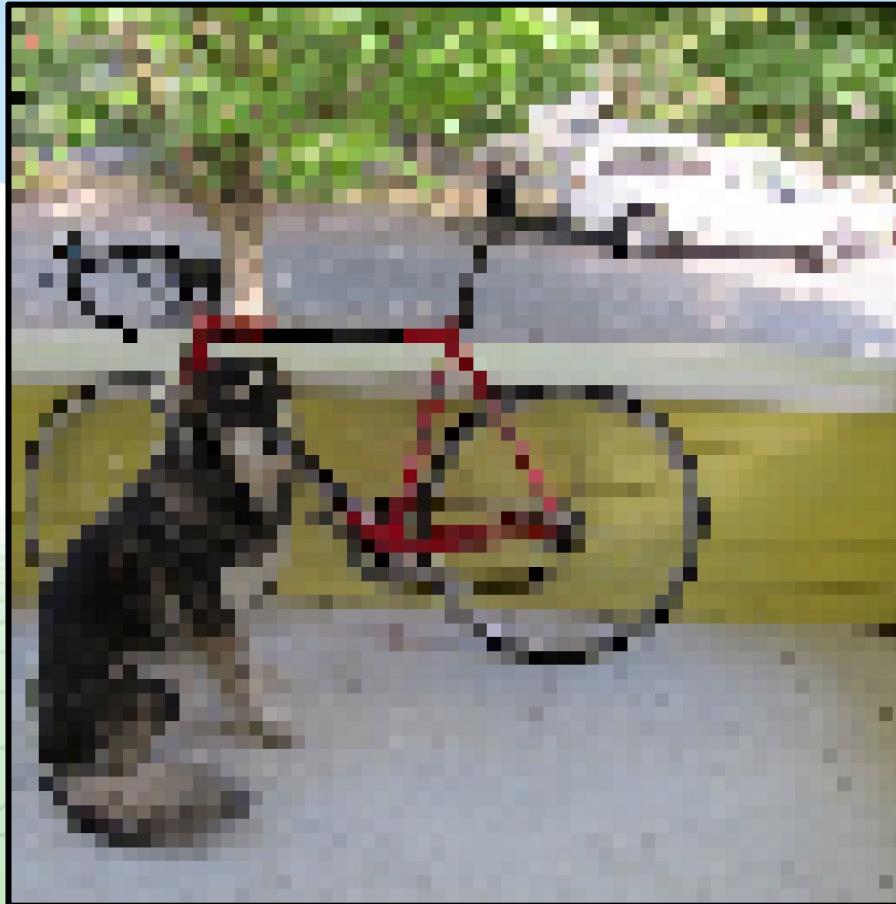
448x448 -> 64x64

---



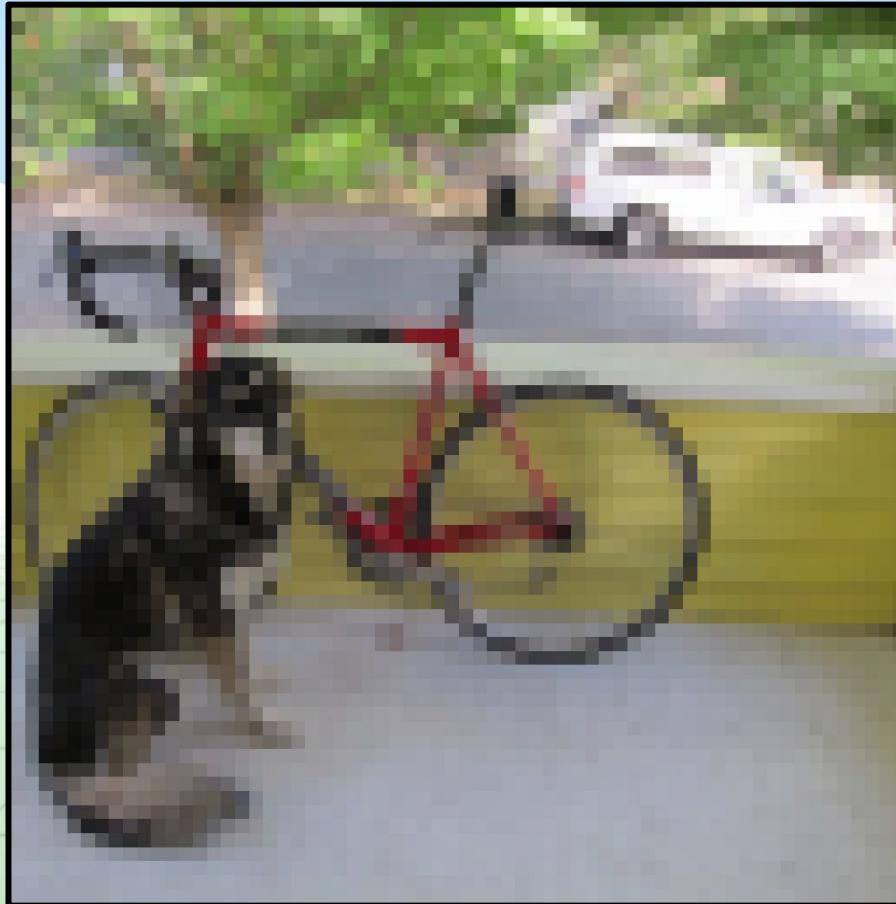
# IS THIS ALL THERE IS??

---



# THERE IS A BETTER WAY!

---

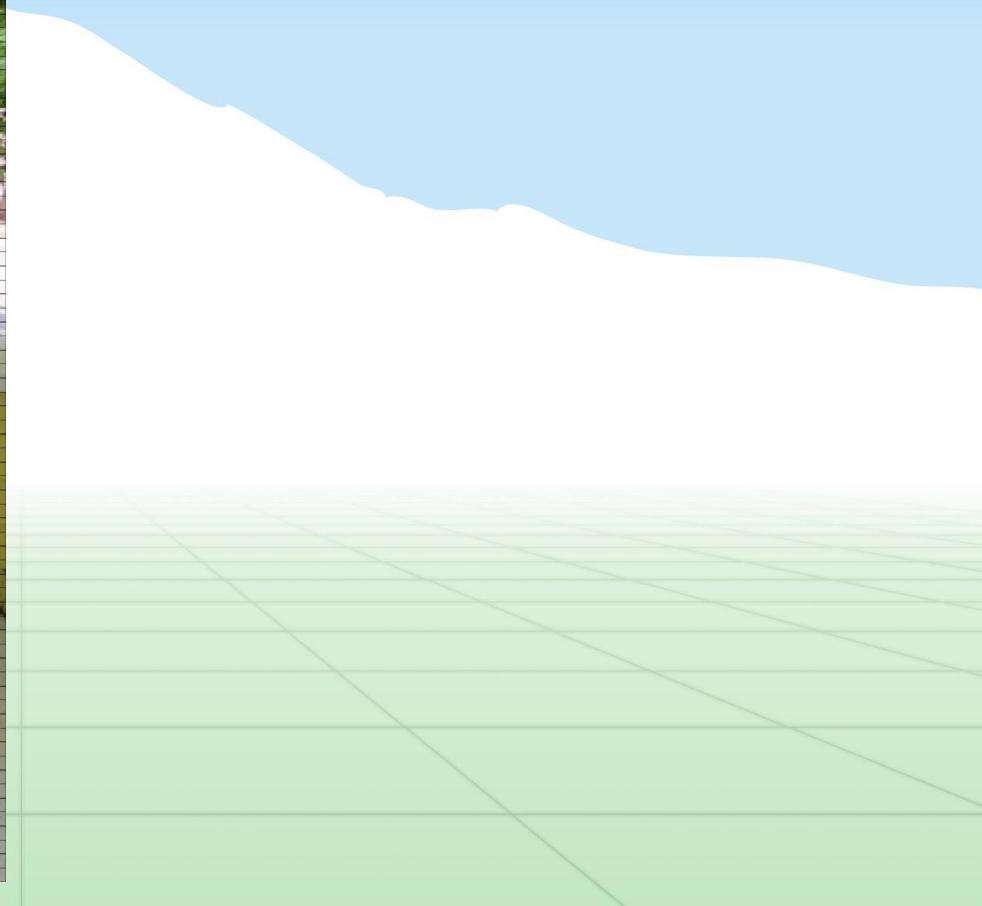
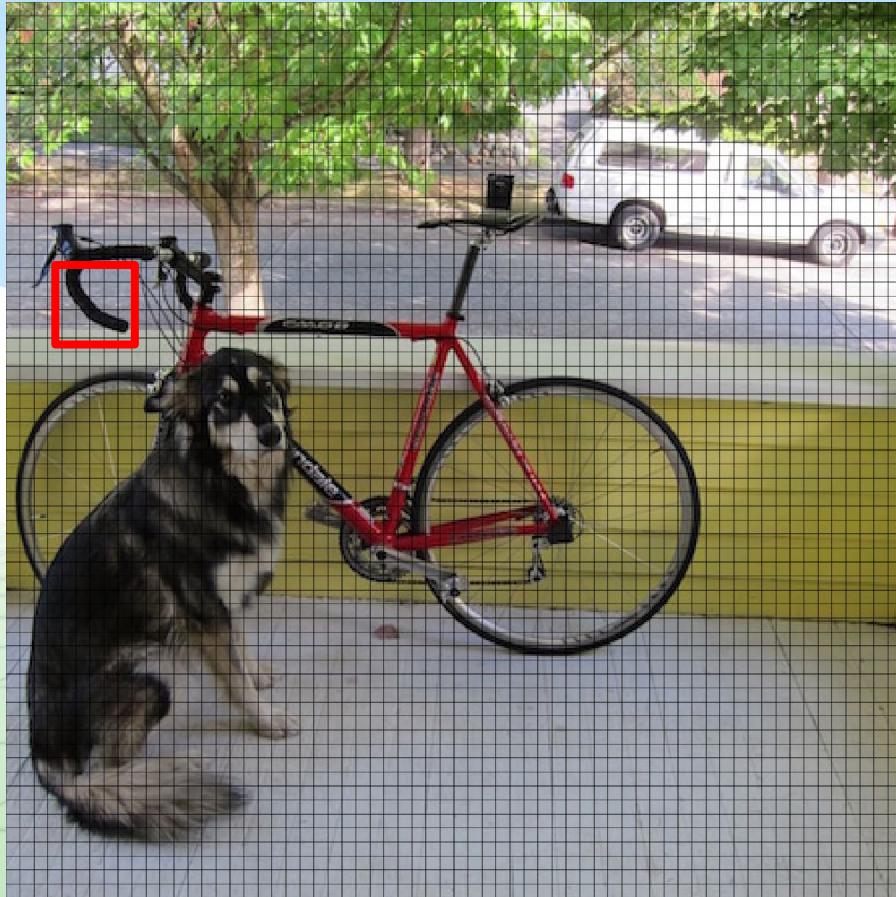


# LOOK AT HOW MUCH BETTER



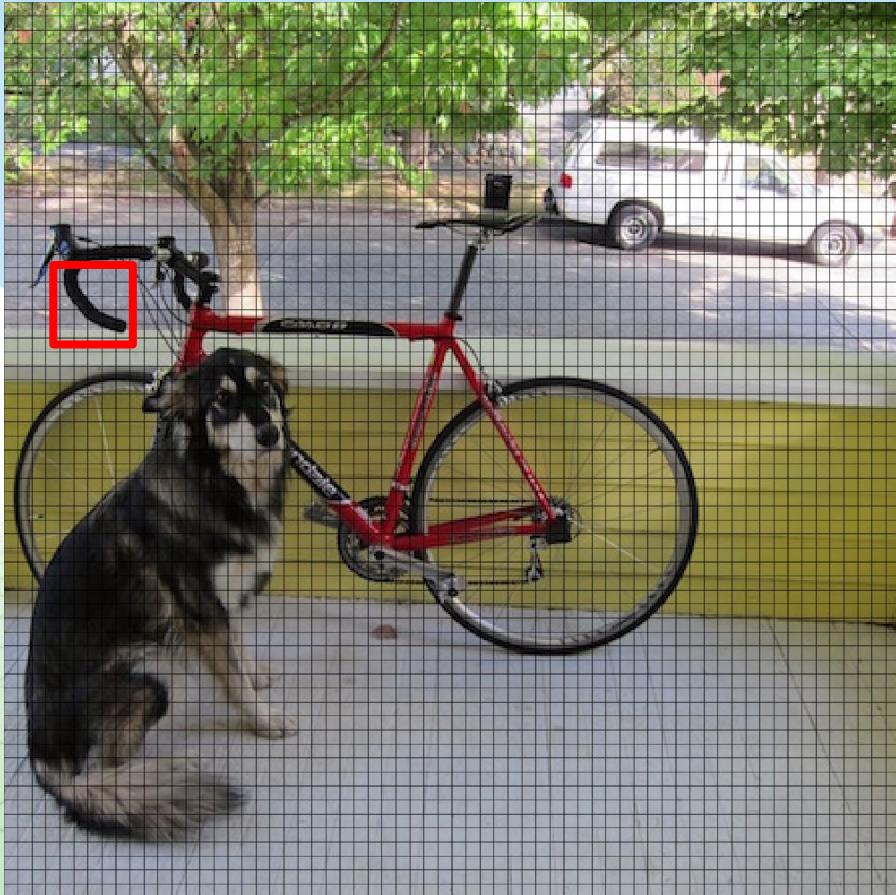
# How do?

---



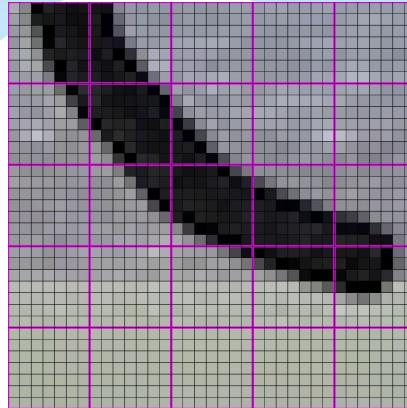
# How do? Averaging!

---

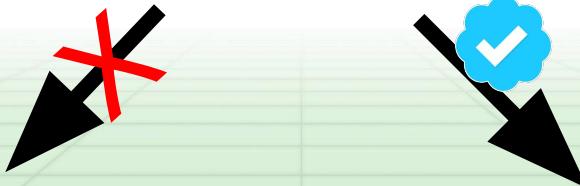
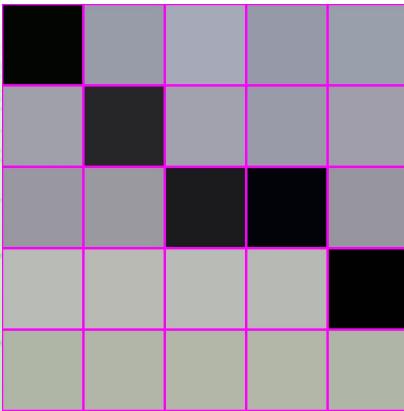


# How do? Averaging!

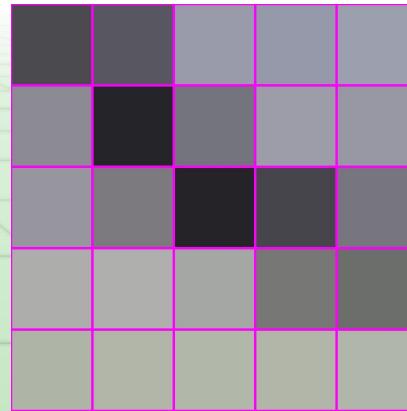
---



“interpolation”

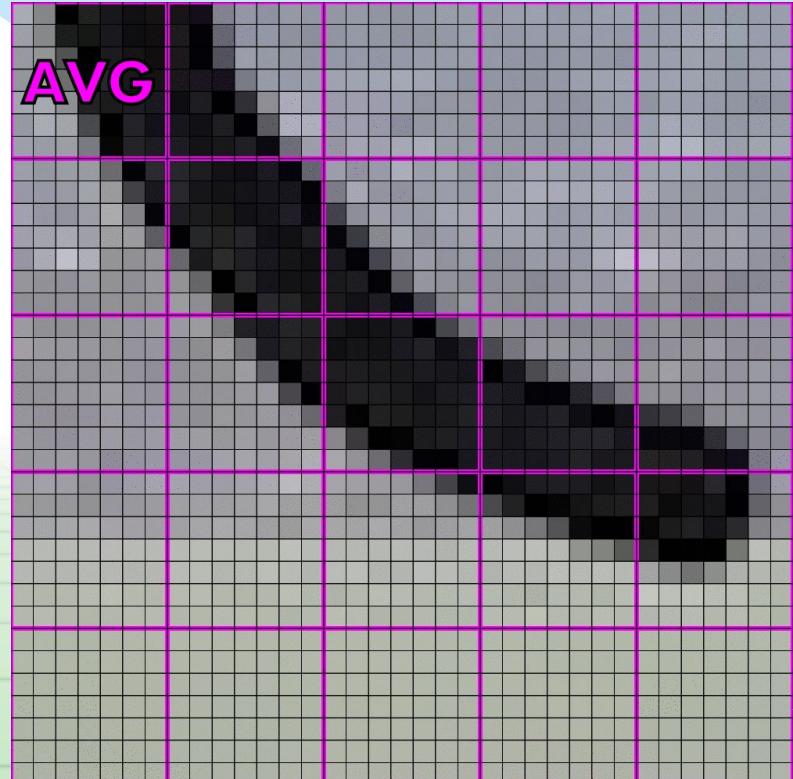


averaging



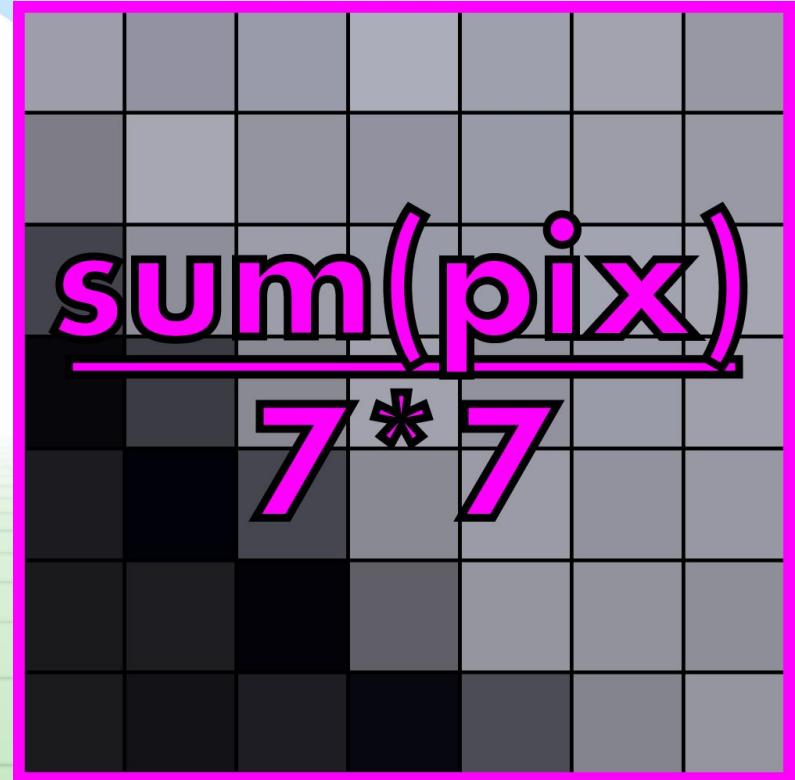
# What is averaging?

---



# What is averaging? A weighted sum

---



# What is averaging? A weighted sum

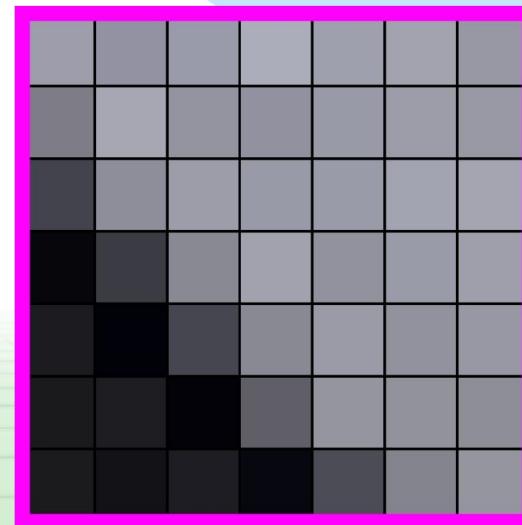
**sum** [  $\frac{1}{49}$  ]

# Call this operation “convolution”

*Filter or kernel*

$\frac{1}{49}$

$1 \times$							
$1 \times$							
$1 \times$							
$1 \times$							
$1 \times$							
$1 \times$							
$1 \times$							
$1 \times$							



# Convolutions on larger images

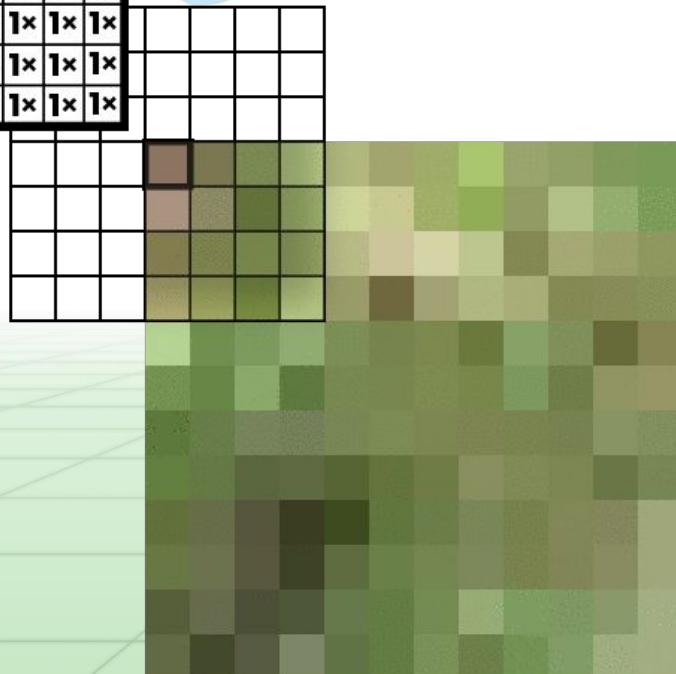
$\frac{1}{49}$

$1 \times$							
$1 \times$							
$1 \times$							
$1 \times$							
$1 \times$							
$1 \times$							
$1 \times$							
$1 \times$							



# Kernel slides across image

1  
49

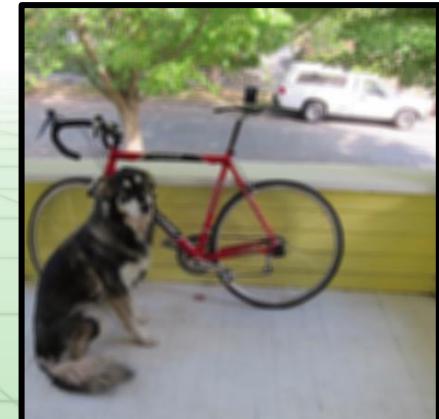
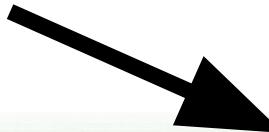


A 10x10 grid of squares. The top-left square is filled black, while all other squares are white.

# Convolutions on larger images

$\frac{1}{49}$

1x							
1x							
1x							
1x							
1x							
1x							
1x							
1x							



# This is called box filter

$\frac{1}{49}$

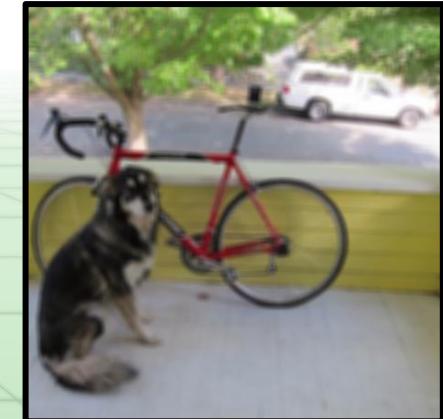
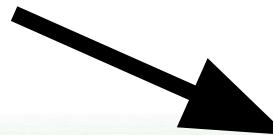
1x							
1x							
1x							
1x							
1x							
1x							
1x							
1x							



Box filters

$\frac{1}{N \times M}$

1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
			...		M	
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x



# Box filters smooth image

$\frac{1}{49}$

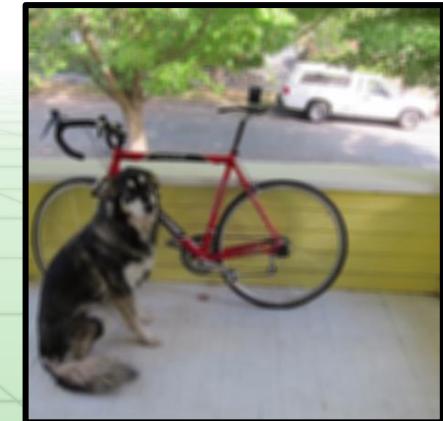
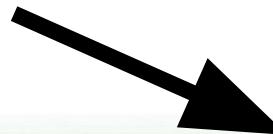
1x						
1x						
1x						
1x						
1x						
1x						
1x						



Box filters

$\frac{1}{N \times M}$

1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
			...		M	
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x



# Box filters smooth image

$\frac{1}{49}$

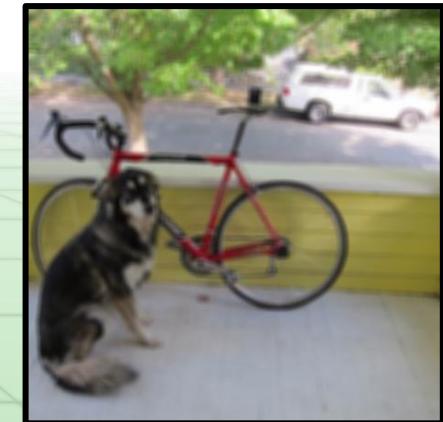
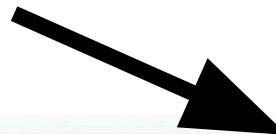
1x						
1x						
1x						
1x						
1x						
1x						
1x						



Box filters

$\frac{1}{N \times M}$

1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
			...		M	
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x
1x	1x	1x	1x	1x		1x



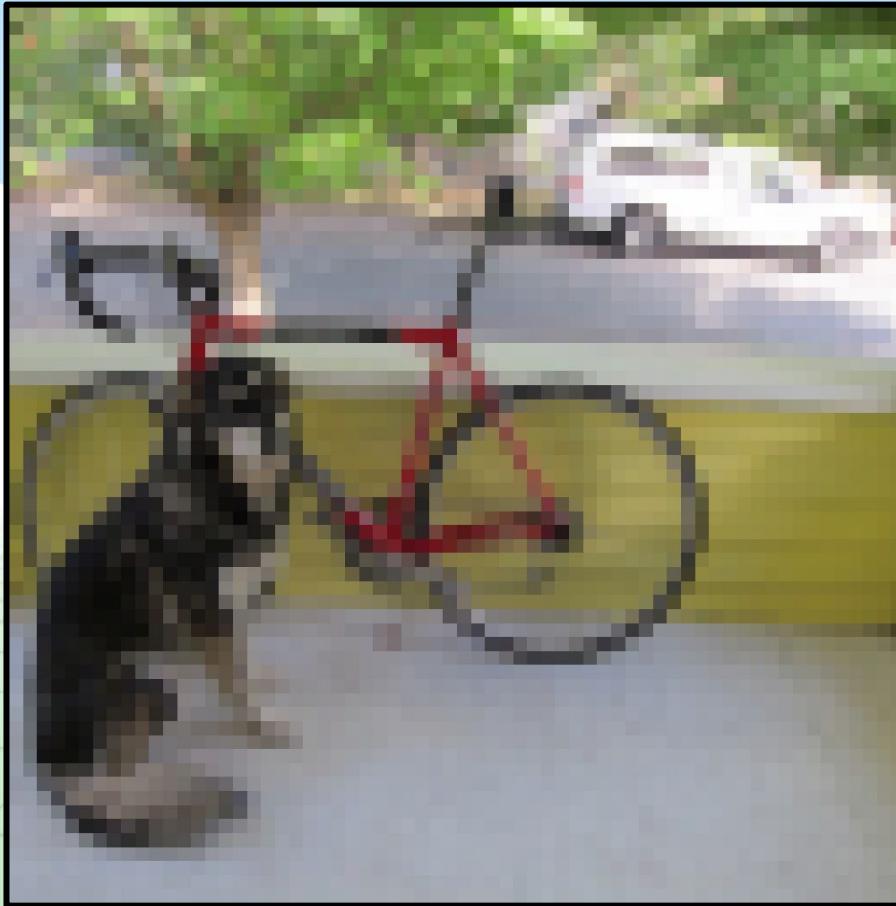
# Now we resize our smoothed image

---



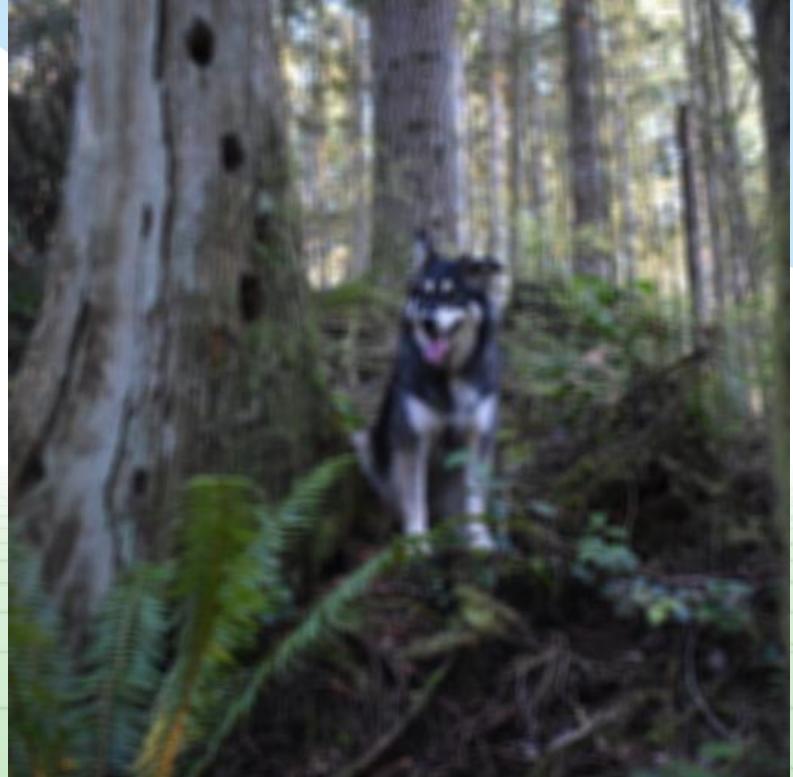
# So much better!

---

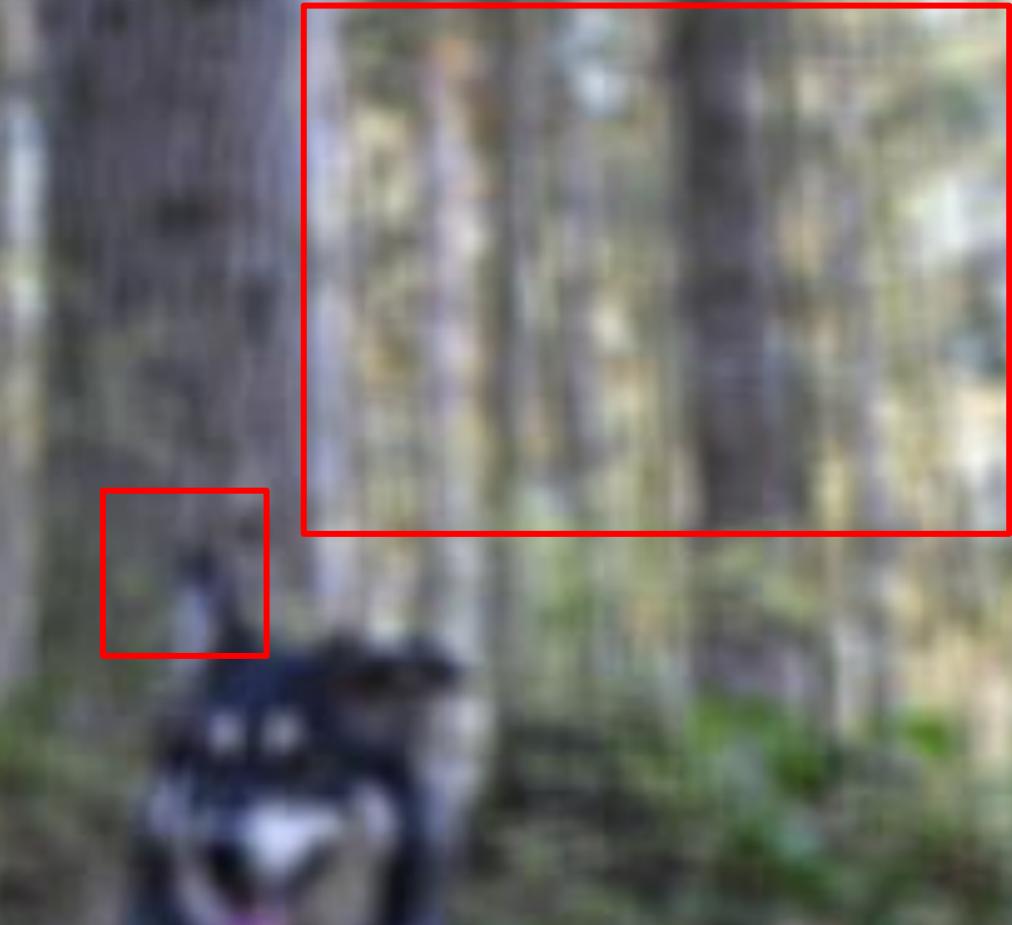


# Box filters have artifacts

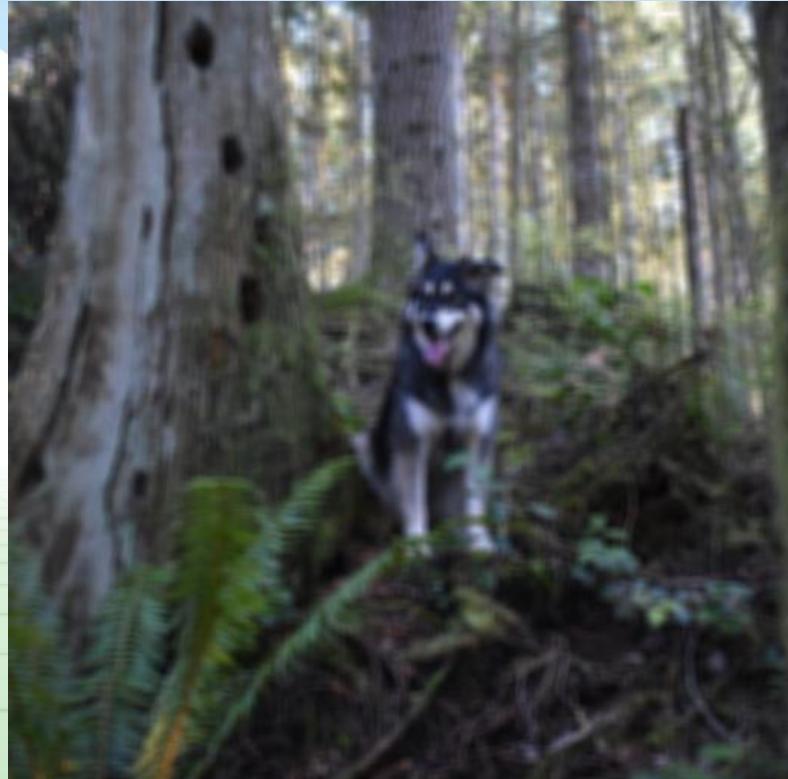
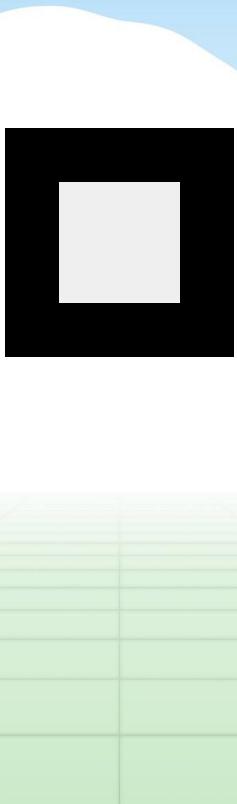
---



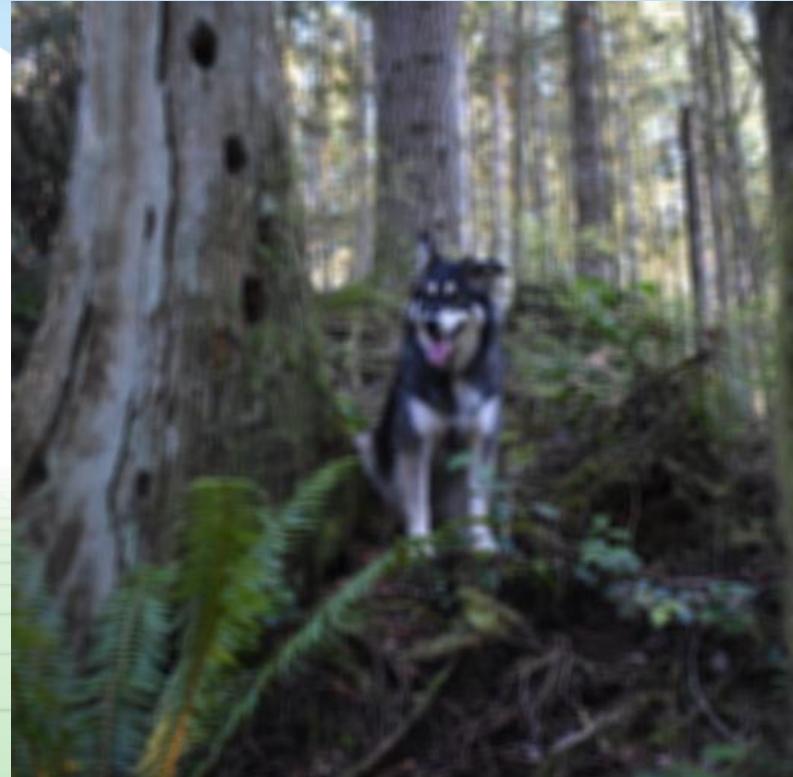
# Box filters have artifacts



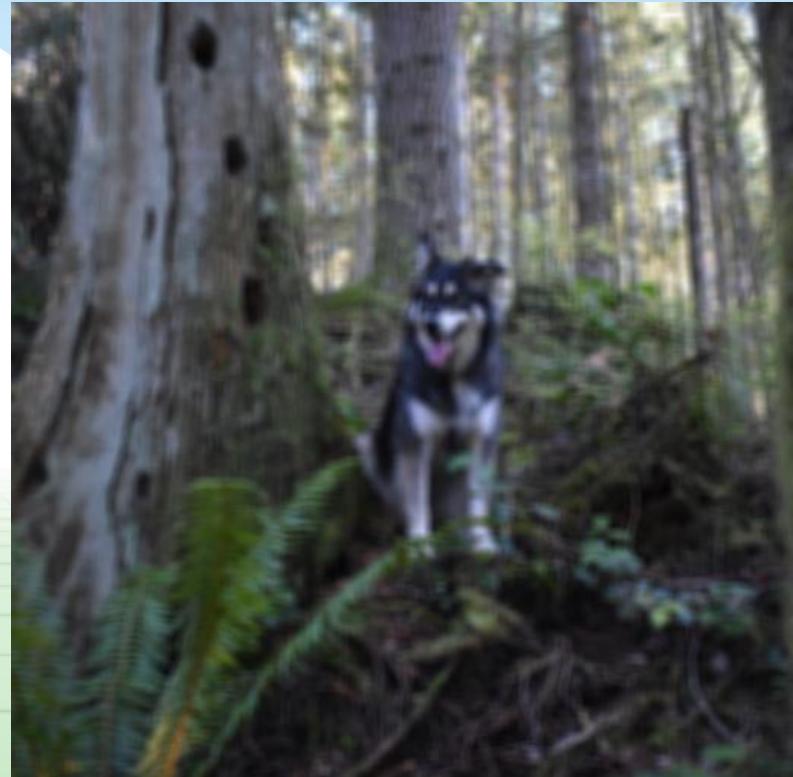
## Box filters: vertical + horizontal streaking



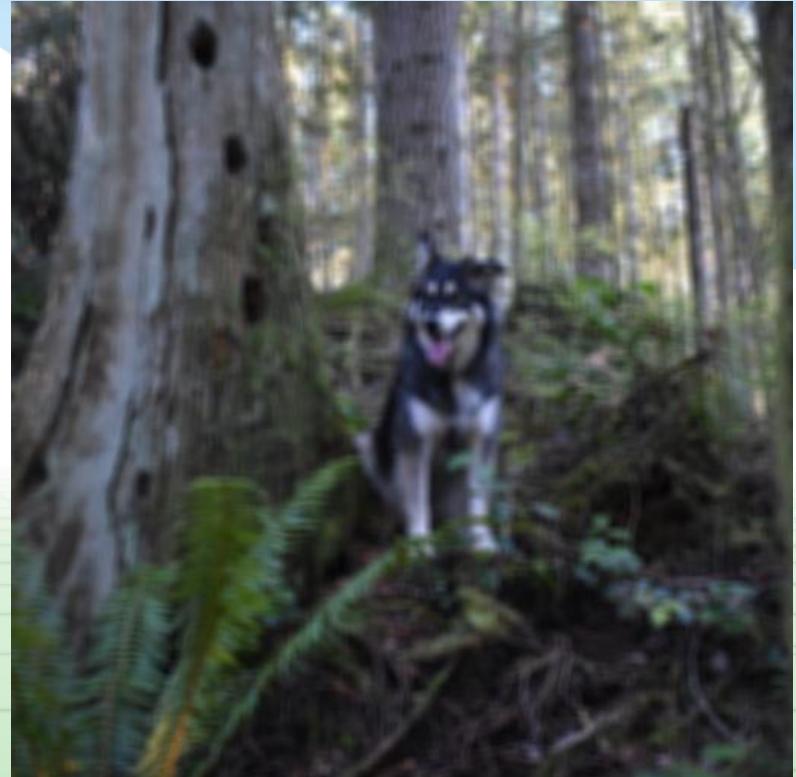
## Box filters: vertical + horizontal streaking



## Box filters: vertical + horizontal streaking

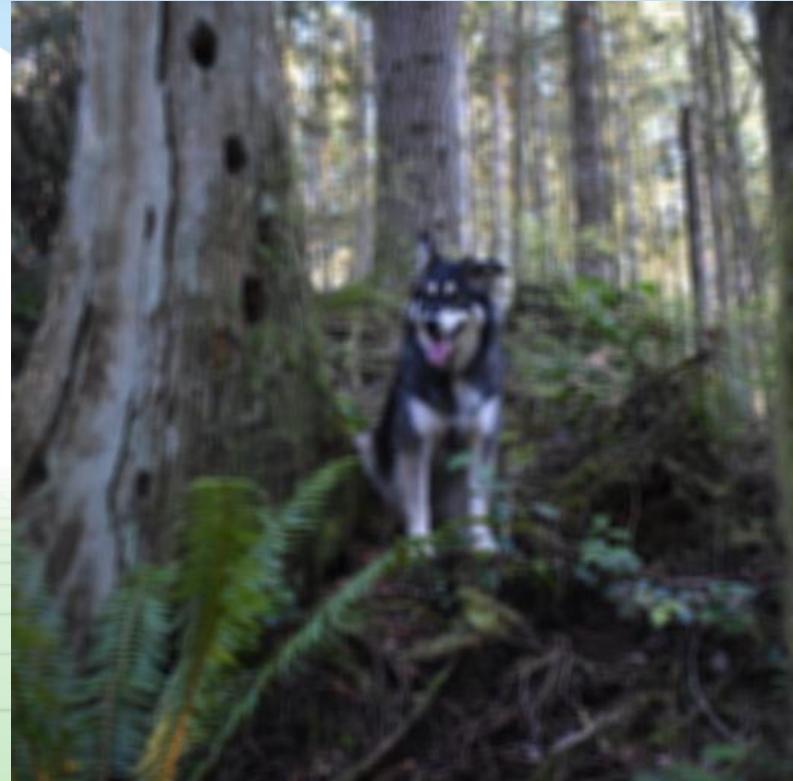
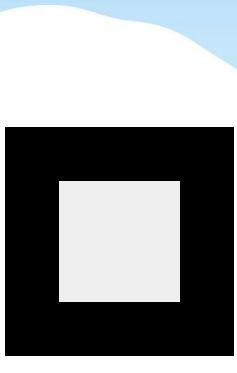


## Box filters: vertical + horizontal streaking



# We want a smoothly weighted kernel

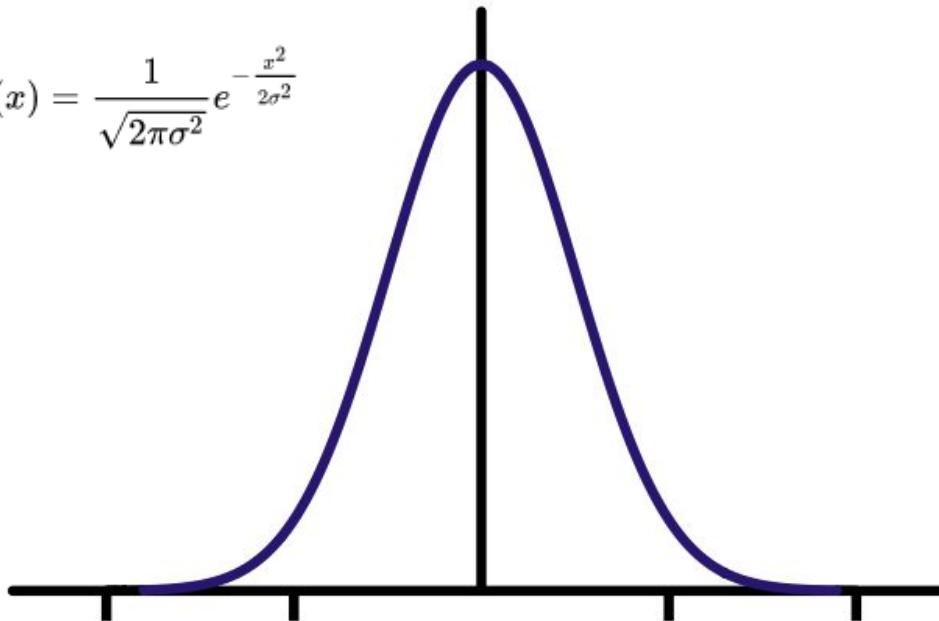
---



# Gaussians

---

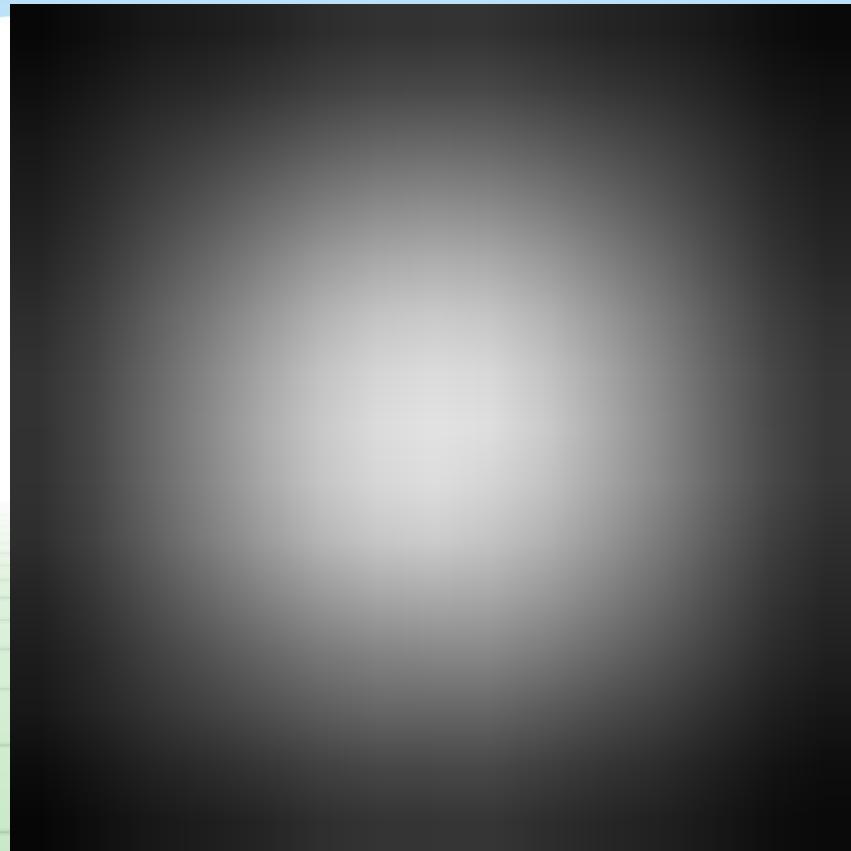
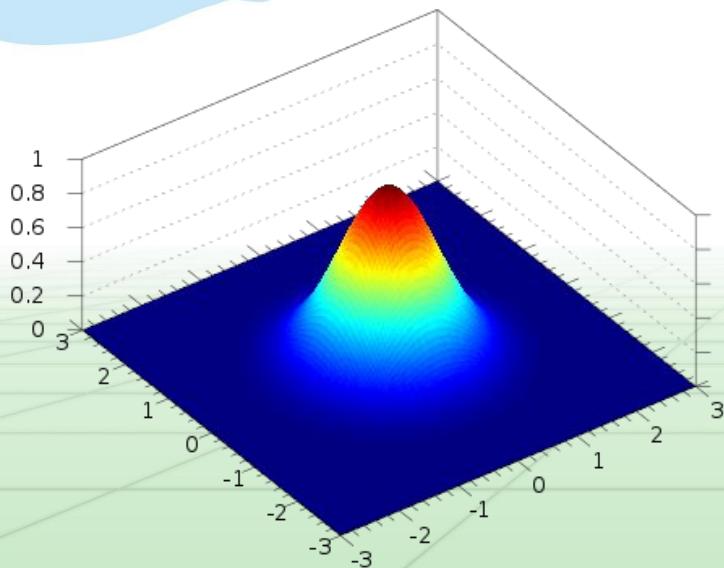
$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$



# 2d Gaussian

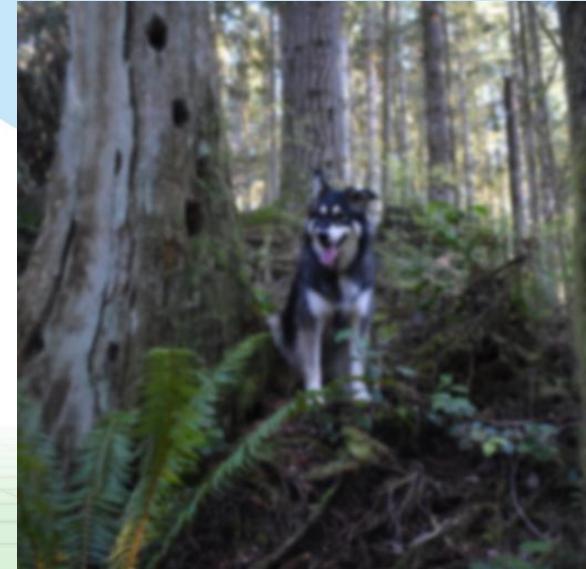
---

$$g(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$



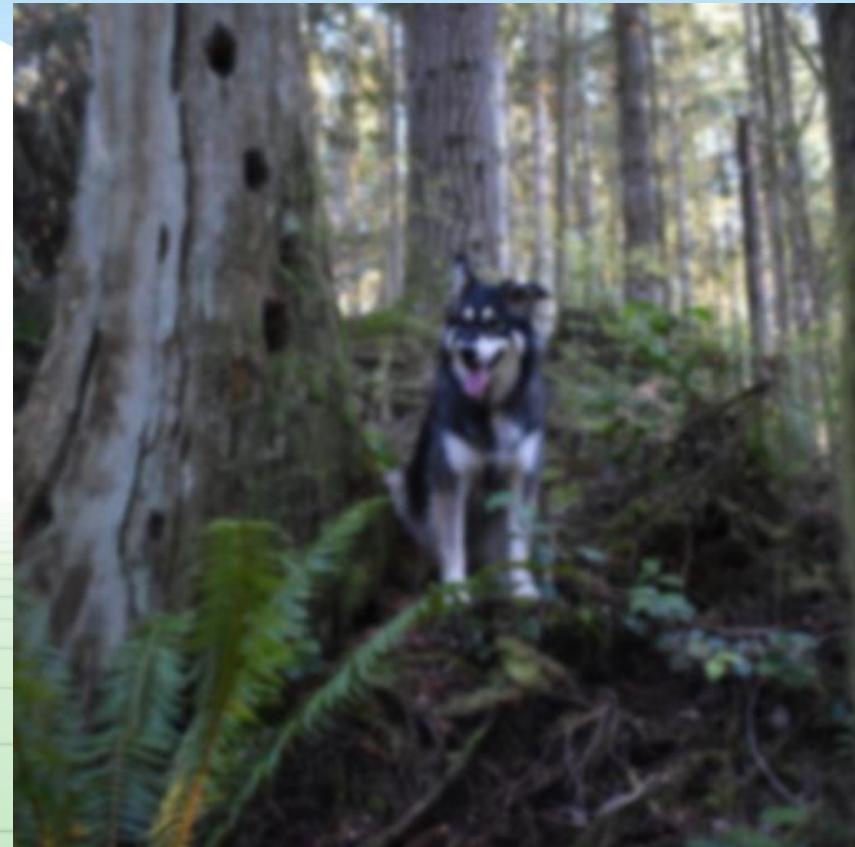
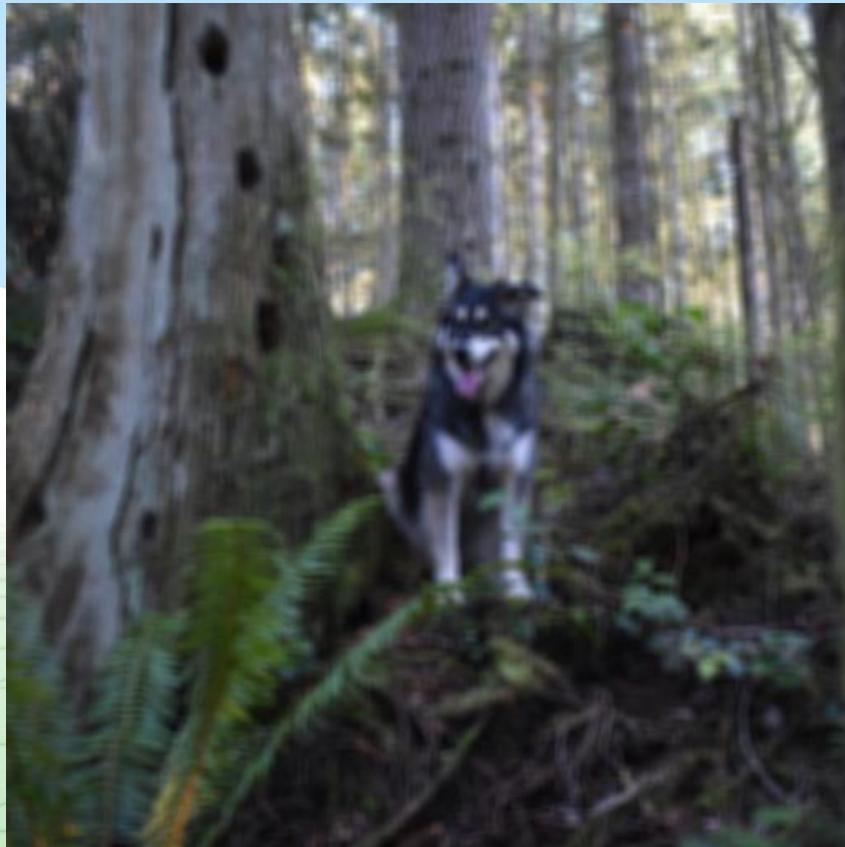
# Better smoothing with Gaussians

---

 $*$  $=$ 

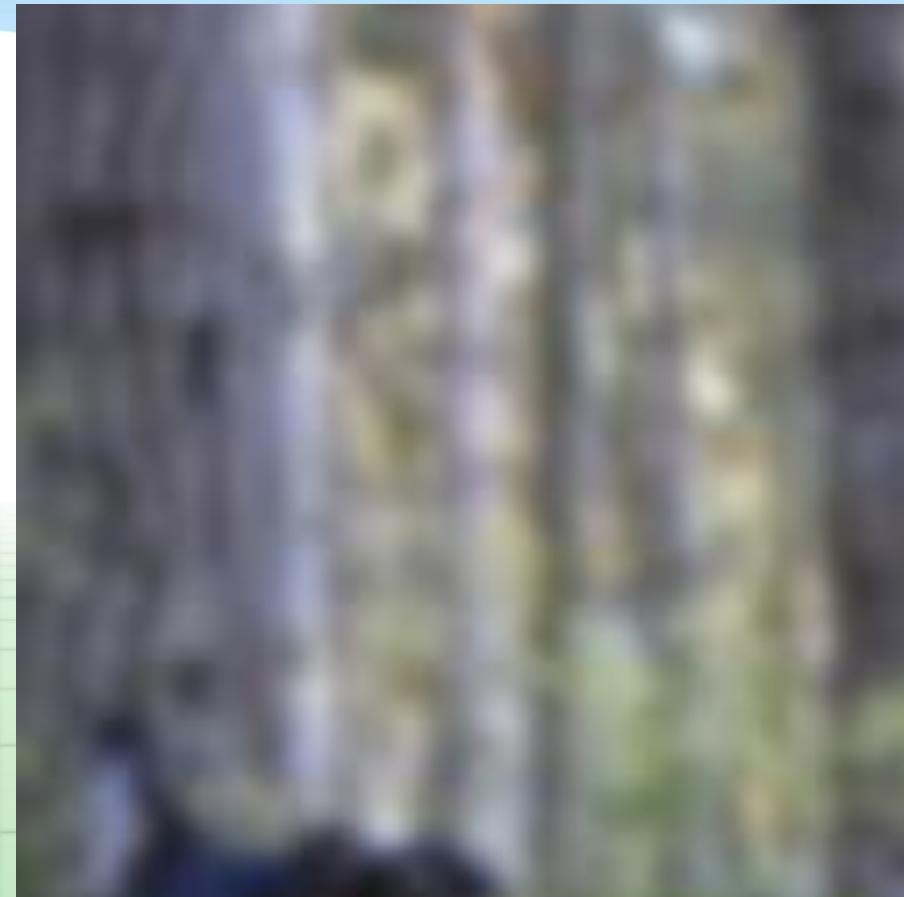
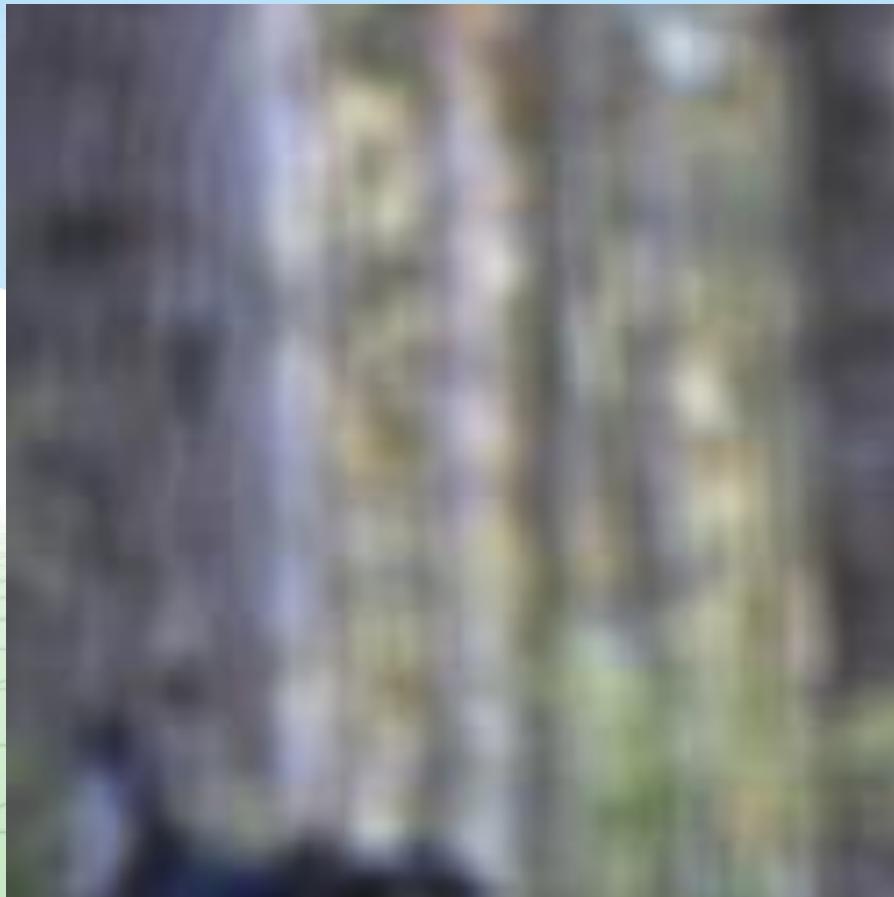
# Better smoothing with Gaussians

---

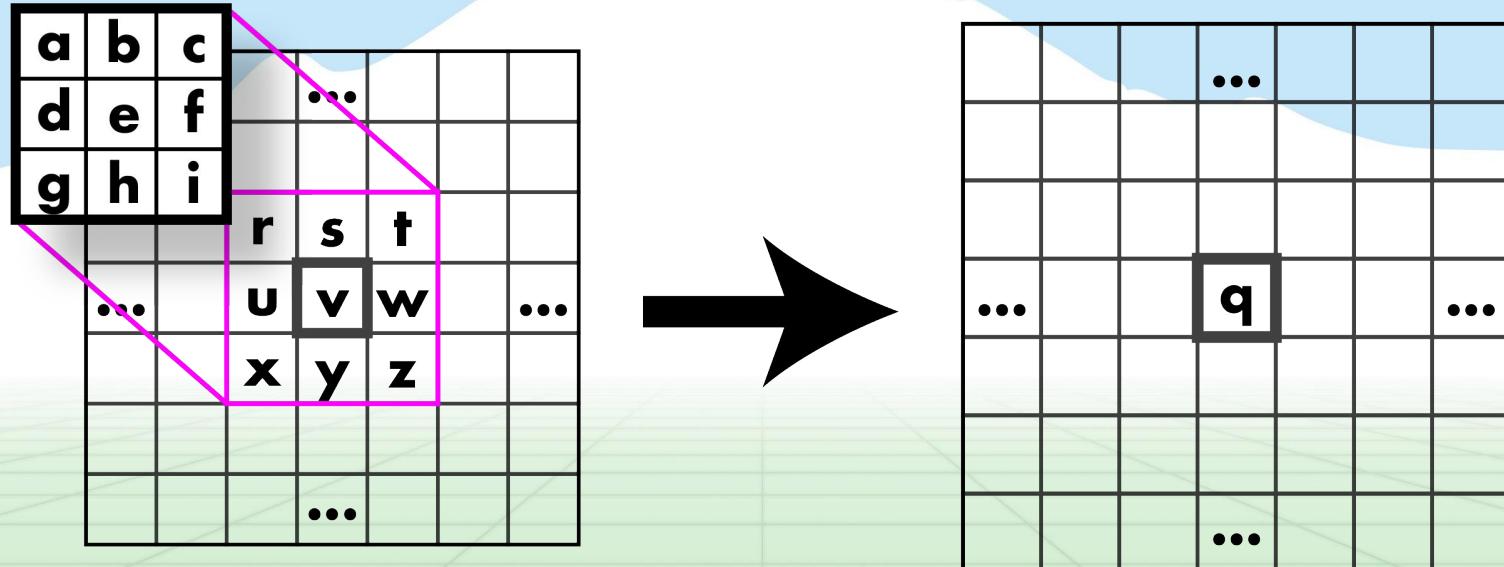


# Better smoothing with Gaussians

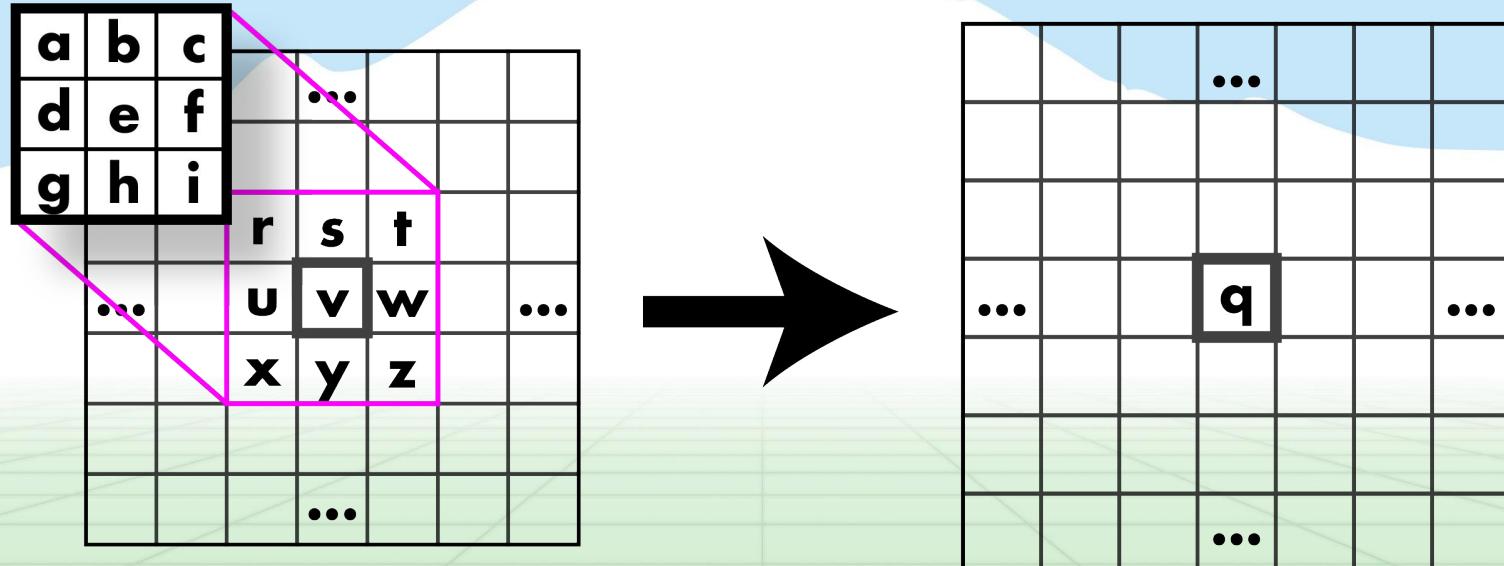
---



# Wow, so what was that convolution thing??



# Wow, so what was that convolution thing??



$$q = a \times r + b \times s + c \times t + d \times u + e \times v + f \times w + g \times x + h \times y + i \times z$$

# Calculate it, go!

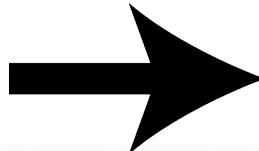
2	-1	-1
-1	2	-1
-1	-1	2

45	51	57
46	46	67
26	19	64

...	...	...
...	...	...



...	...	...
...	q	...
...	...	...

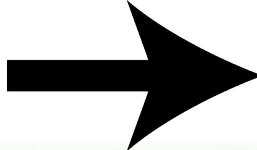
# Calculate it, go!

A diagram illustrating matrix multiplication. On the left, a 3x3 matrix of numbers is shown being multiplied by a 3x3 identity matrix. The numbers in the first row of the matrix are -1, -1, and -1. The numbers in the second row are -1, 8, and -1. The numbers in the third row are -1, -1, and -1. The identity matrix has 1s on the diagonal and 0s elsewhere. The result of the multiplication is a 3x3 matrix where each row is a scalar multiple of the first row of the original matrix. The first row of the result is -1, -1, and -1. The second row is 16, 105, and 153. The third row is 18, 111, and 136. Ellipses indicate that the pattern continues.

-1	-1	-1
-1	8	-1
-1	-1	-1

16	105	153
15	104	113
18	111	136



The final result of the multiplication is a 3x3 matrix where the central cell contains the letter q. All other cells are empty.

...		
...	q	
...		...

# Guess that kernel!

---

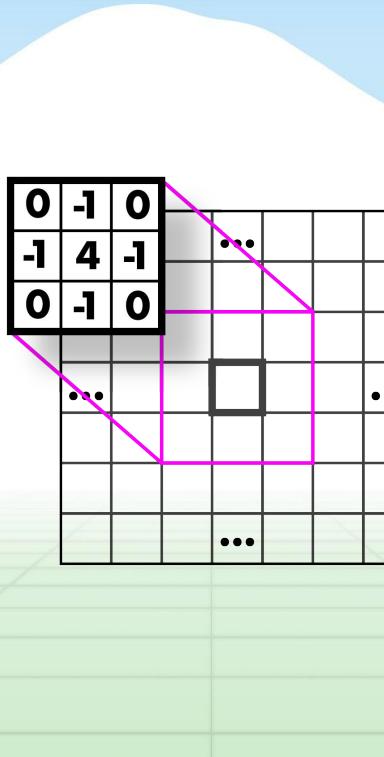


0	-1	0
-1	4	-1
0	-1	0

A 3x3 matrix with values 0, -1, and 4. A pink line starts from the top-left cell (0) and follows a path through the grid, ending at the center cell (4). Ellipses indicate the matrix continues beyond the visible 3x3 area.

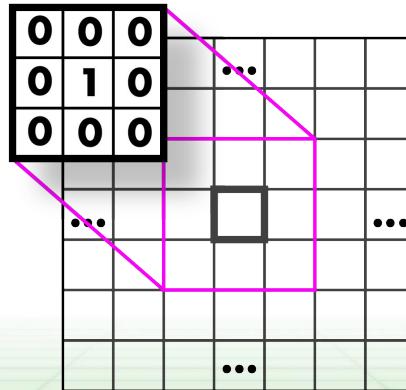
...  
...  
...  
...  
...  
...

# Highpass Kernel: finds edges

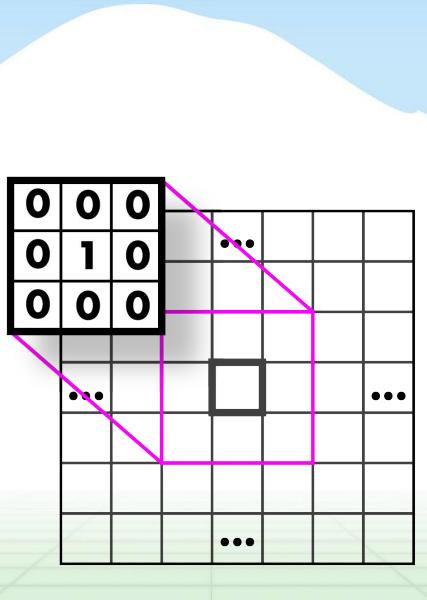


# Guess that kernel!

---

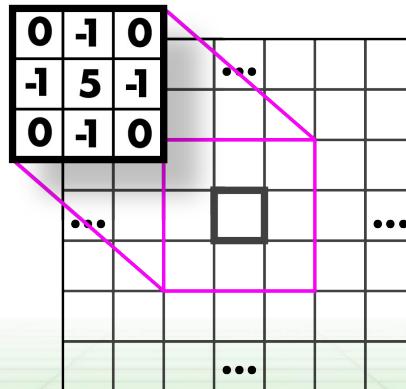


# Identity Kernel: Does nothing!



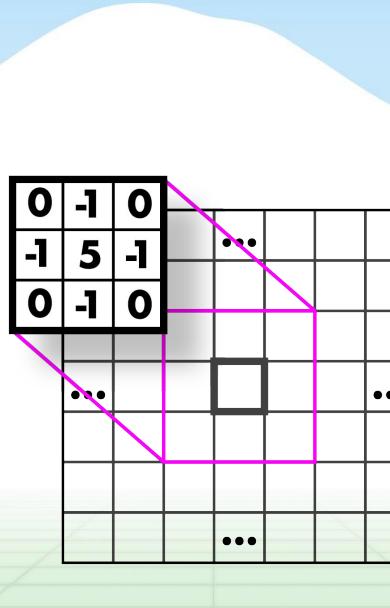
# Guess that kernel!

---



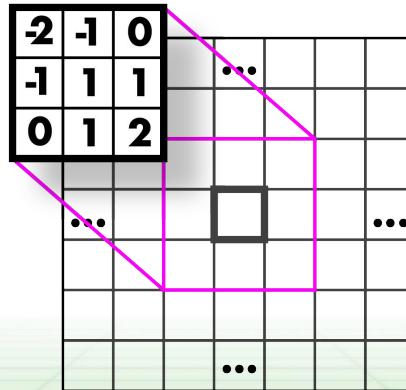
# Sharpen Kernel: sharpens!

---



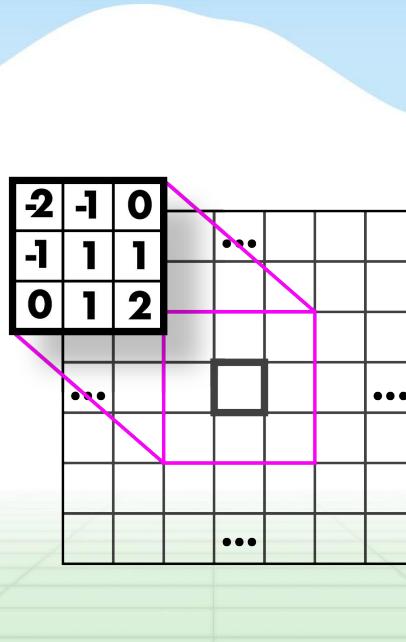
Note: sharpen = highpass + identity!

# Guess that kernel!



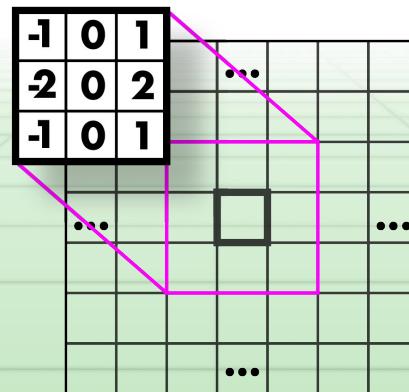
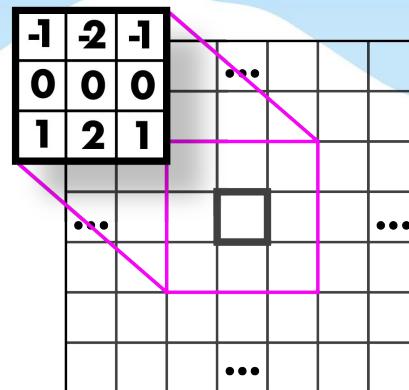
# Emboss Kernel: stylin'

---

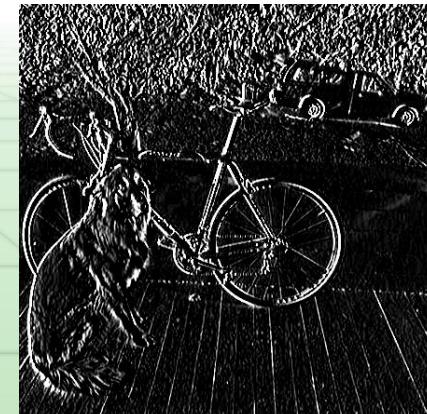
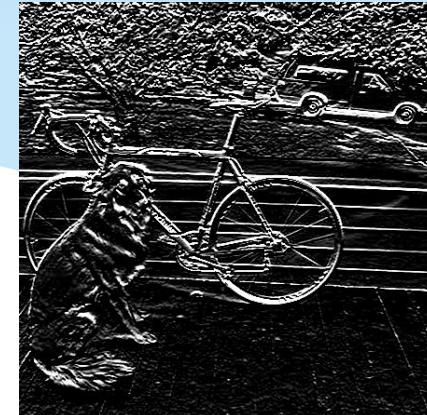
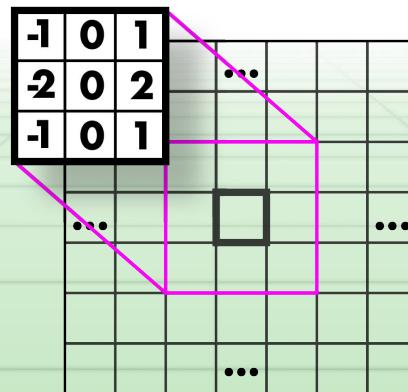
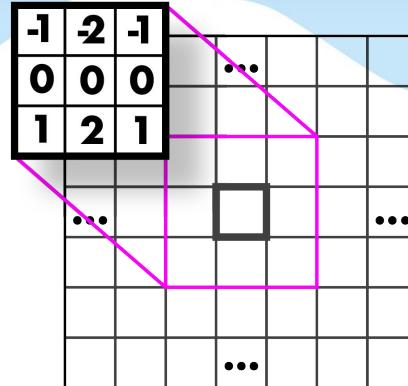


# Guess those kernels!

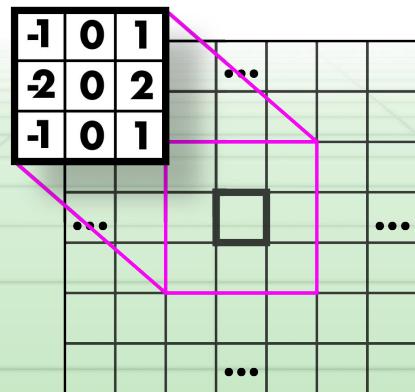
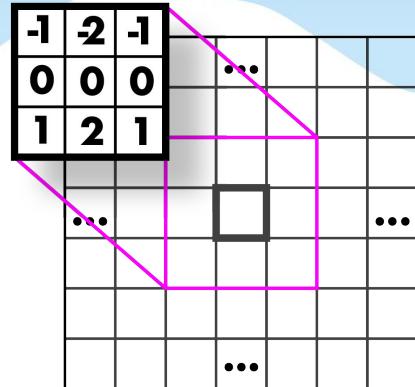
---



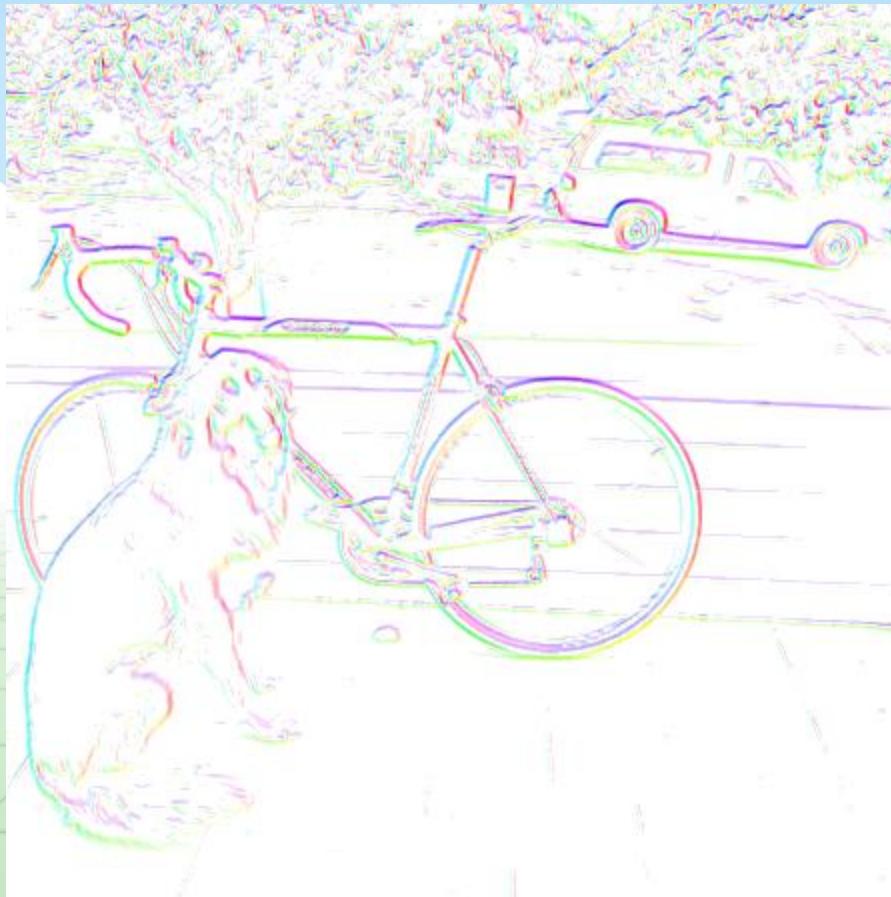
# Sobel Kernels: edges and...



# Sobel Kernels: edges and gradient!



# Sobel Kernels: edges and gradient!



And so much more!!

