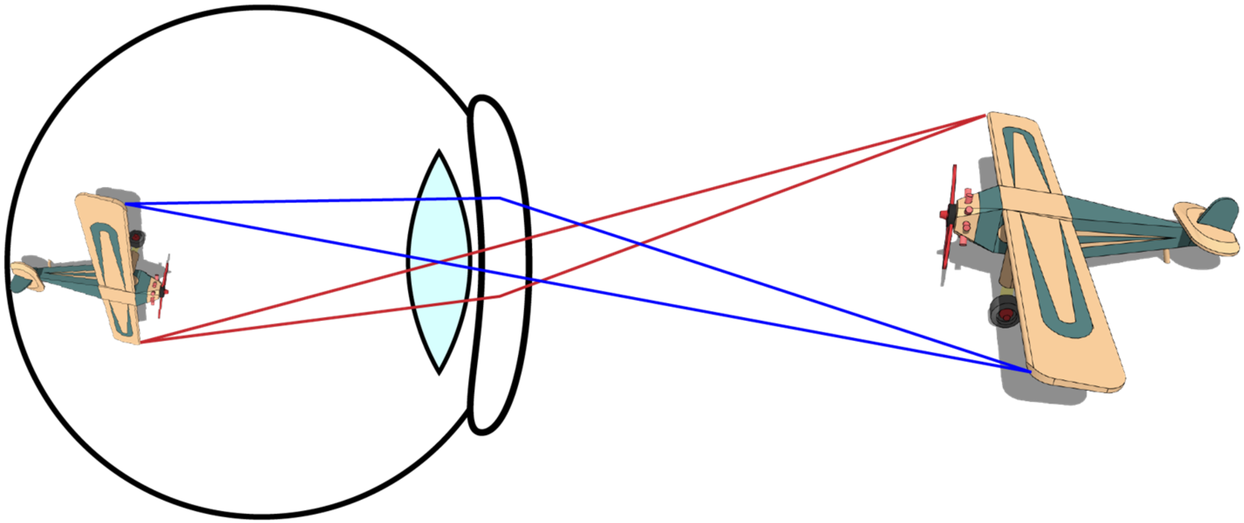
Computer Vision

CSE 455
Image Coordinates and Resizing

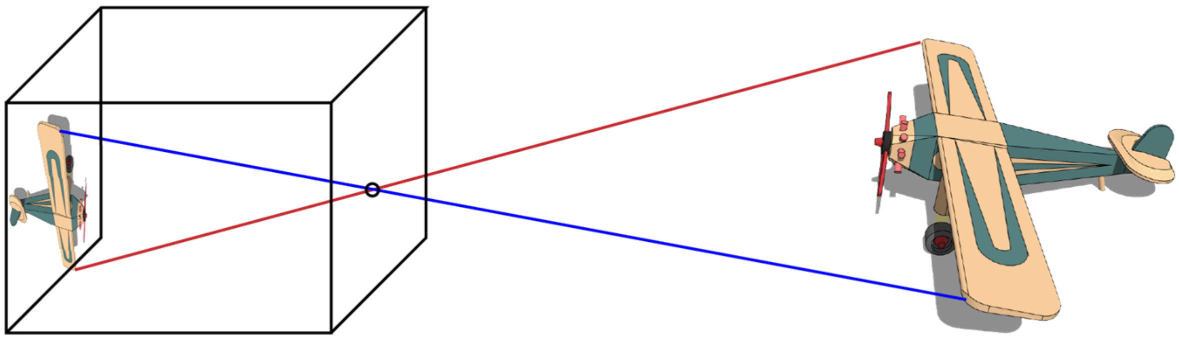
Linda Shapiro
Professor of Computer Science & Engineering
Professor of Electrical Engineering

What is an image?

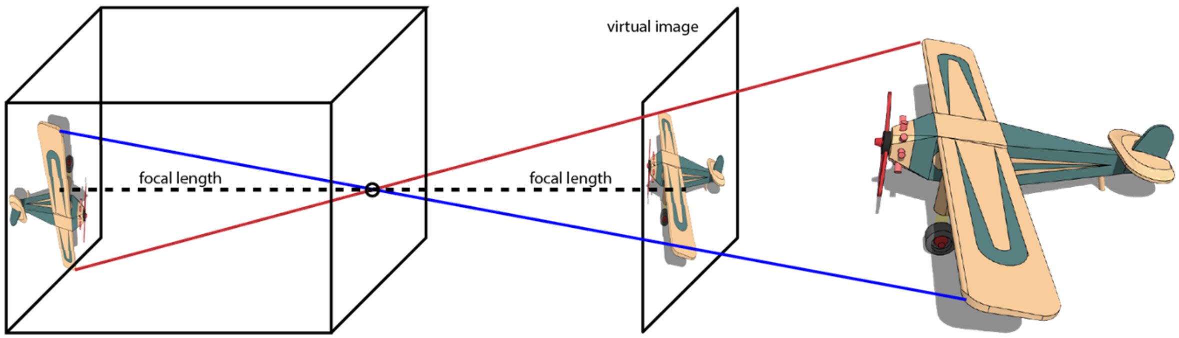
Eyes: projection onto retina



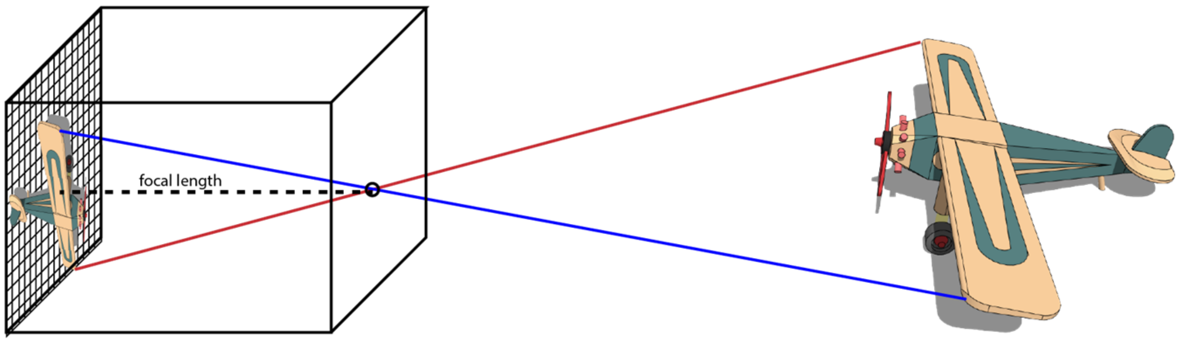
Model: pinhole camera



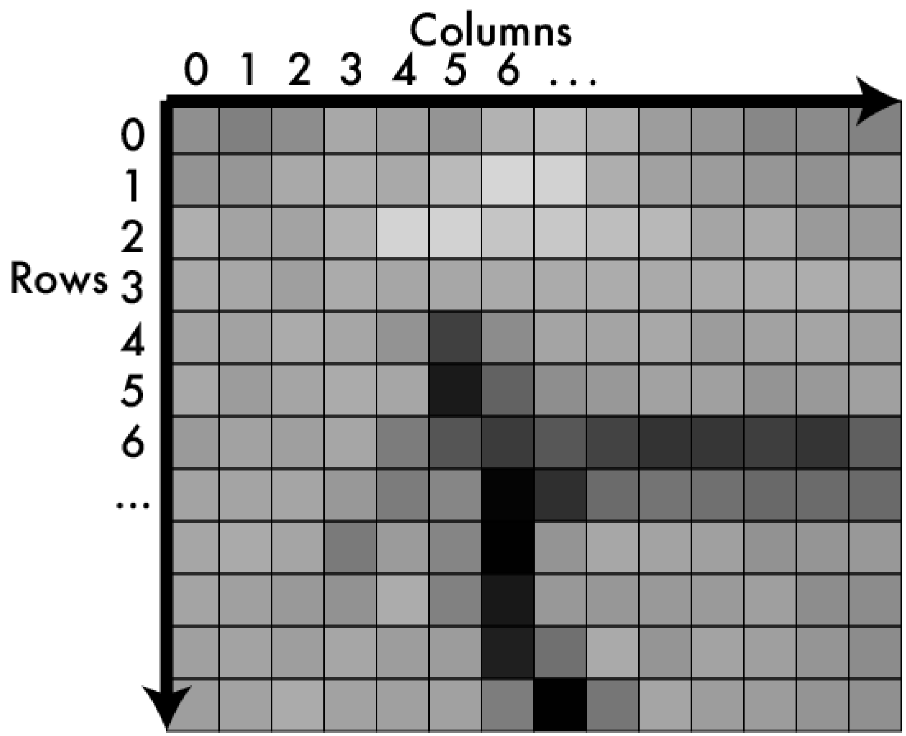
Model: pinhole camera



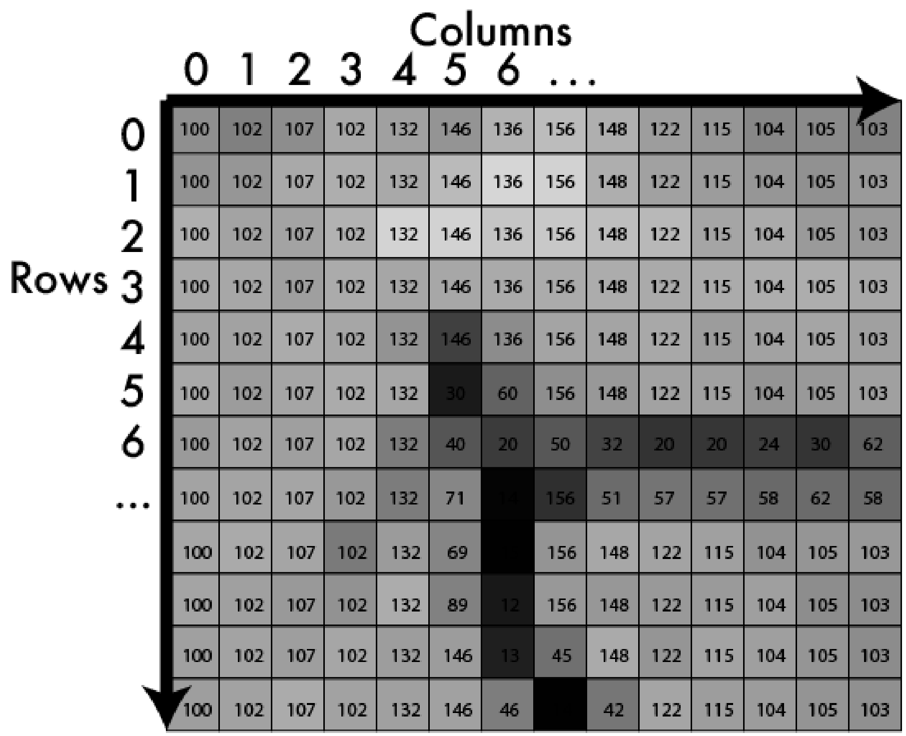
At each point we record incident light



An image is a matrix of light

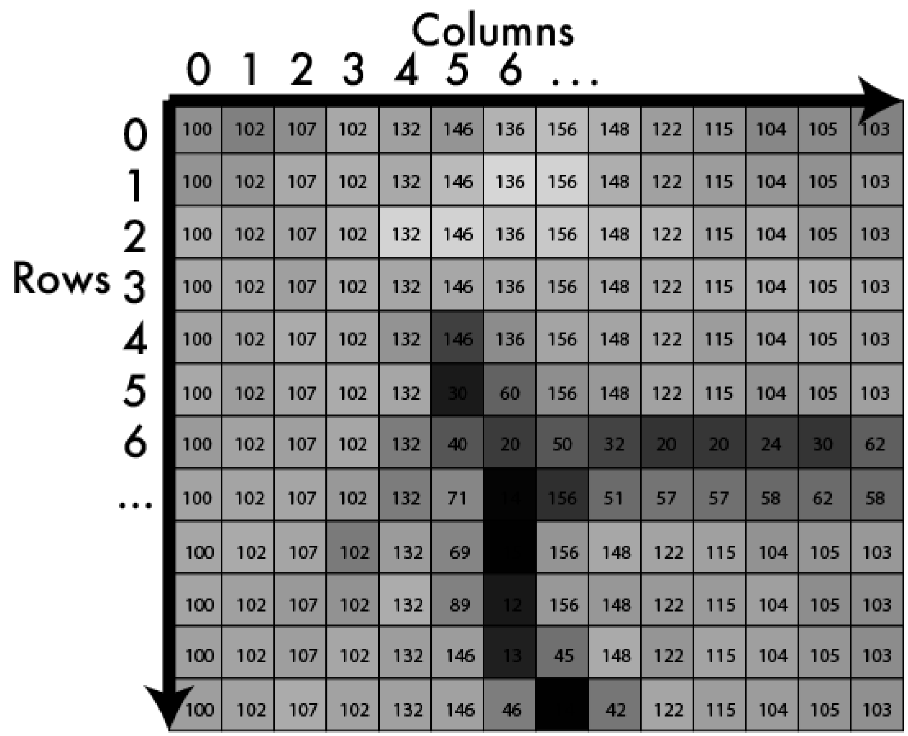


Values in matrix = how much light



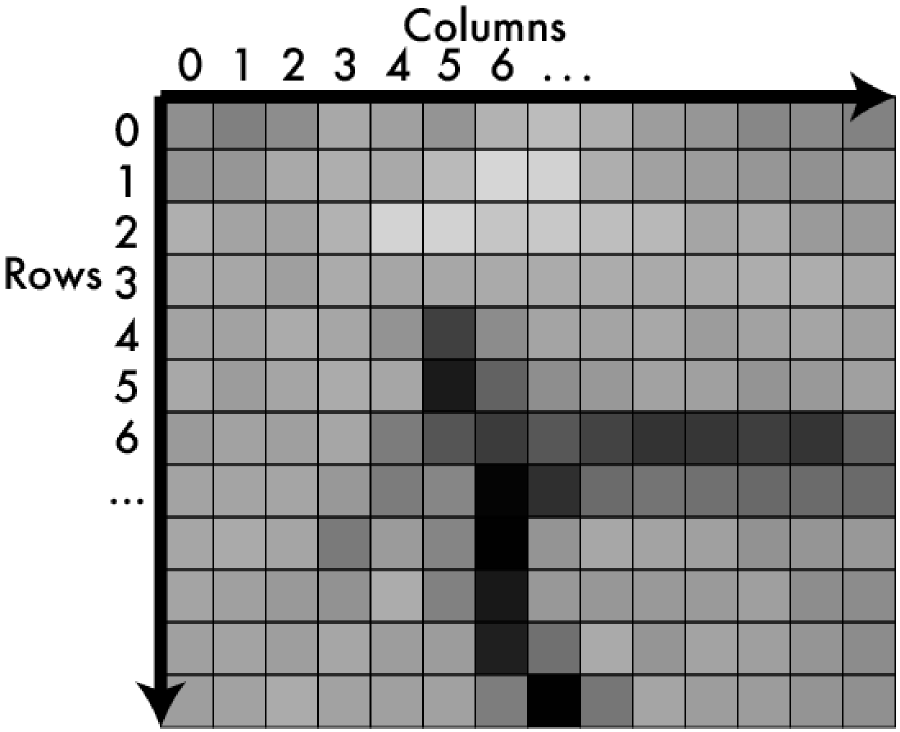
Values in matrix = how much light

- Higher = more light
- Lower = less light
- Bounded
  - No light = 0
  - Sensor/device limit = max
  - Typical ranges:
    - [0-255], fit into byte
    - [0-1], floating point
- Called **pixels**

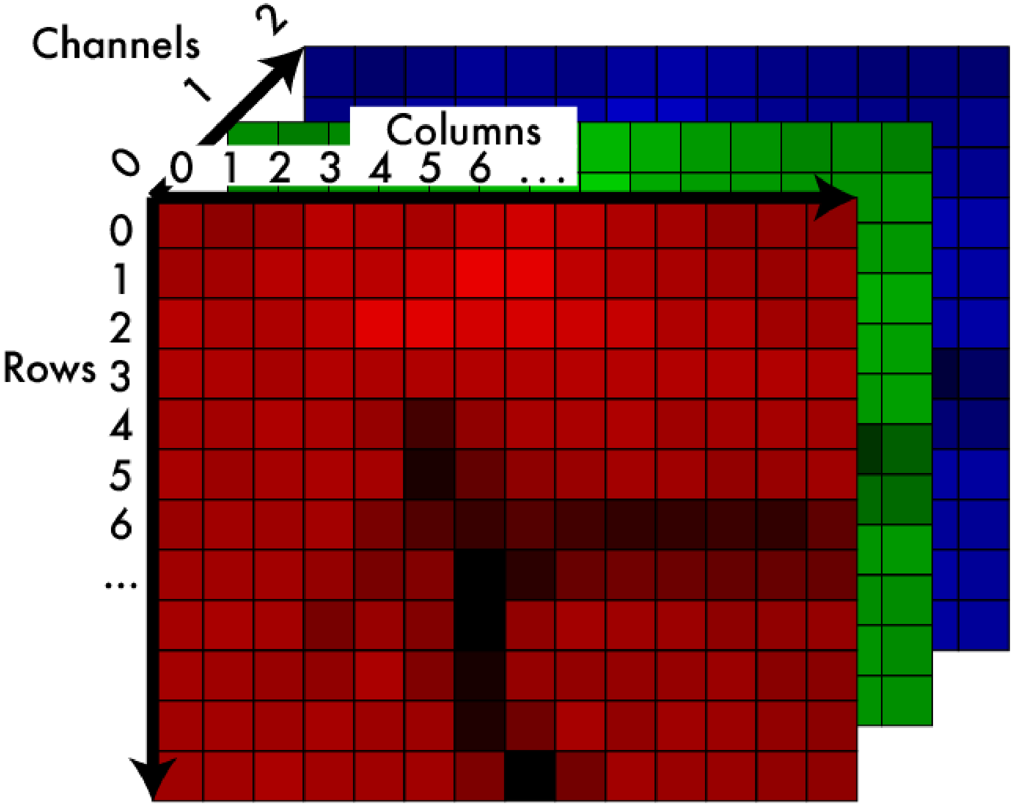


Addressing pixels

- Ways to index:
    - (r,c)
        - Like matrix notation
        - (3,6) is row 3 column 6
    - **(x,y)**
        - Like cartesian coordinates (but from the TOP)
        - (3,6) is column 3 row 6
- **We use (x,y)**
    - So does your homework!
    - Arbitrary
    - Only thing that matters is consistency



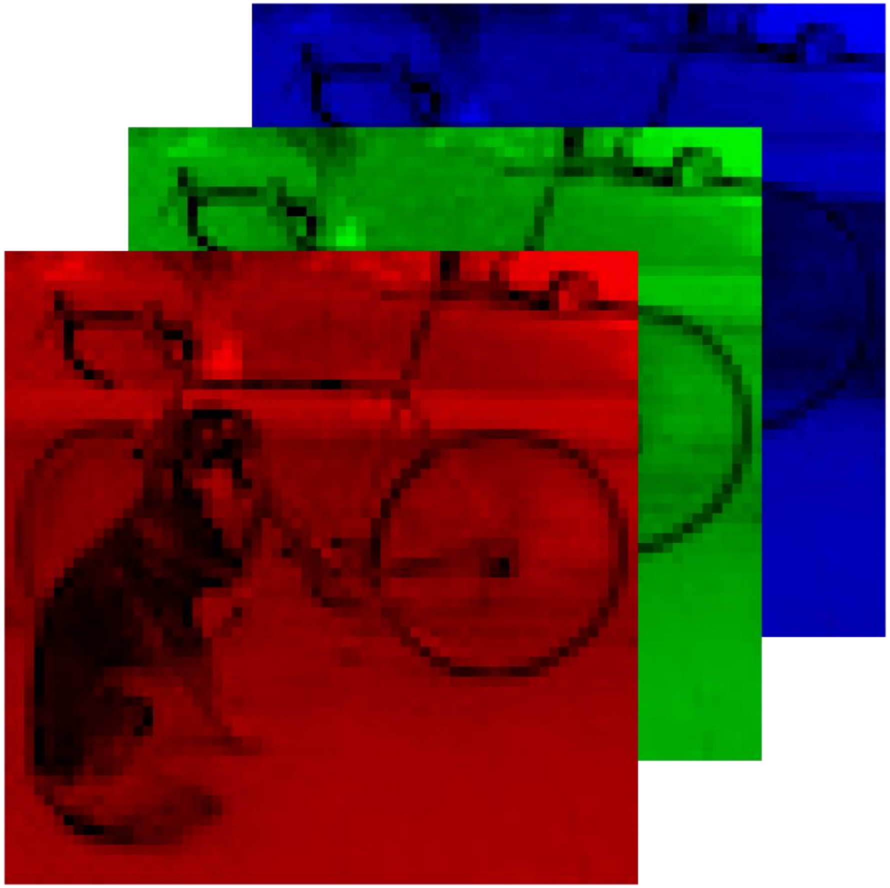
Color image: 3d tensor in colorspace



RGB information in separate "channels"

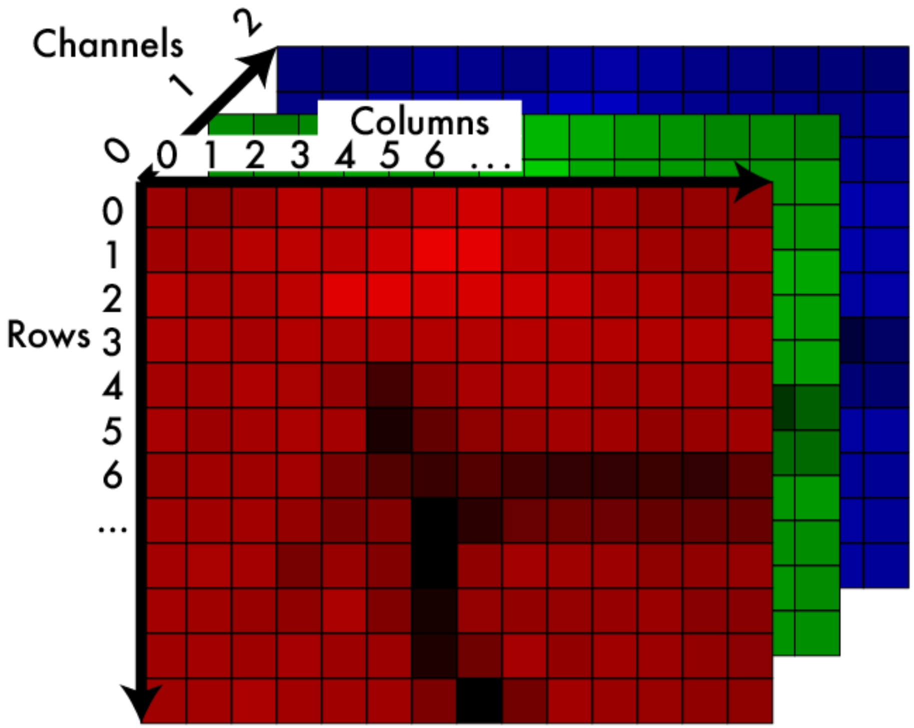
Remember: we can match "real" colors using a mix of primaries.

Each channel encodes one primary. Adding the light produced from each primary mimics the original color.

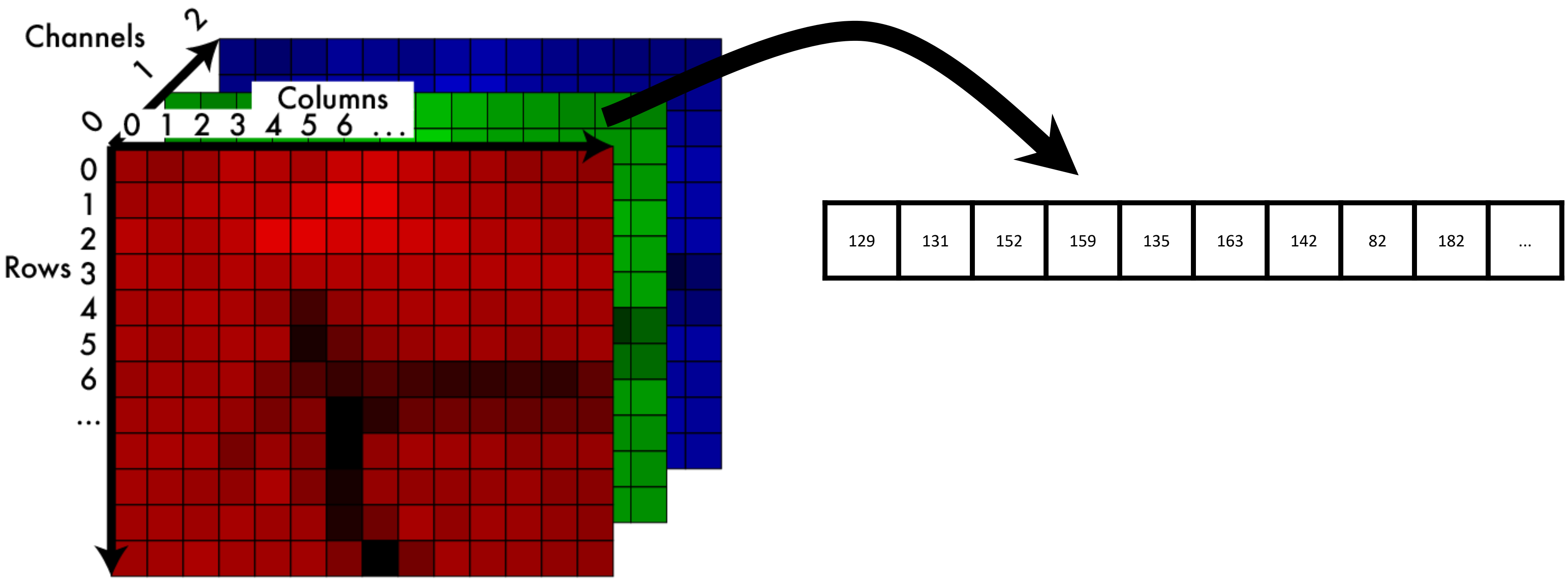


Addressing pixels

- We use (x,y,c)
  - (1,2,0):
    - column 1, row 2, channel 0
- Be consistent
- But do what we do for homeworks :-)
- Also for size:
  - 1920 x 1080 x 3 image:
    - 1920 px wide
    - 1080 px tall
    - 3 channels

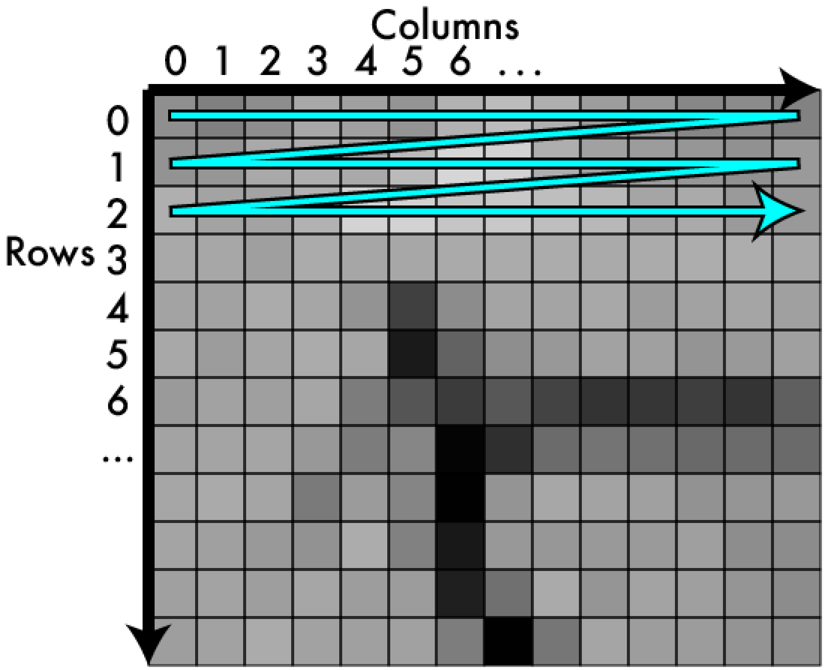


How do we store them?

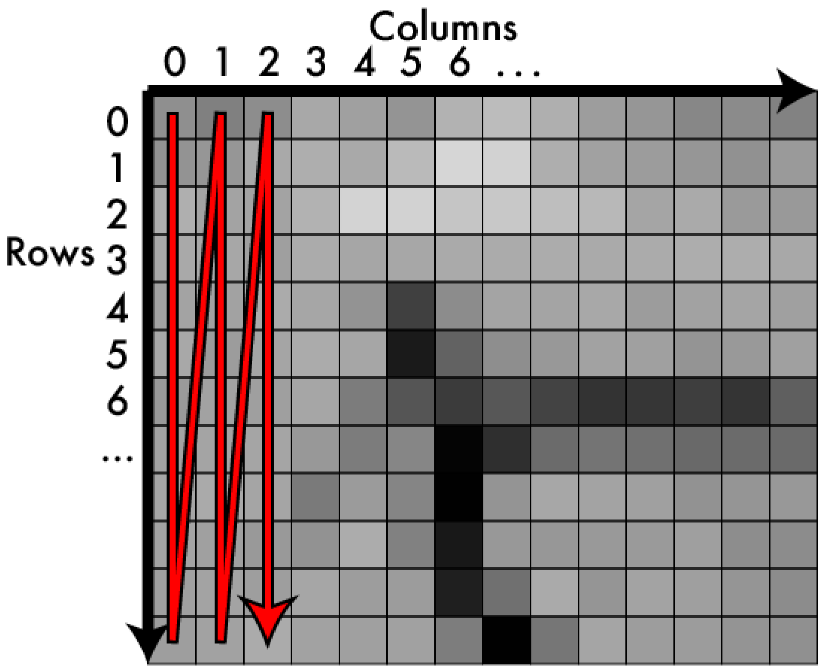


Storage: row major vs column major

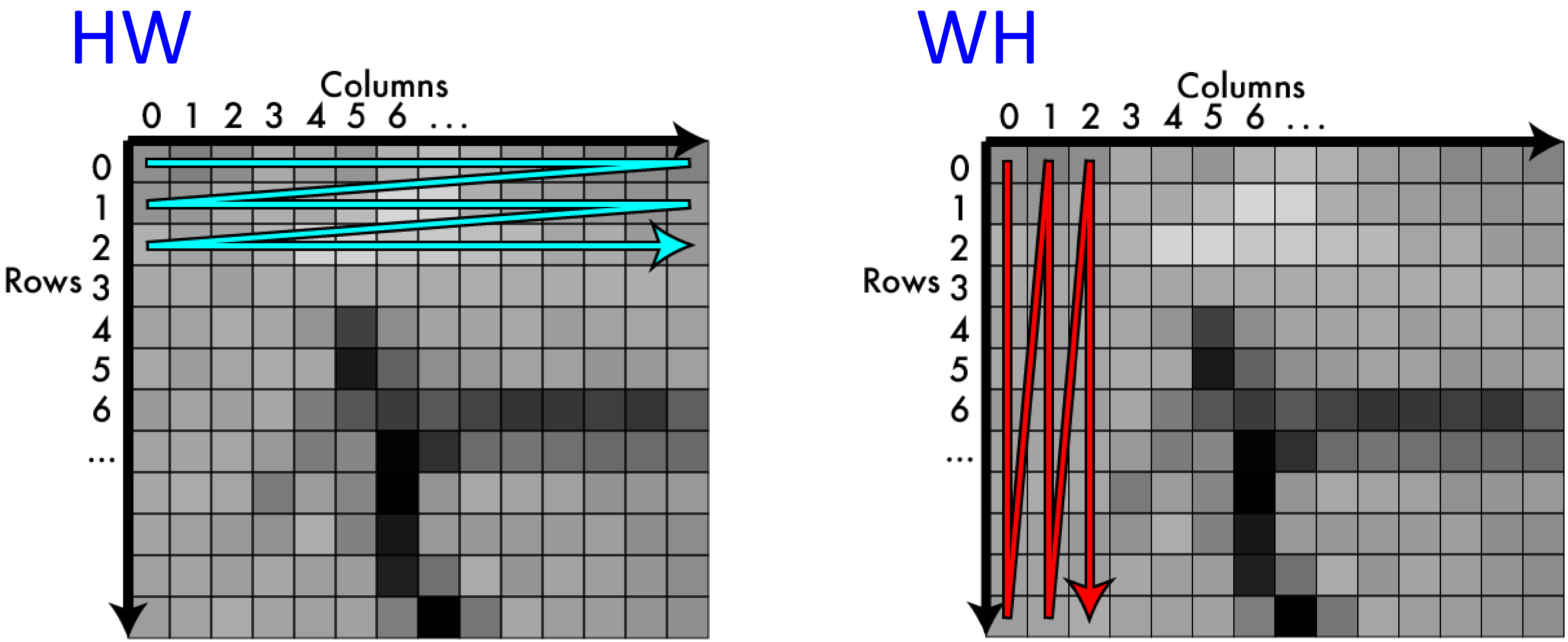
Row Major



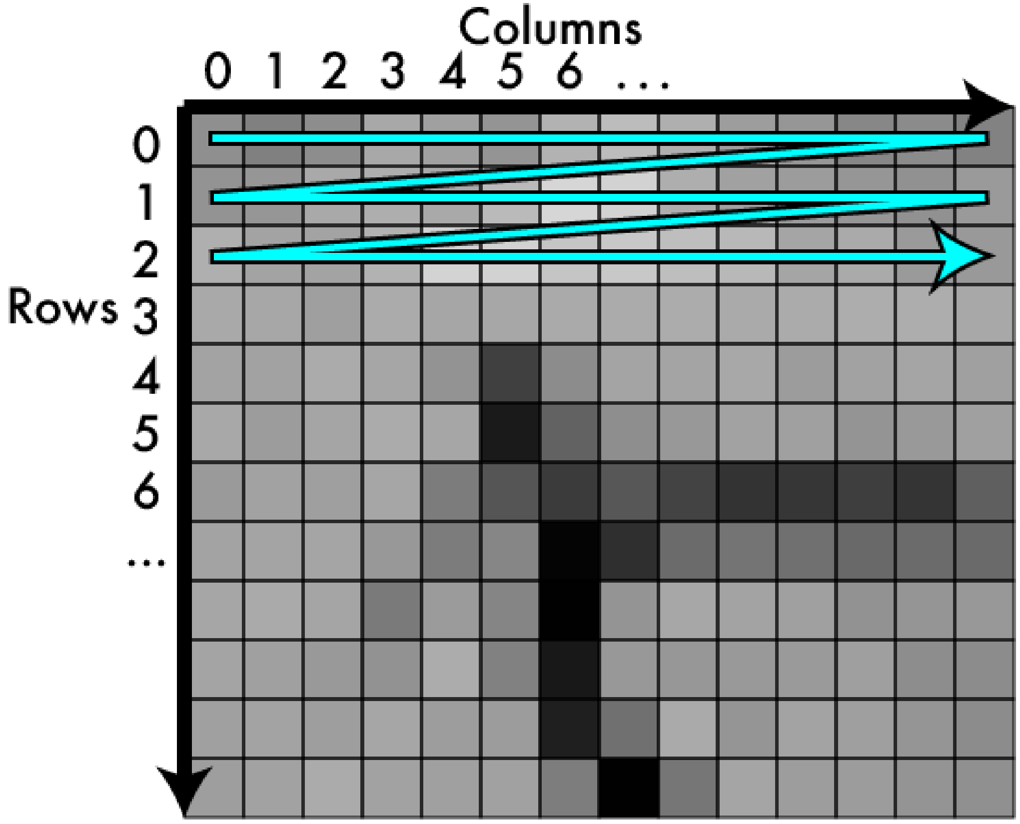
Column Major



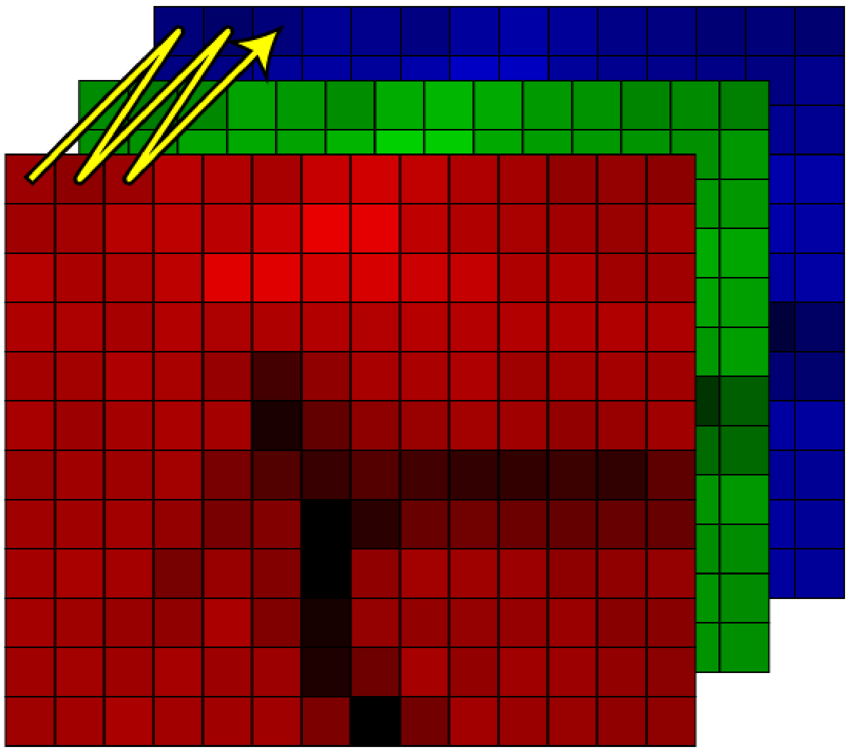
Storage: row major vs column major

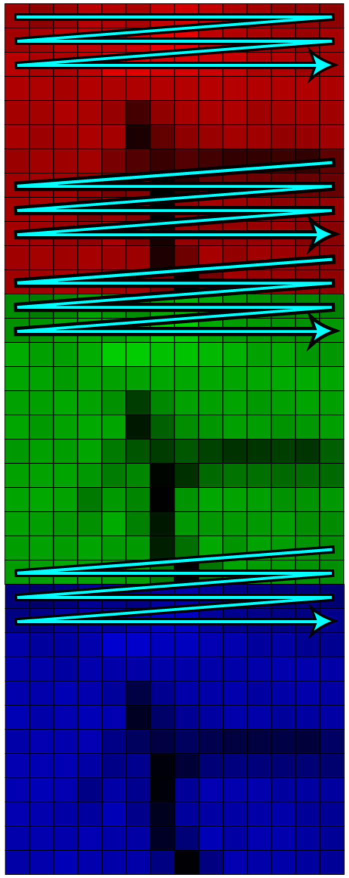


Typically use row-major or HW

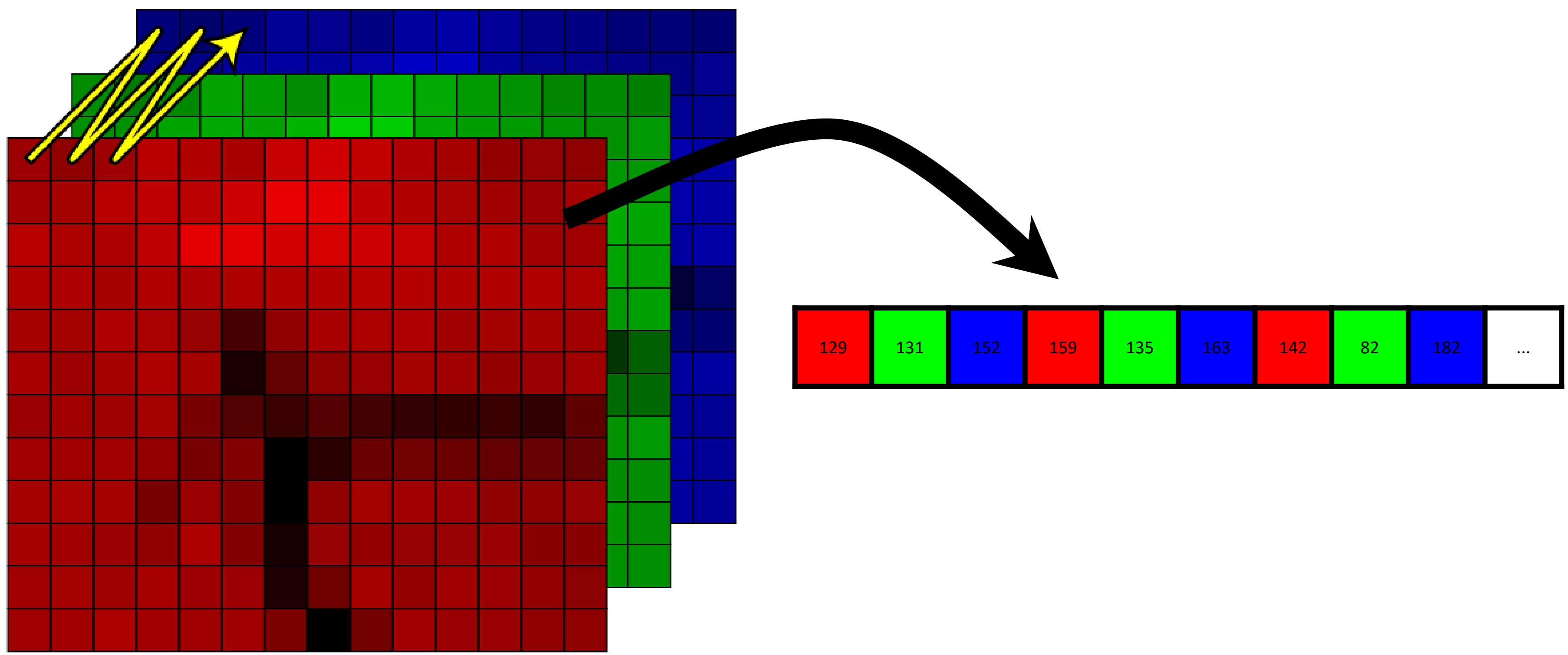


In 3d we have more choices!

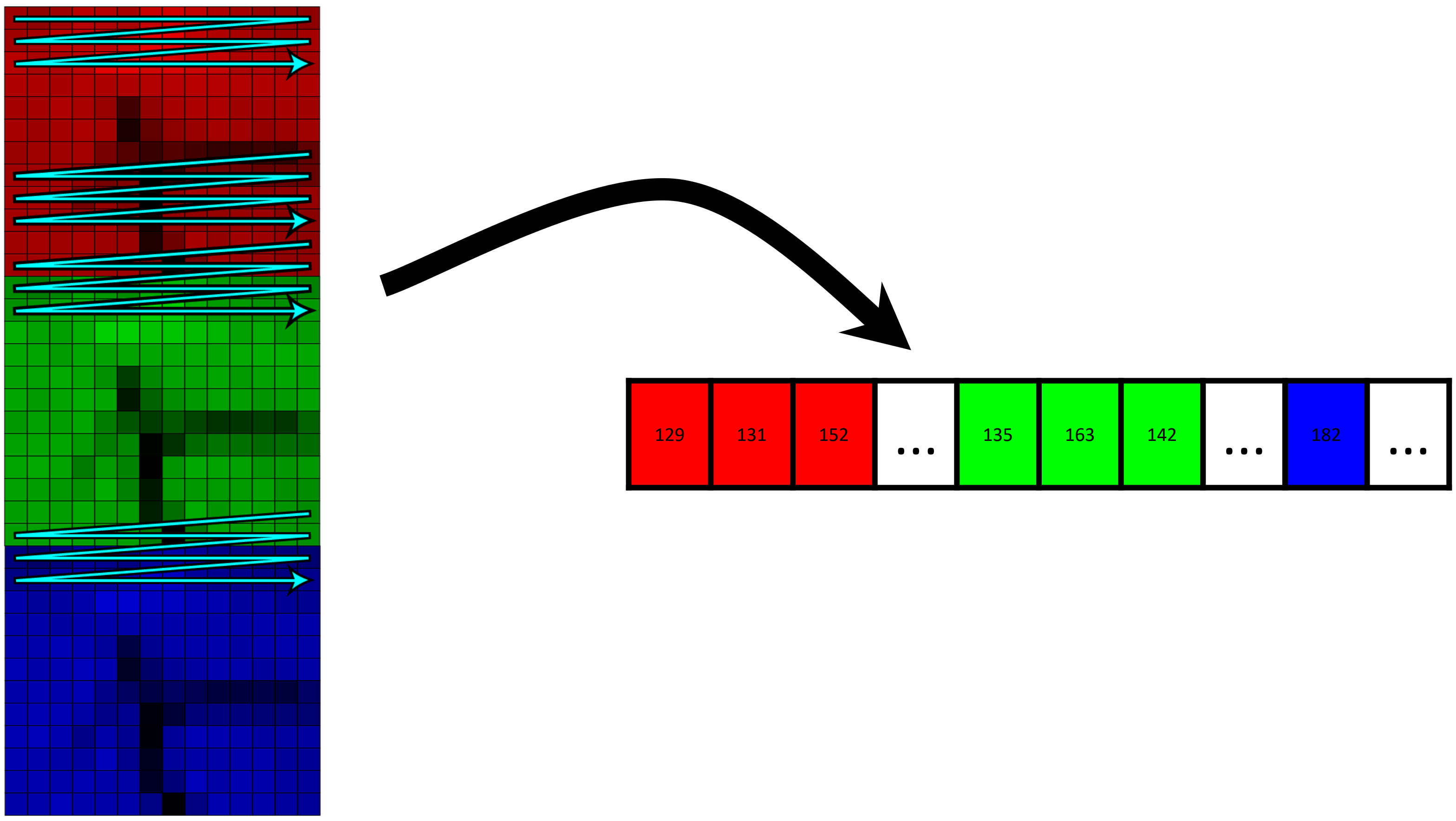




HWC: channels interleaved



CHW: channels separated



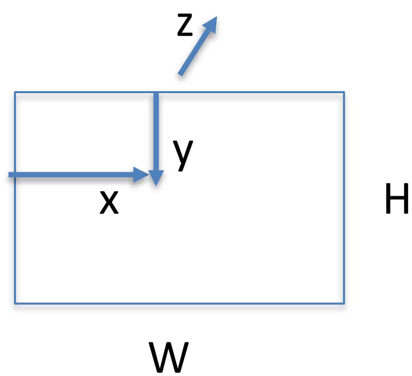
CHW Pop quiz

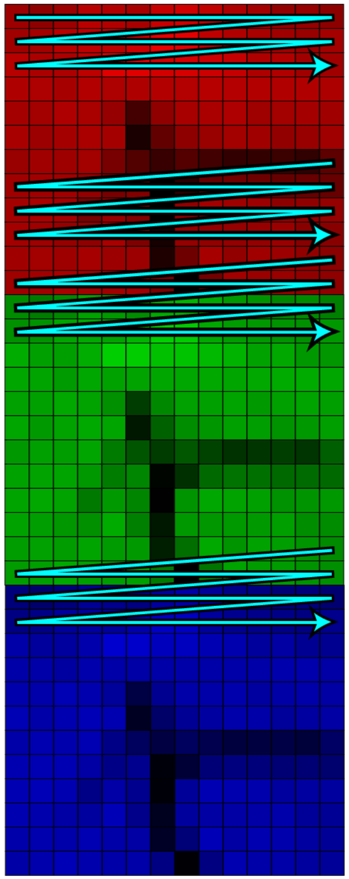
We'll use CHW, it's what a lot of other libraries use.

In an array for a 1920 x 1080 x 3 image what entry would contain the pixel (15,192,2)?

Formula:

x + y*W + z*W*H





CHW Pop quiz

In an array for a 1920 x 1080 x 3 image what entry would contain the pixel (15,192,2)?

In general for (x,y,z) of image (W,H,C)

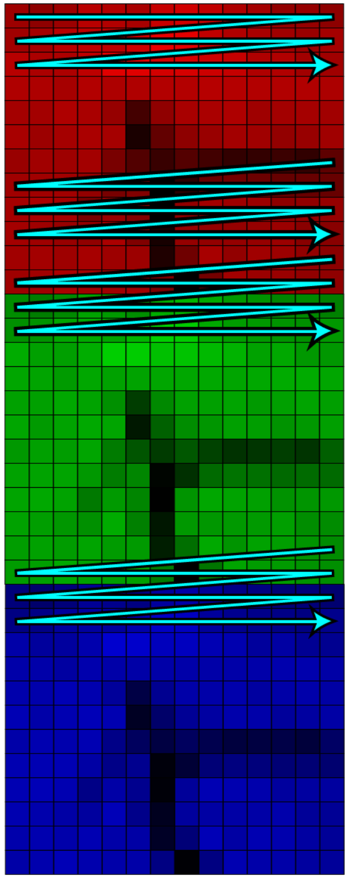
x + y*W + z*W*H

15 + 192*1920 + 2*1920*1080 = 4,515,855

Remember, everything is 0 indexed

Where does (0,0,0) go?

Position 0 + 0 + 0 = 0



In your homework

```
typedef struct {
    int w,h,c;
    float *data;
} image;
```

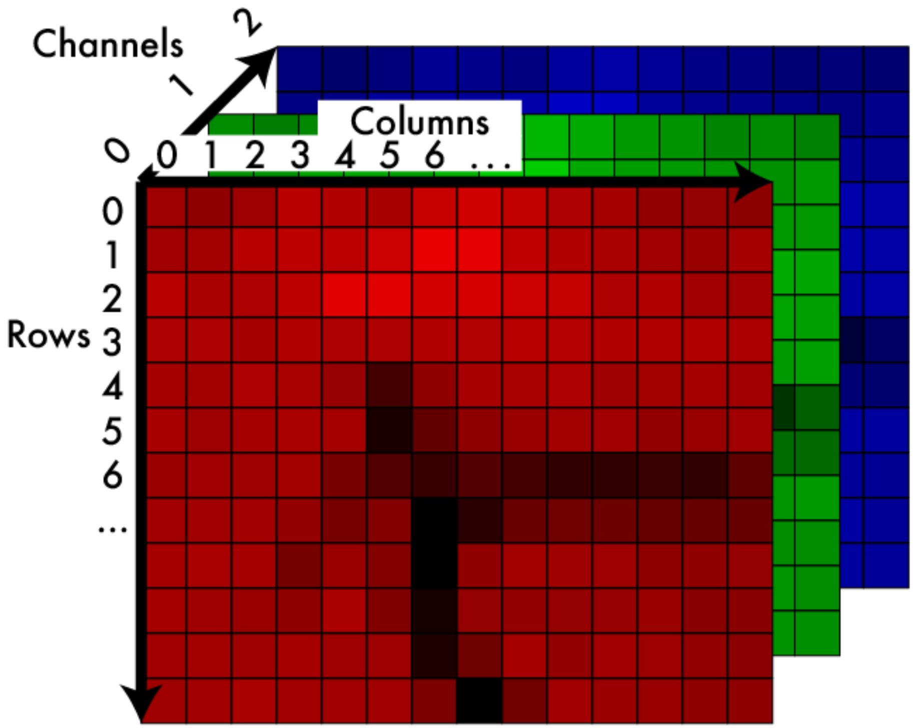


Image interpolation and resizing

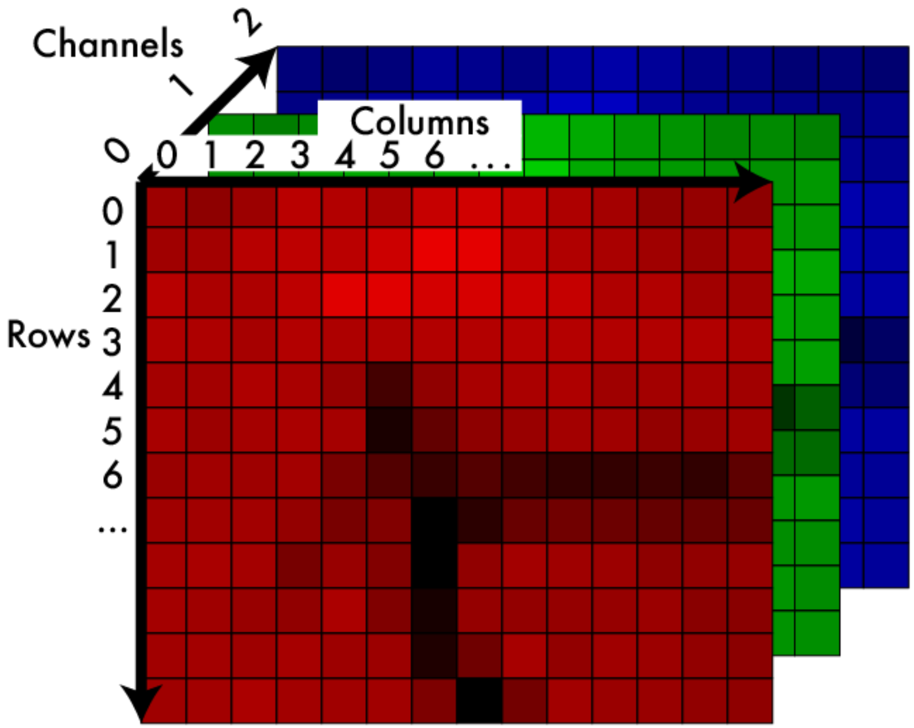
An image is kinda like a function

An image is a mapping from indices to pixel value:

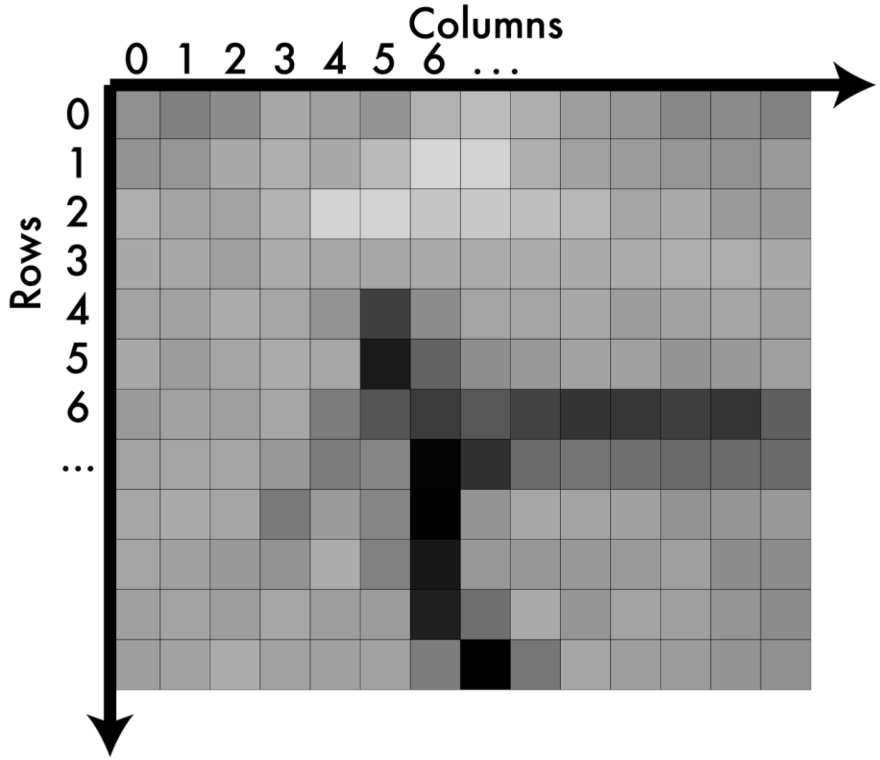
- Im: I x I x I -> R

We may want to pass in non-integers:

- Im’: R x R x I -> R

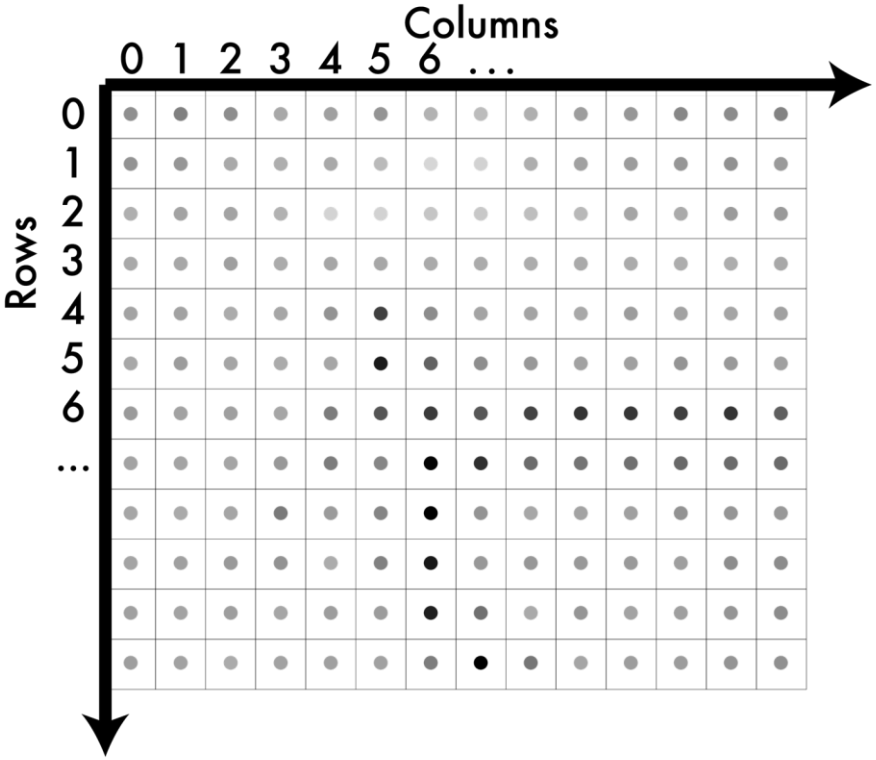


A note on coordinates in images



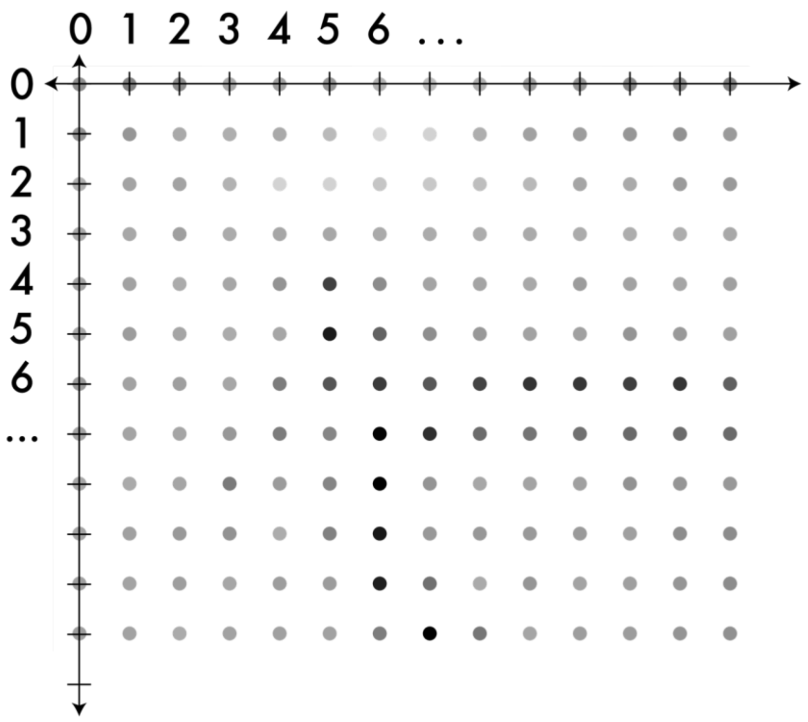
integer pixels

A note on coordinates in images



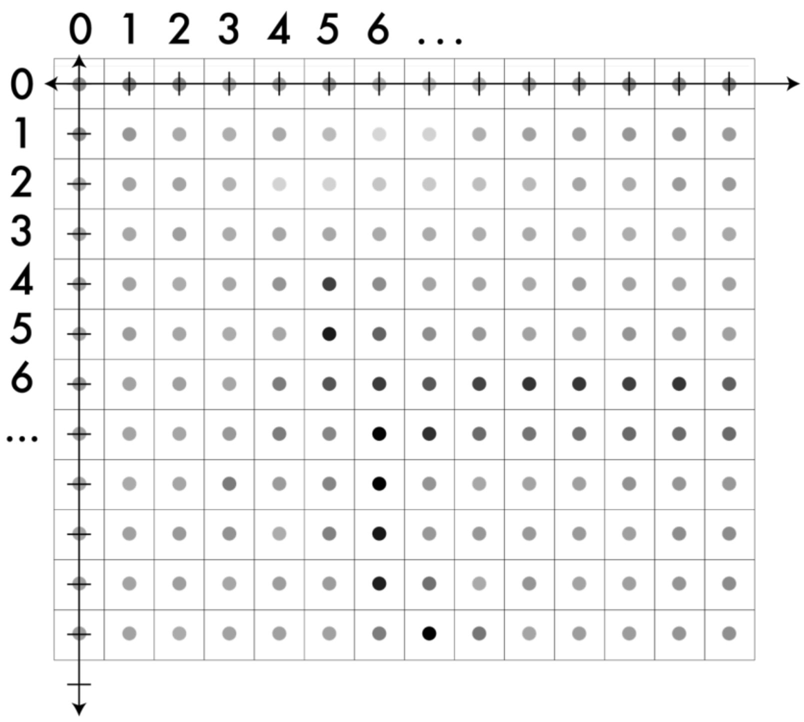
We can think of their values as being at the centers.

A note on coordinates in images



Now we can move to a real coordinate system.

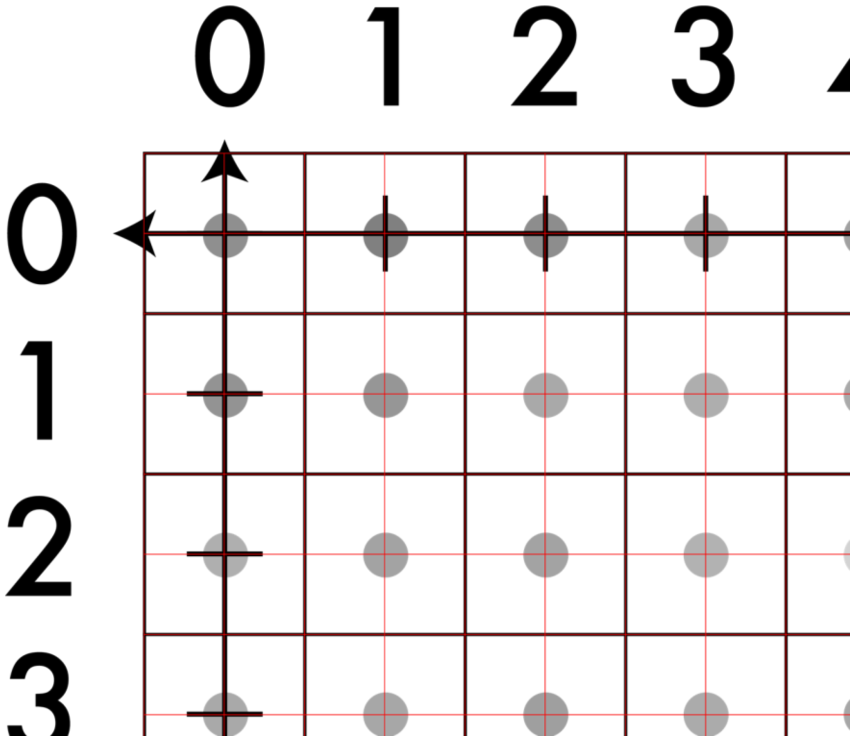
A note on coordinates in images



On the image

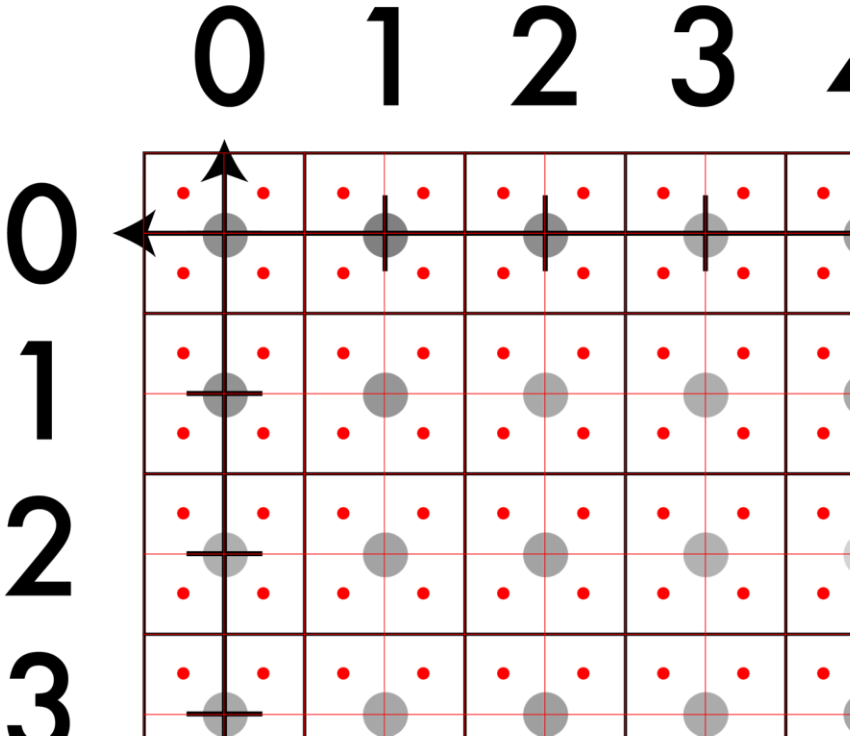
A note on coordinates in images

So, the value of the pixel (x,y) is now centered at (x,y).

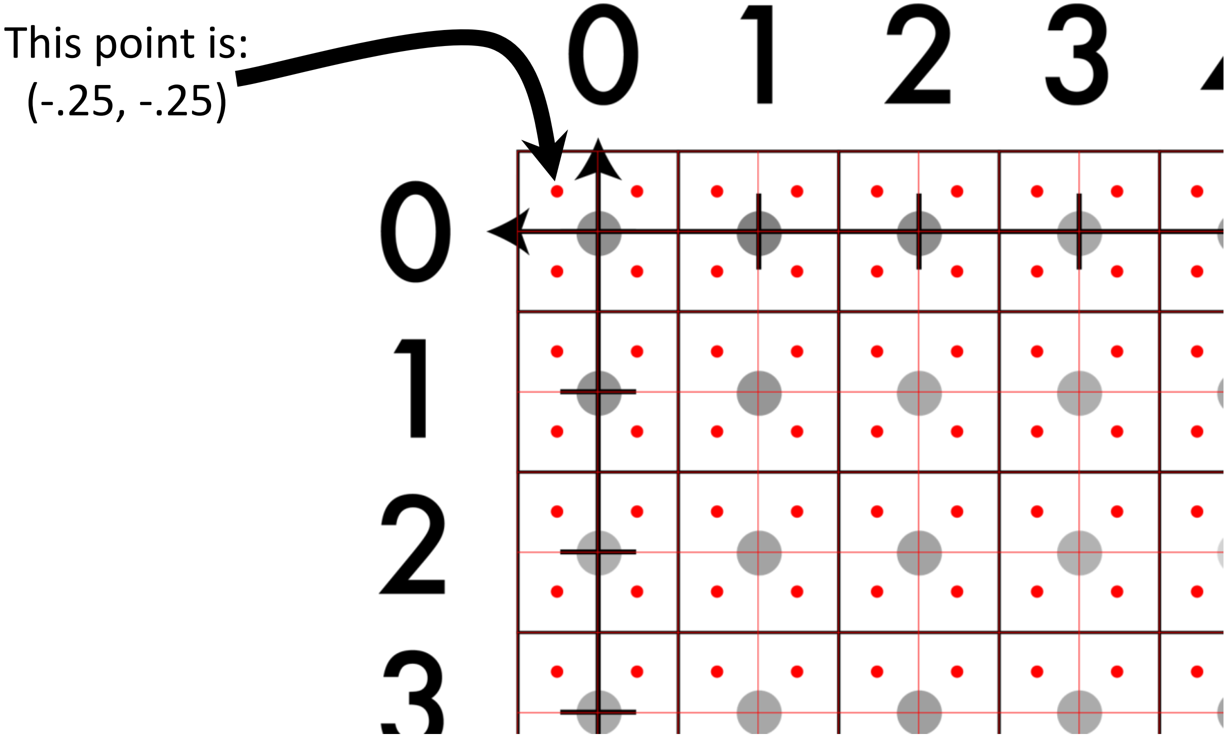


A note on coordinates in images

But there are other real-valued points.

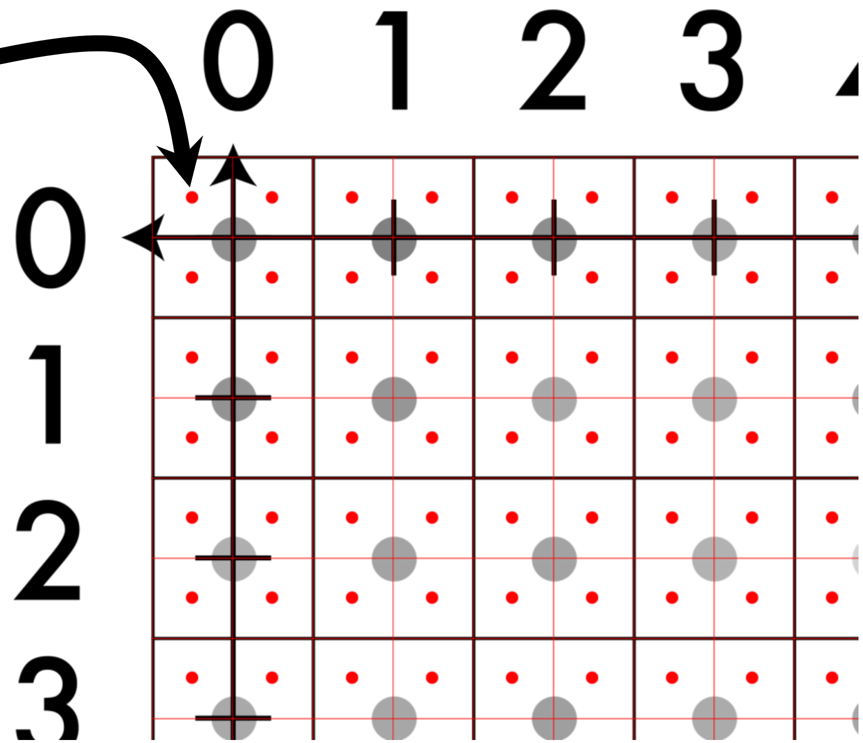


A note on coordinates in images



Just be careful

This point is:
(-.25, -.25)



Interpolation

- How do we find out the VALUE of a non-integer point, when the image only comes with integer points, ie (25,45,3).
- For our assignment:
  1. Nearest-Neighbor Interpolation
  2. Bilinear Interpolation

Nearest neighbor: what it sounds like

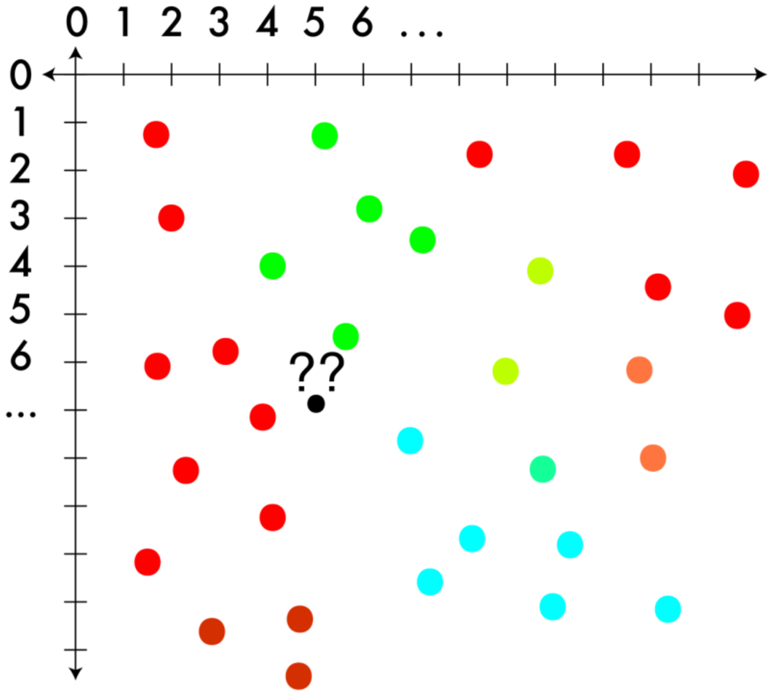
f(x,y,z) = Im(round(x), round(y), z)

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

Triangle interpolation: for less structured image (alternate approach)

Sometimes you have a regular grid, sometimes you don't.

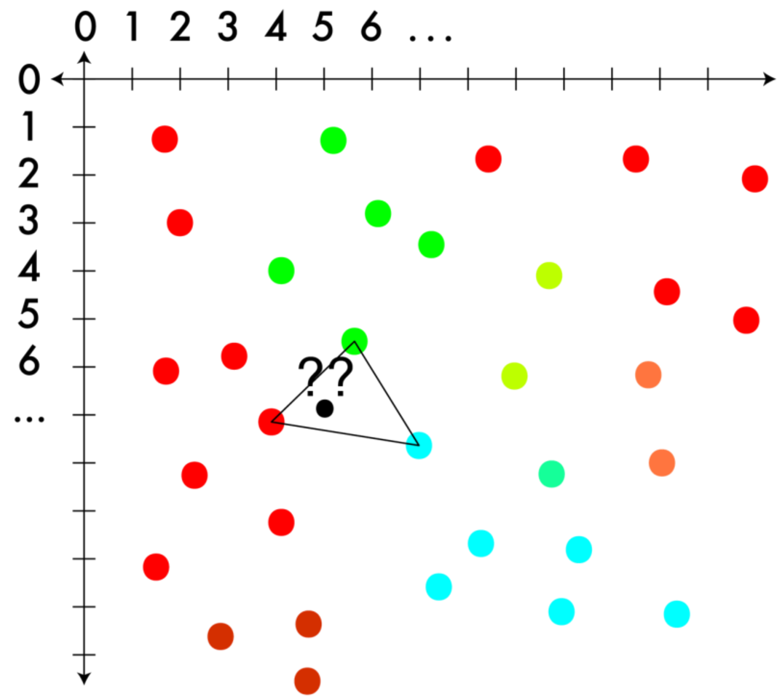
When you don't, you can look for triangles!



Triangle interpolation: for less structured image

Sometimes you have a regular grid, sometimes you don't.

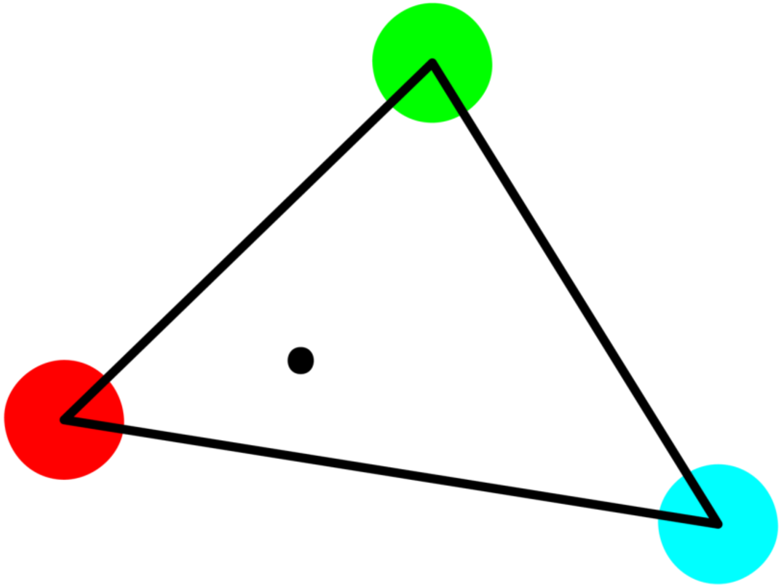
When you don't look for triangles!



Triangle interpolation: for less structured image

Sometimes you have a regular grid, sometimes you don't.

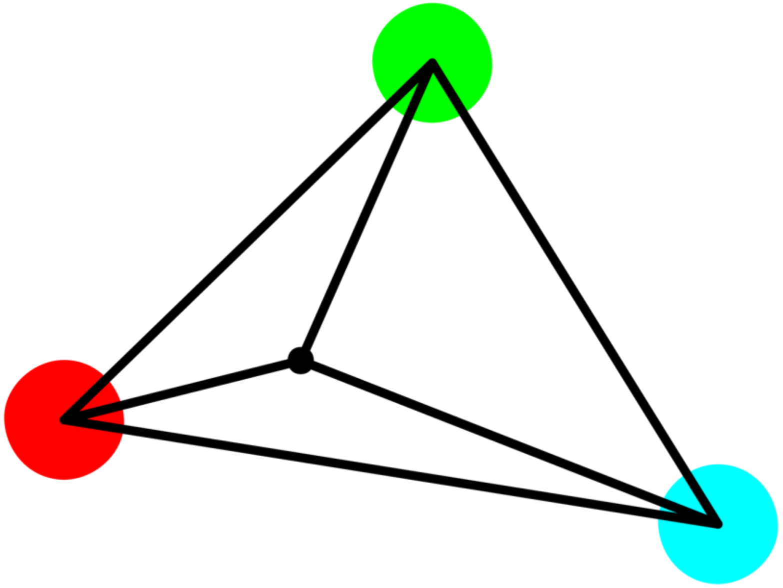
When you don't look for triangles!



Triangle interpolation: for less structured image

Sometimes you have a regular grid, sometimes you don't.

When you don't look for triangles!



Triangle interpolation: for less structured image

Weighted sum using triangles:

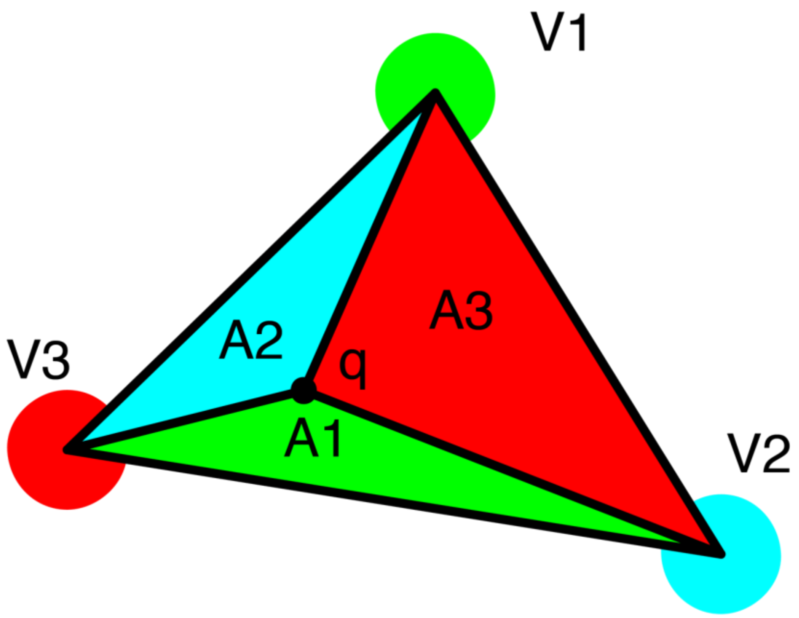
Q = V1*A1 + V2*A2 + V3*A3

WHY?

V1 is the furthest from q and A1 gives the smallest area.

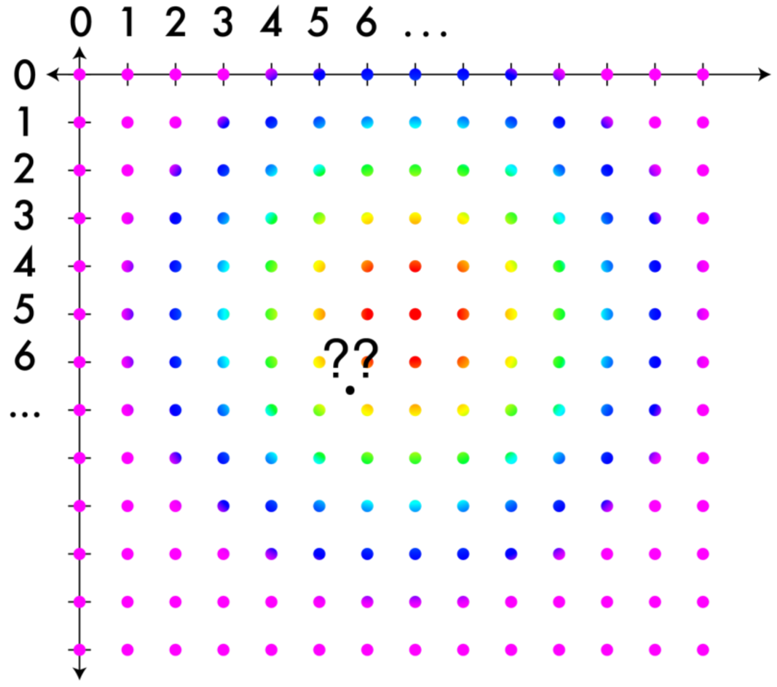
V2 is next furthest from 1 and A2 gives the next smallest area…

Should normalize this based on total area, but we won’t use this.



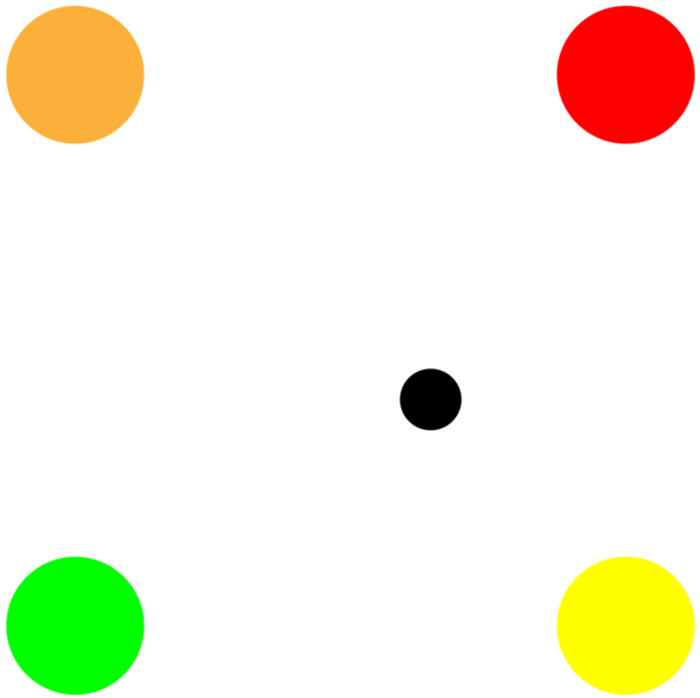
Bilinear interpolation: for grids, pretty good; easier than triangles

This time find the closest pixels in a box



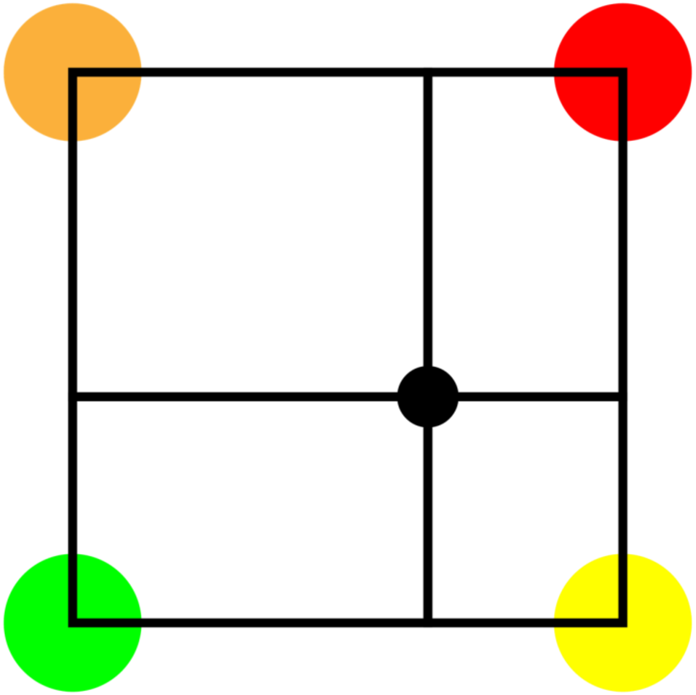
Bilinear interpolation: for grids, pretty good

This time find the closest pixels in a box



Bilinear interpolation: for grids, pretty good

This time find the closest pixels in a box

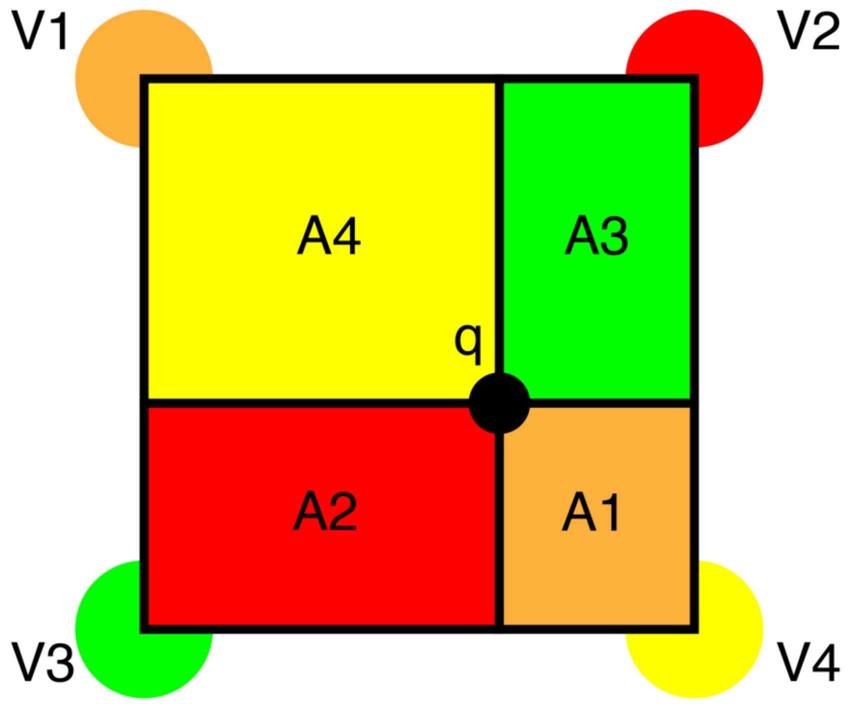


Bilinear interpolation: for grids, pretty good

This time find the closest pixels in a box

Same plan, weighted sum based on area of opposite rectangle

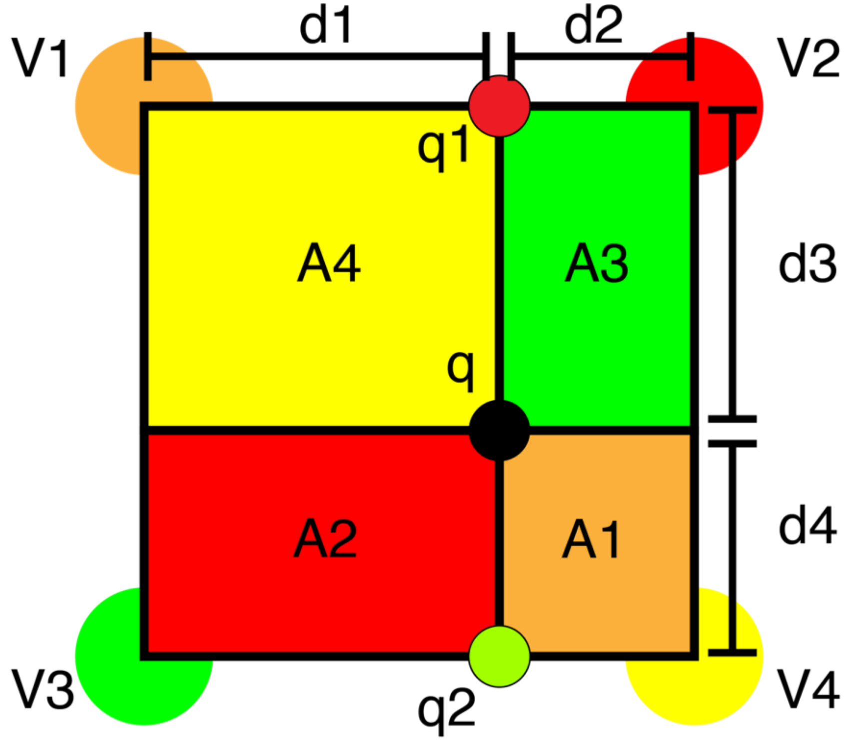
Q = V1*A1 + V2*A2 + V3*A3 + V4*A4



Bilinear interpolation: for grids, pretty good

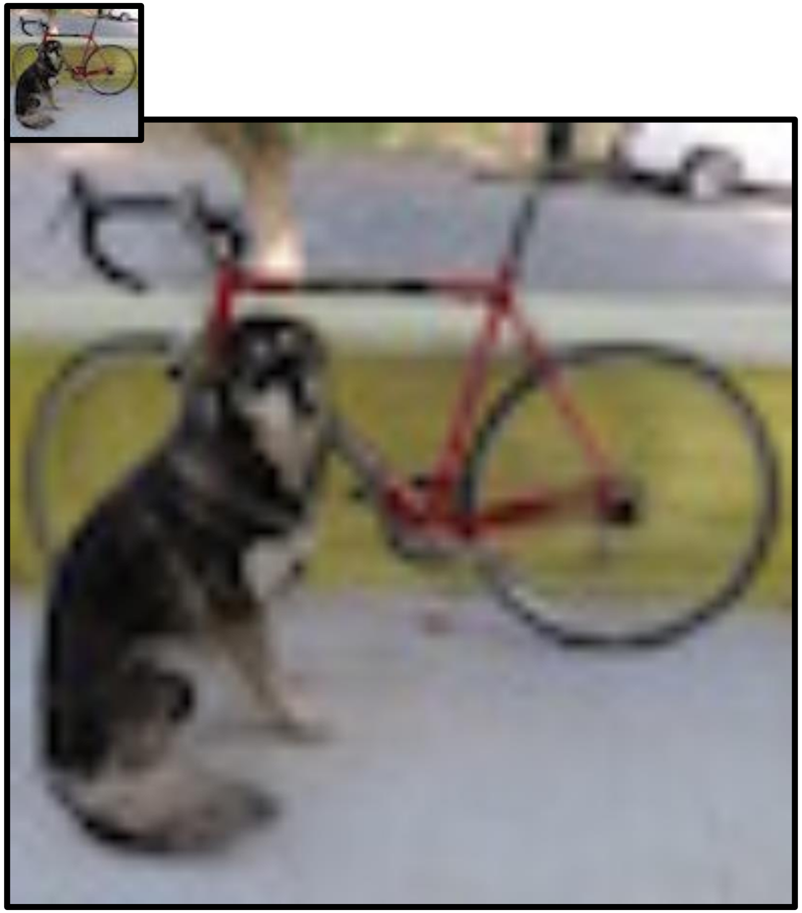
A1 = d2*d4
A2 = d1*d4
A3 = d2*d3
A4 = d1*d3

q = V1*A1 + V2*A2 + V3*A3 + V4*A4



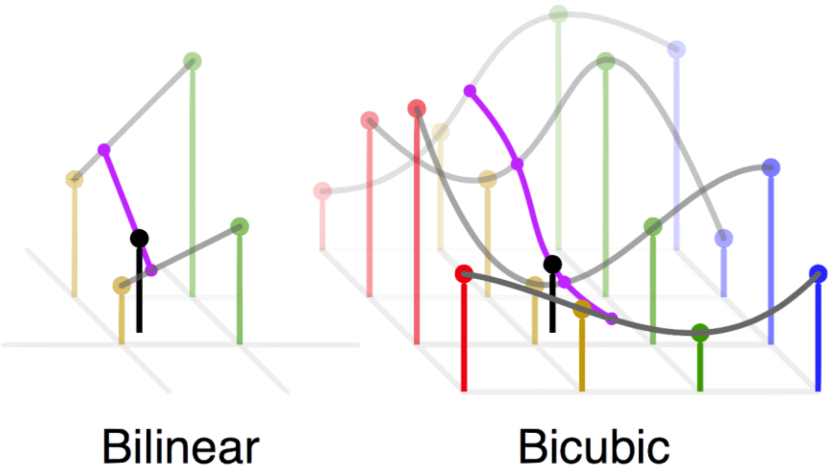
Bilinear interpolation: for grids, pretty good

- Smoother than NN
- More complex
  - 4 lookups
  - Some math
- Often the right tradeoff of speed vs final result

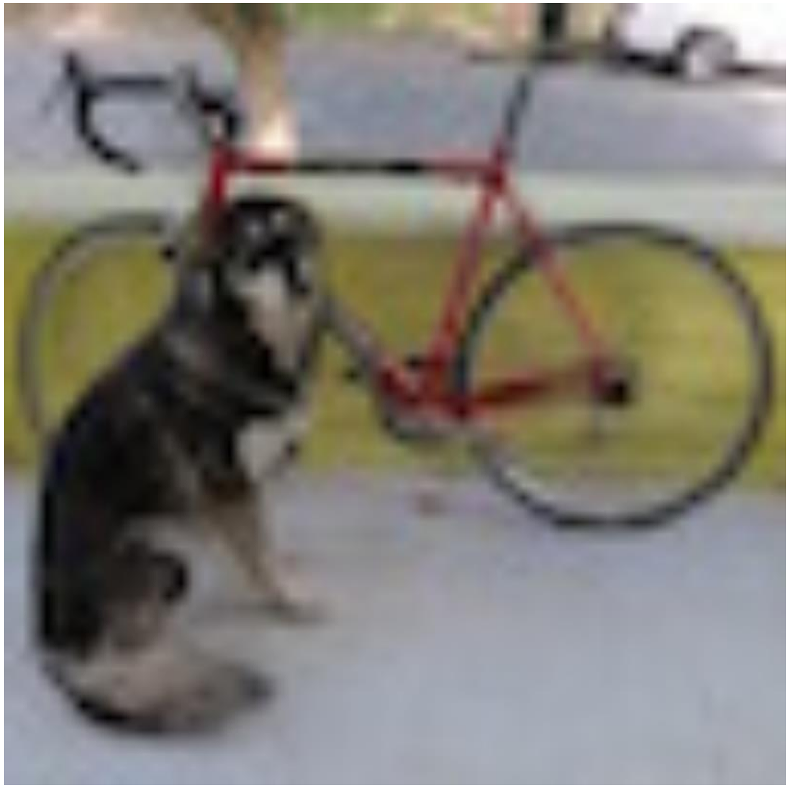


Bicubic sampling: more complex, maybe better?

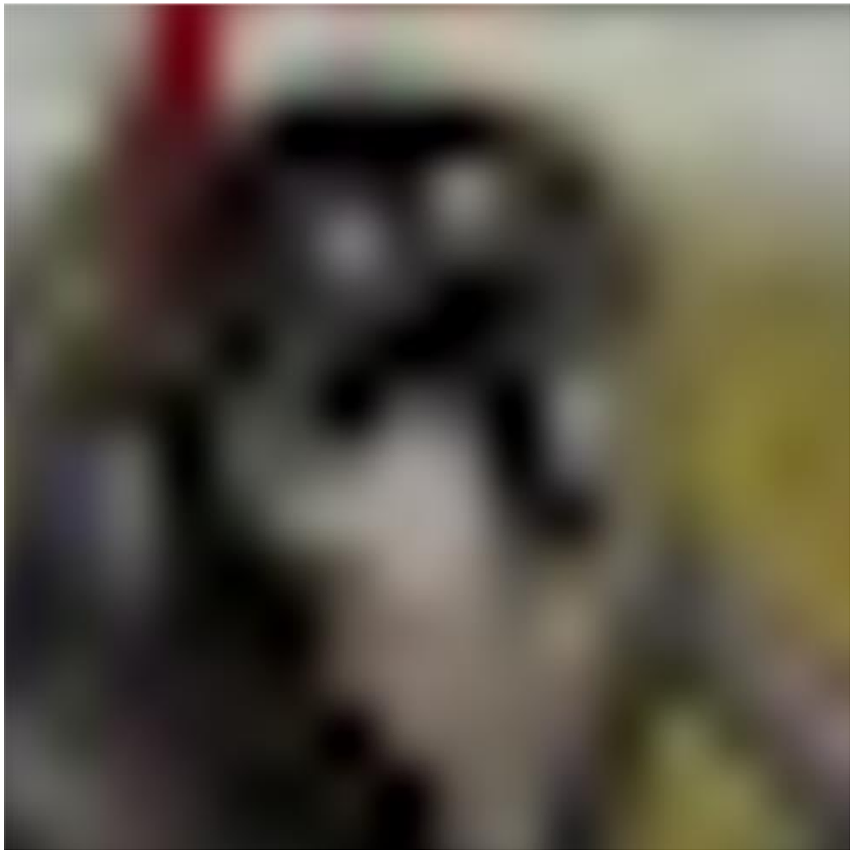
- A cubic interpolation of 4 cubic interpolations
- Smoother than bilinear, no "star"
- 16 nearest neighbors
- Fit 3rd order poly:
  - f(x) = a + bx + cx^2 + dx^3
- Interpolate along axis
- Fit another poly to interpolated values

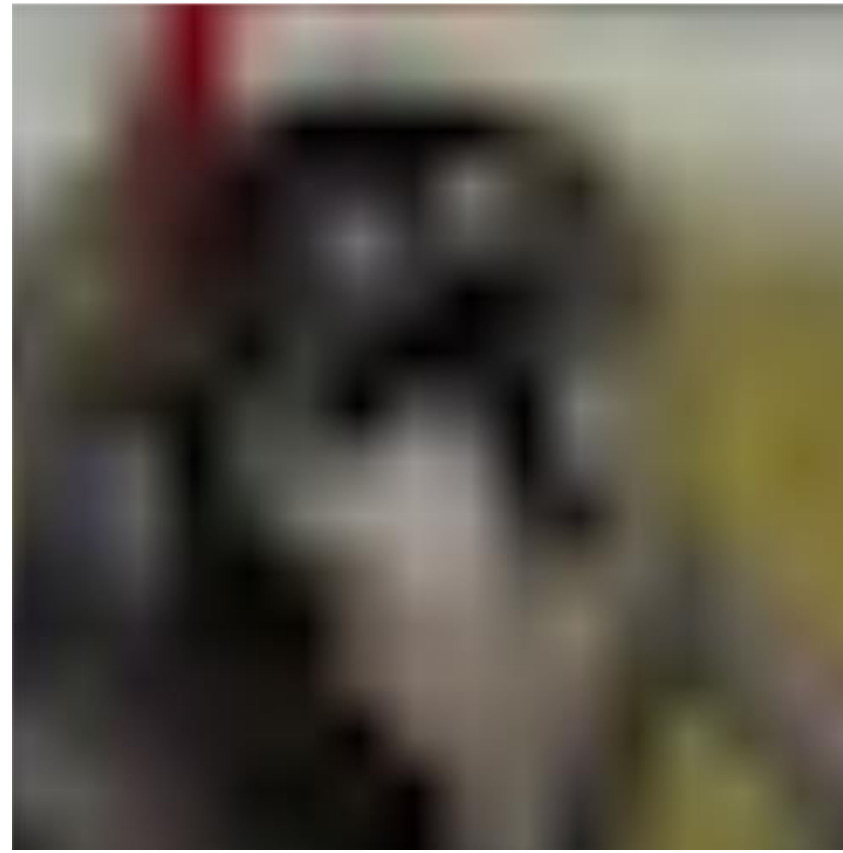


Bicubic vs bilinear



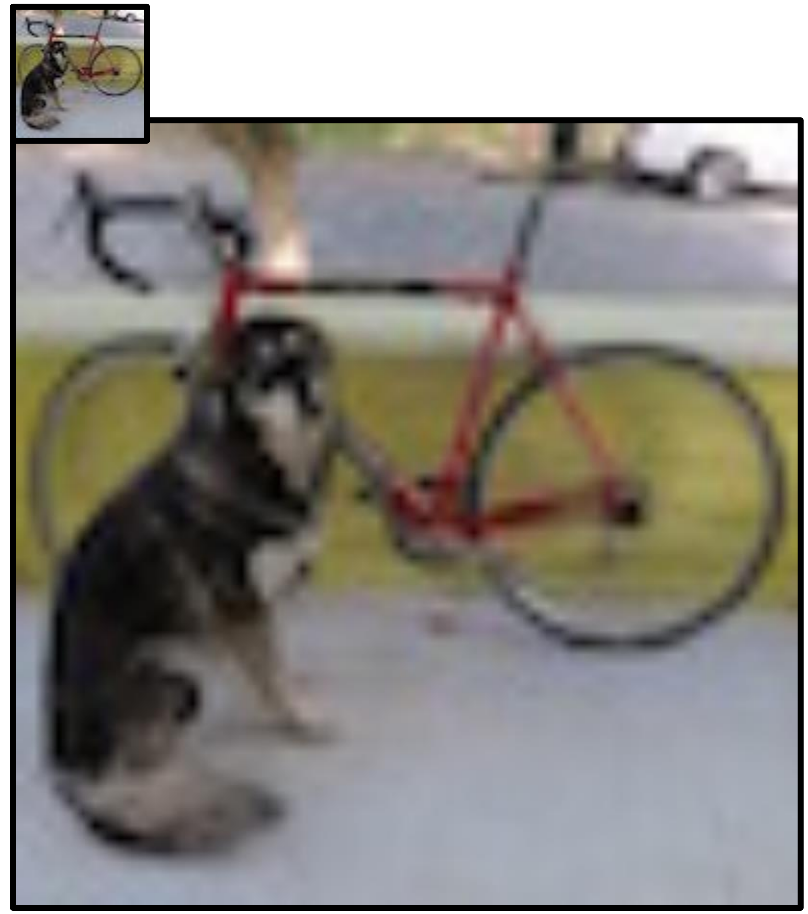
Bicubic vs bilinear





Resize algorithm:

- For each pixel in new image:
  1. Map to old im coordinates
  2. Interpolate value
  3. Set new value in image



What about shrinking?

- NN and Bilinear only look at small area
- Lots of artifacting
- Staircase pattern on diagonal lines
- We'll fix this next class with filters!

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…

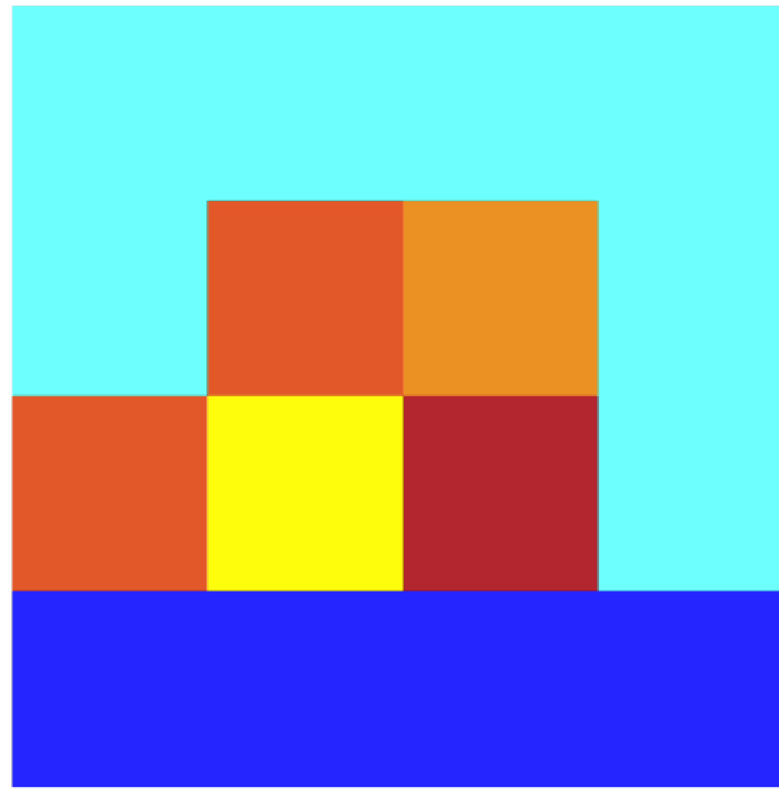
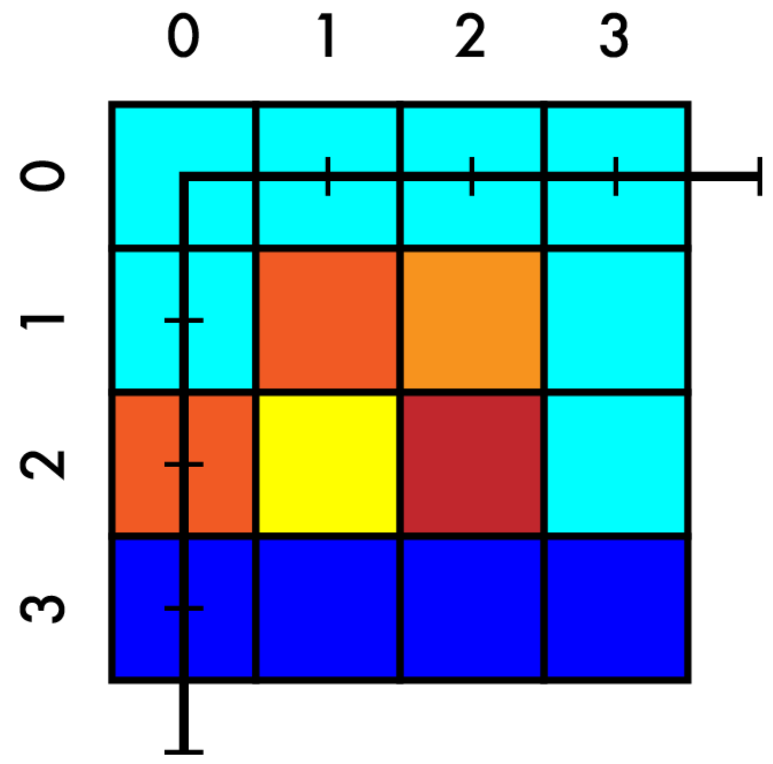


Image resizing!

Say we want to increase the size of an image…

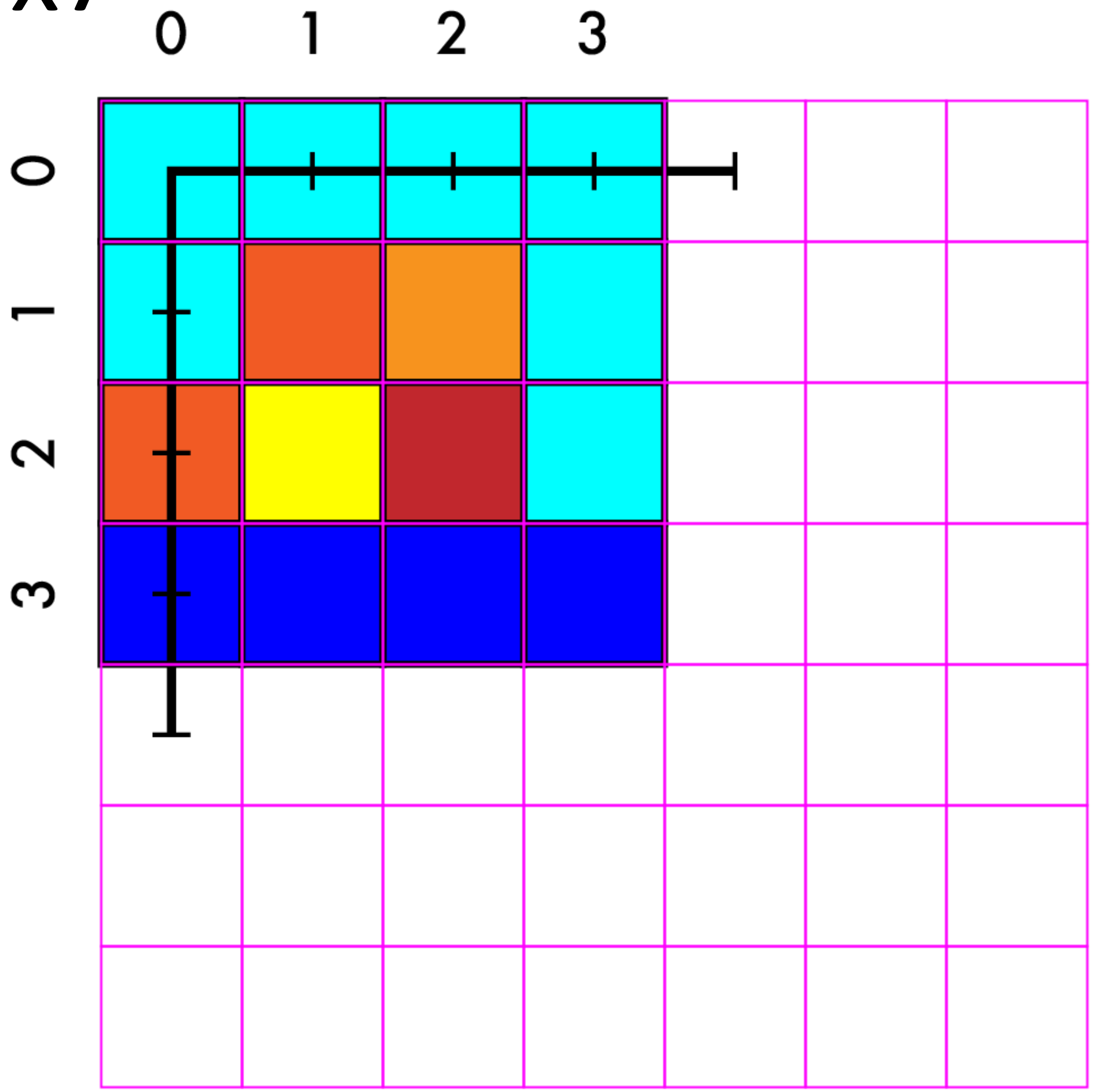
This is a beautiful image of a sunset… it’s just very small…

Say we want to increase size 4x4 -> 7x7



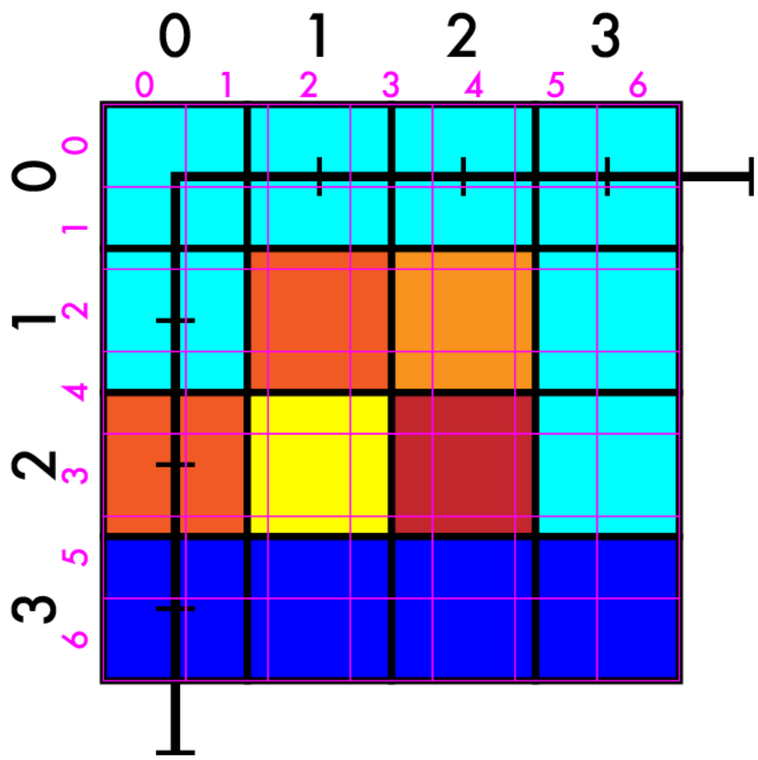
Resize 4x4 -> 7x7

- Create our new image



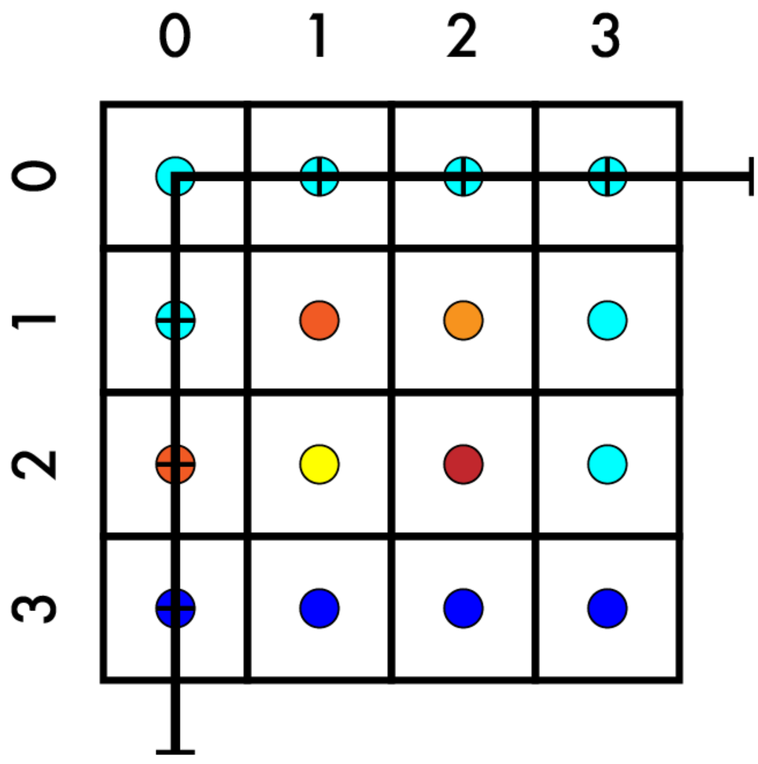
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates



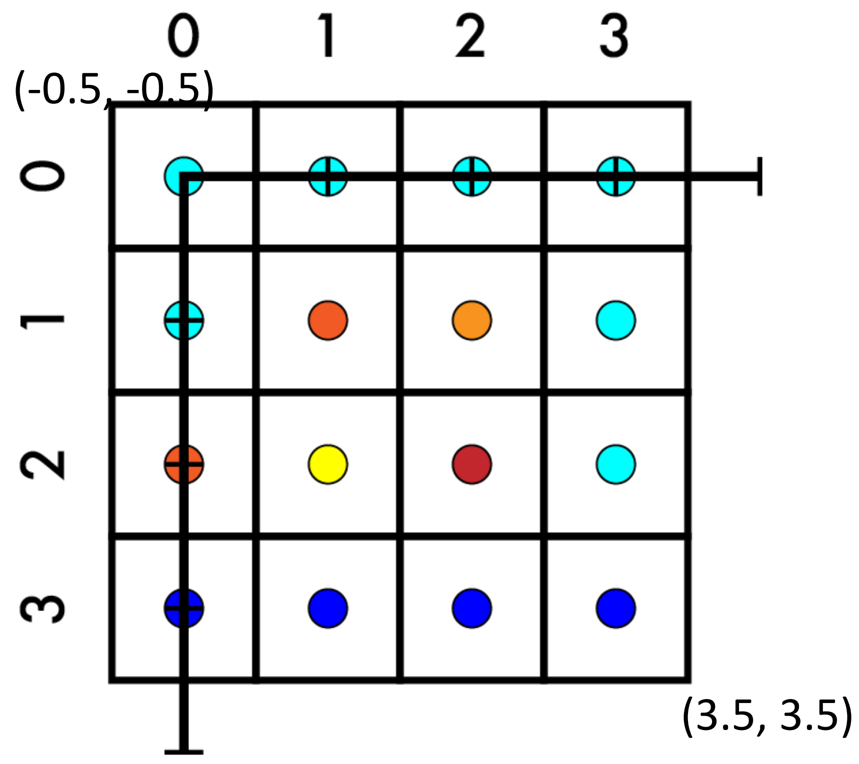
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates



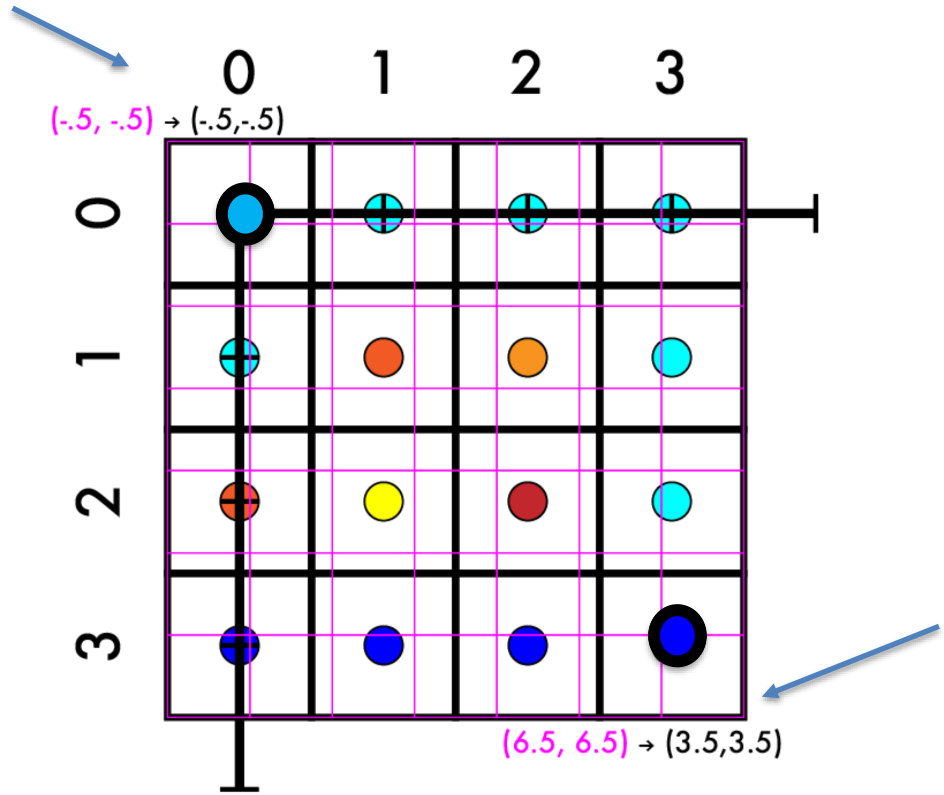
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates



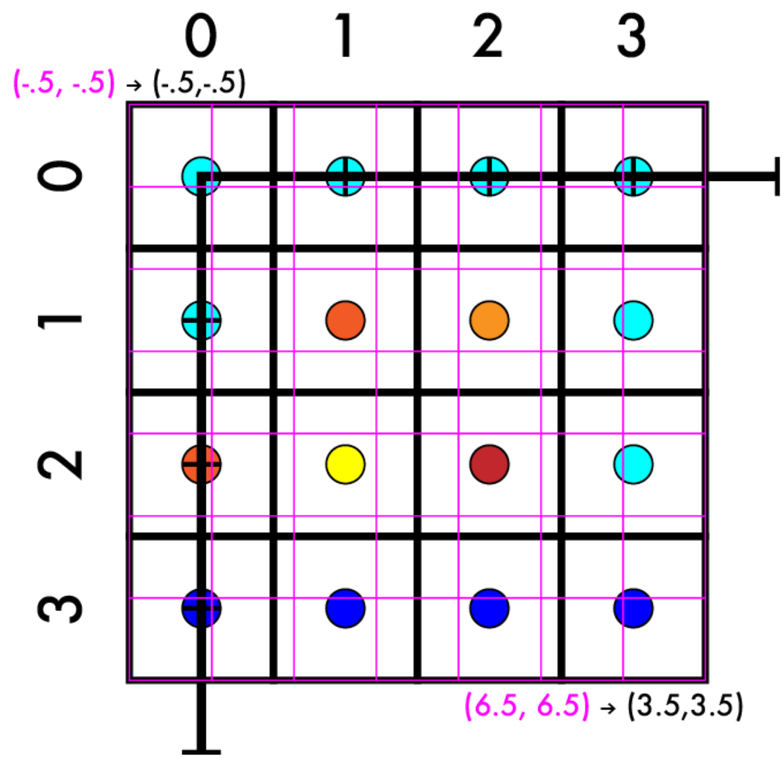
Resize 4x4 -> 7x7

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



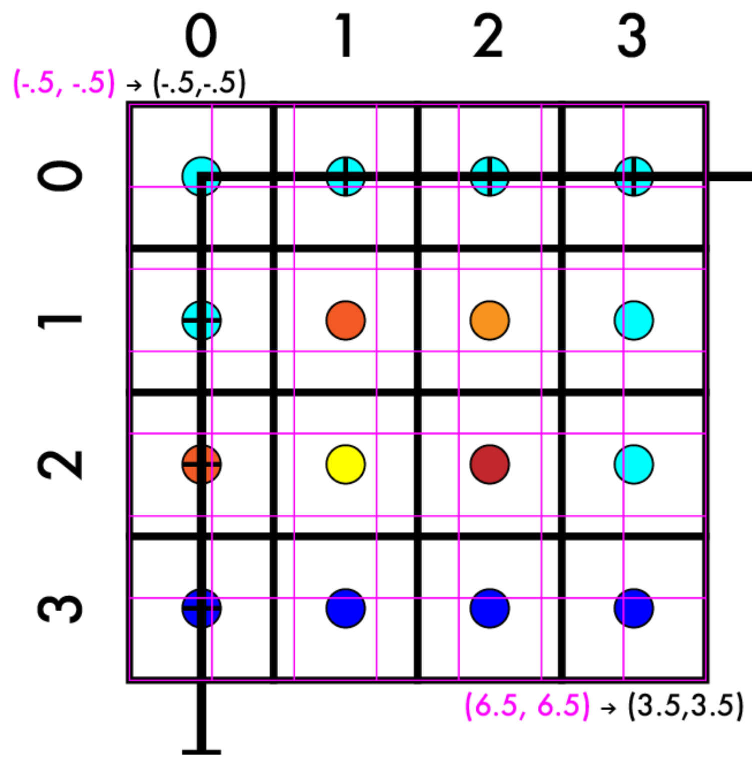
Resize 4x4 -> 7x7

- 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



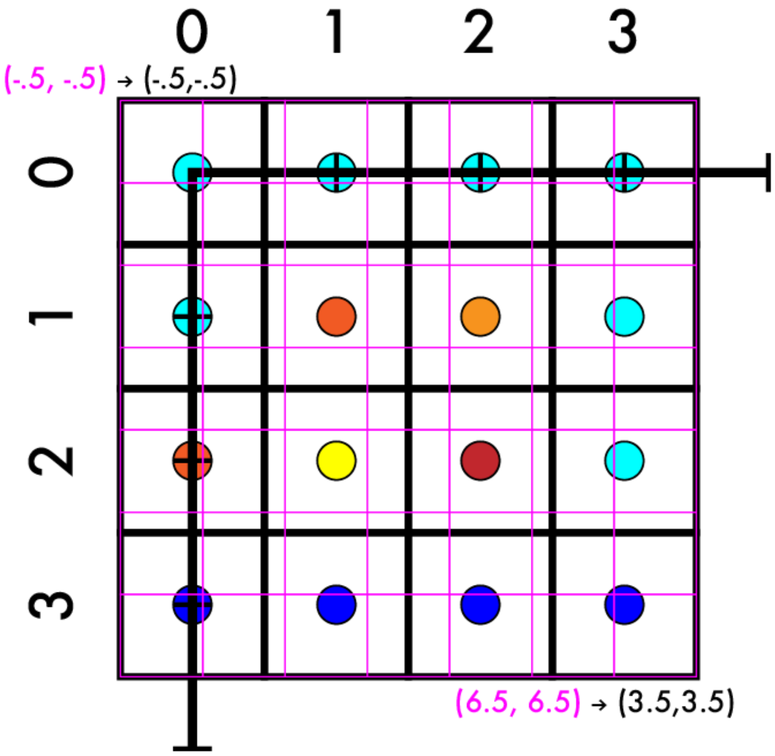
Resize 4x4 -> 7x7

- 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



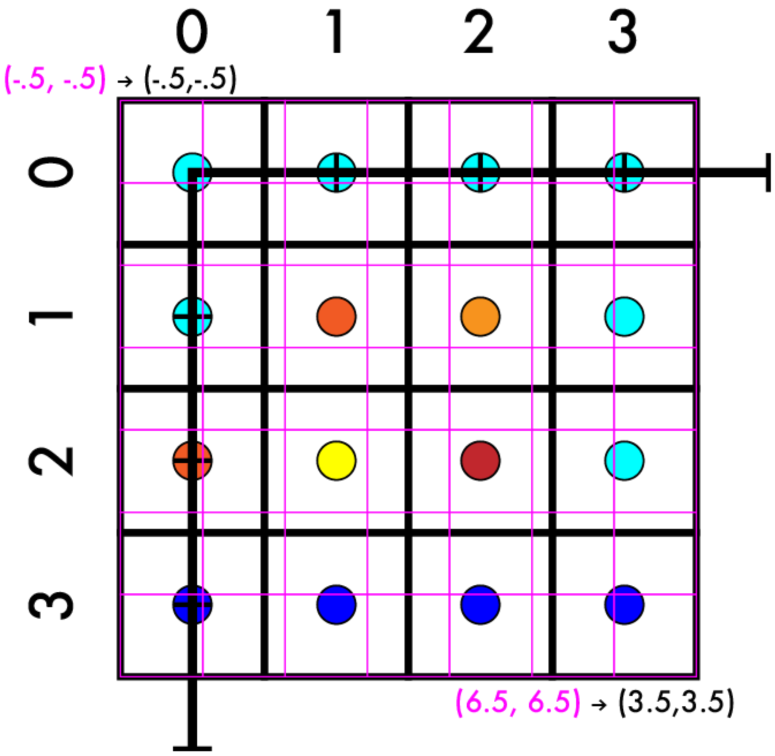
Resize 4x4 -> 7x7

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



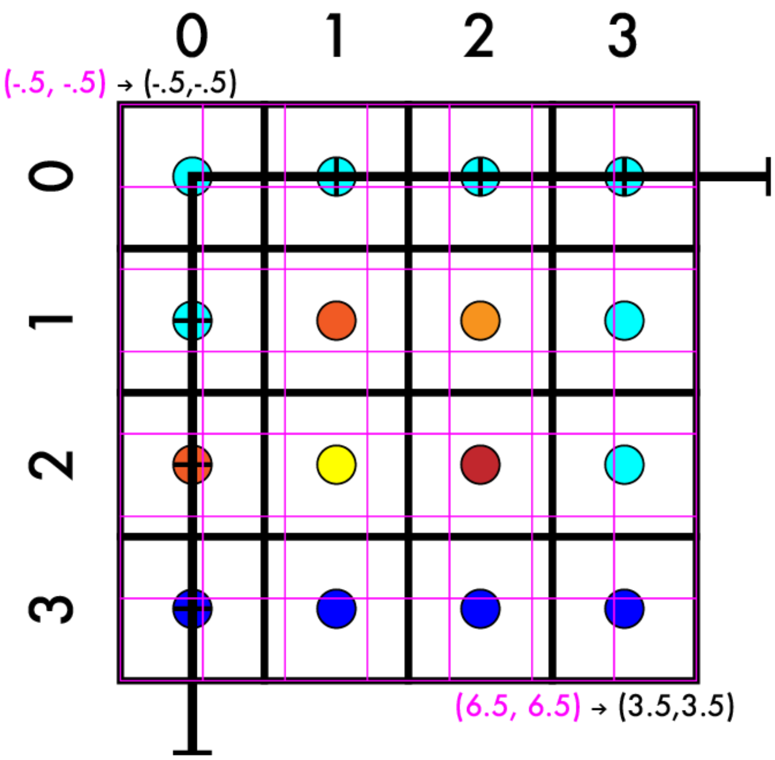
Resize 4x4 -> 7x7

- 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



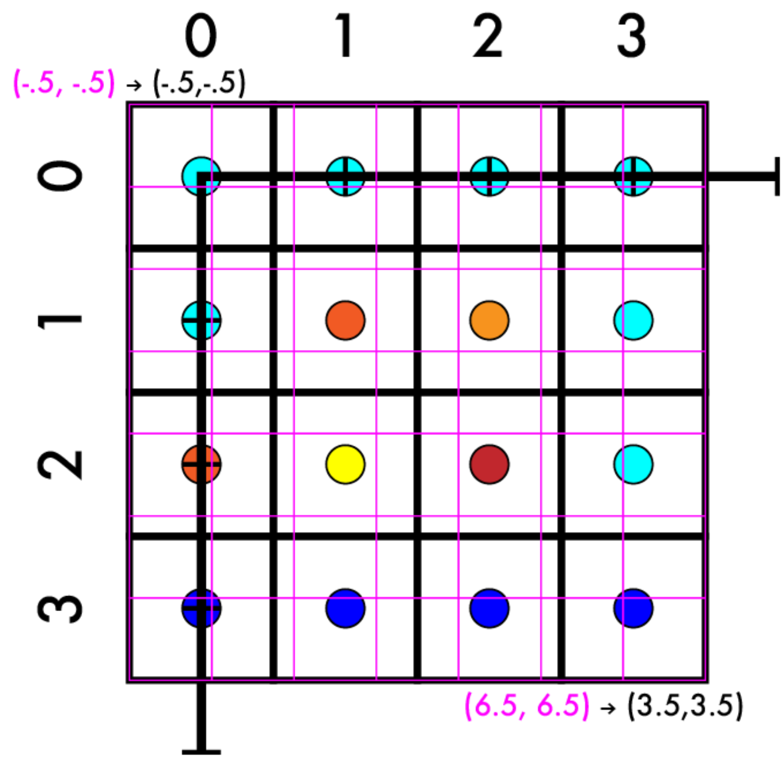
Resize 4x4 -> 7x7

- 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



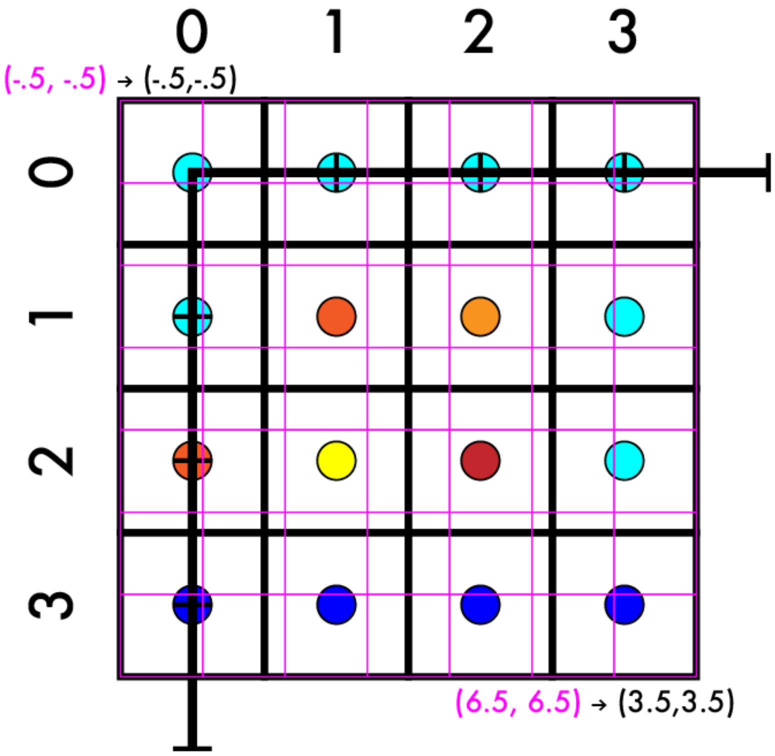
Resize 4x4 -> 7x7

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



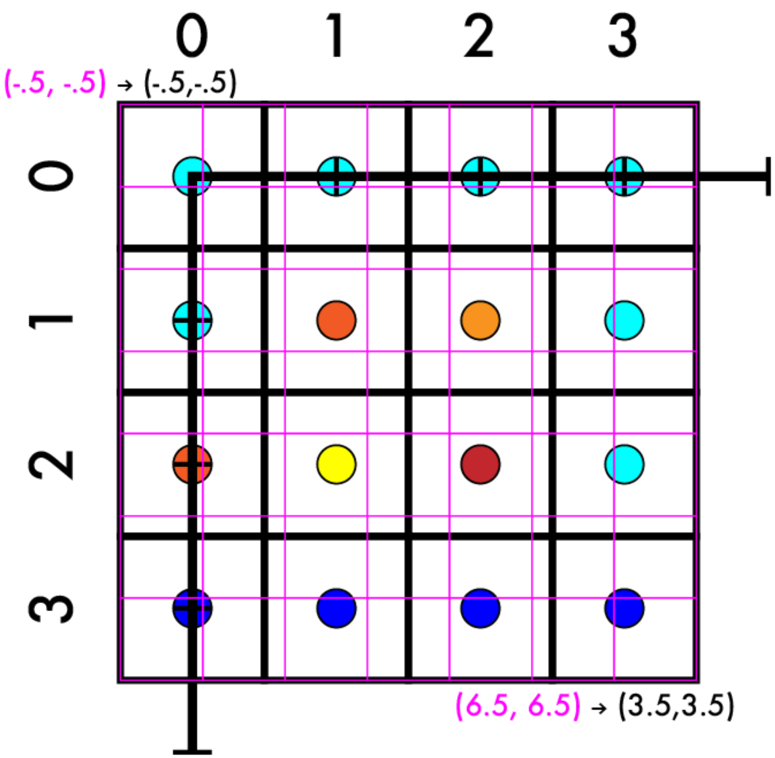
Resize 4x4 -> 7x7

- 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



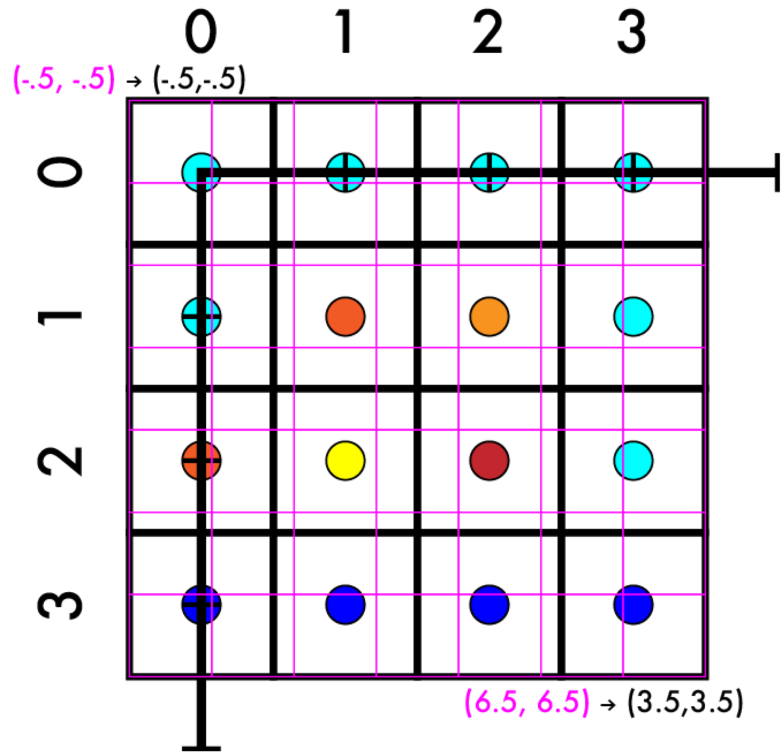
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - System of equations
  - $aX + b = Y$
  - $a*-.5 + b = -.5$
  - $a*6.5 + b = 3.5$
  - $a = 4/7$
  - $b = -3/14$
- So, we can start with any coordinate X of the big (new) image and use a and b to get Y on the smaller (old) image.



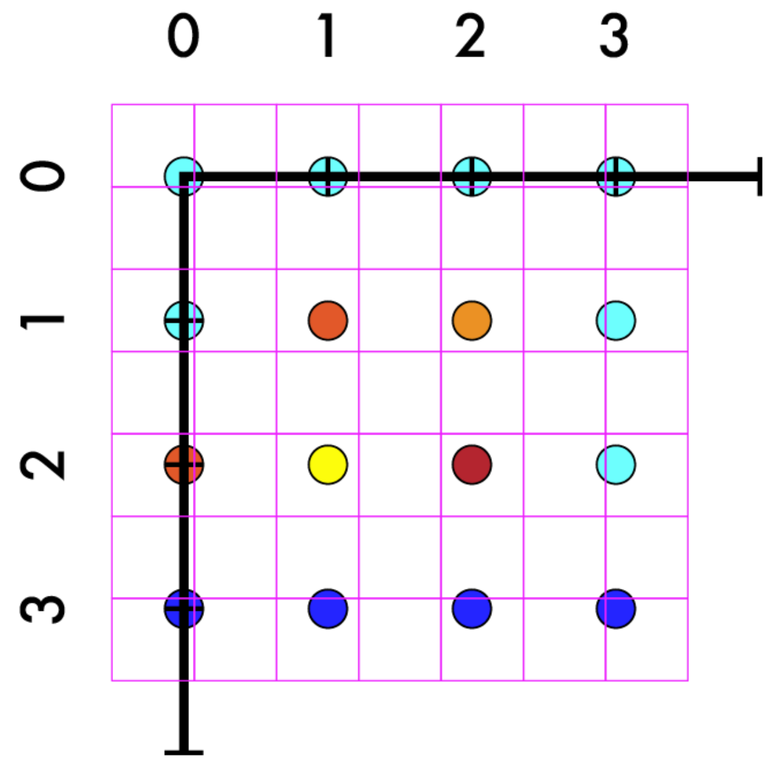
Resize 4x4 -> 7x7

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



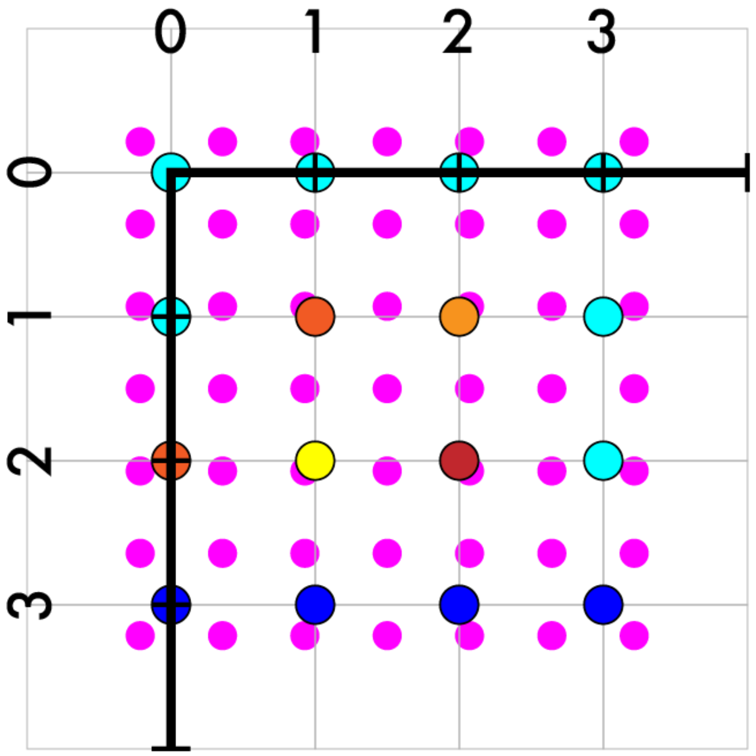
Resize 4x4 -> 7x7

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



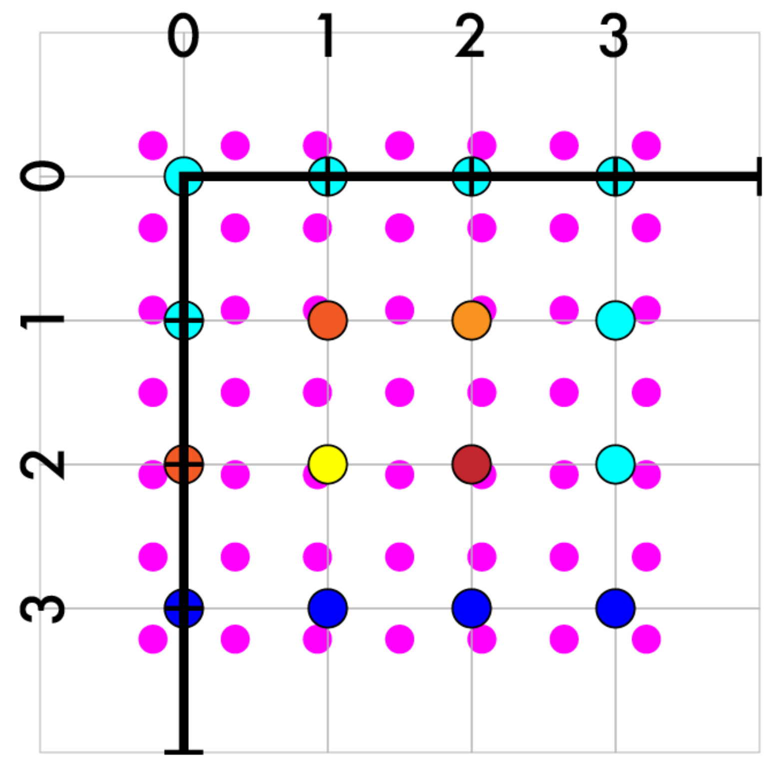
Resize 4x4 -> 7x7

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



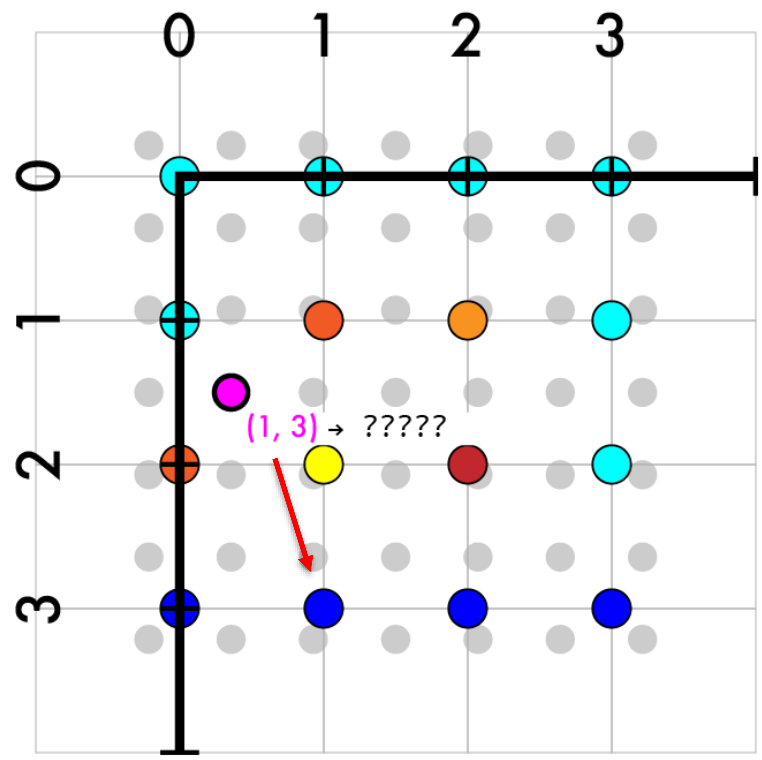
Resize 4x4 -> 7x7

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



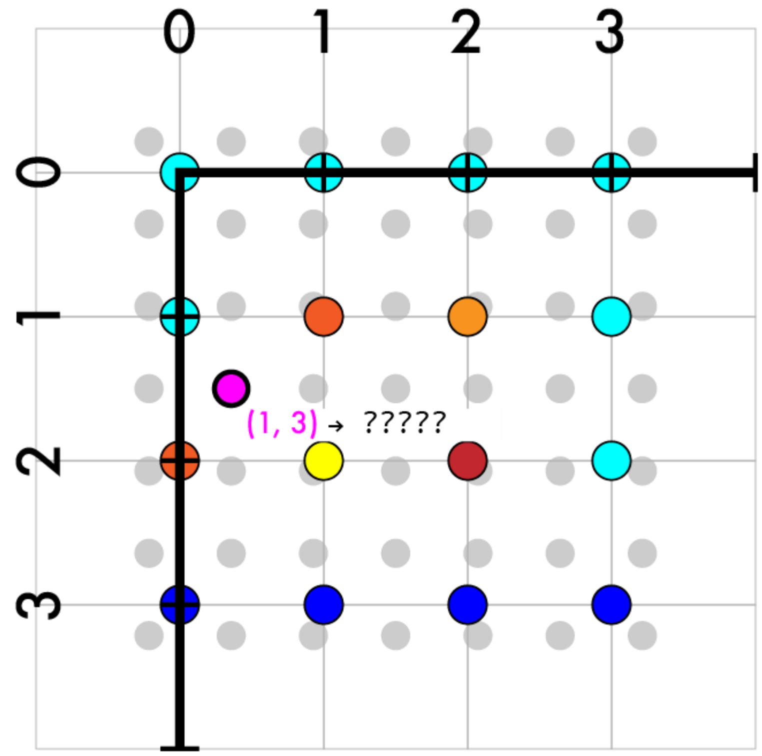
Resize 4x4 -> 7x7

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



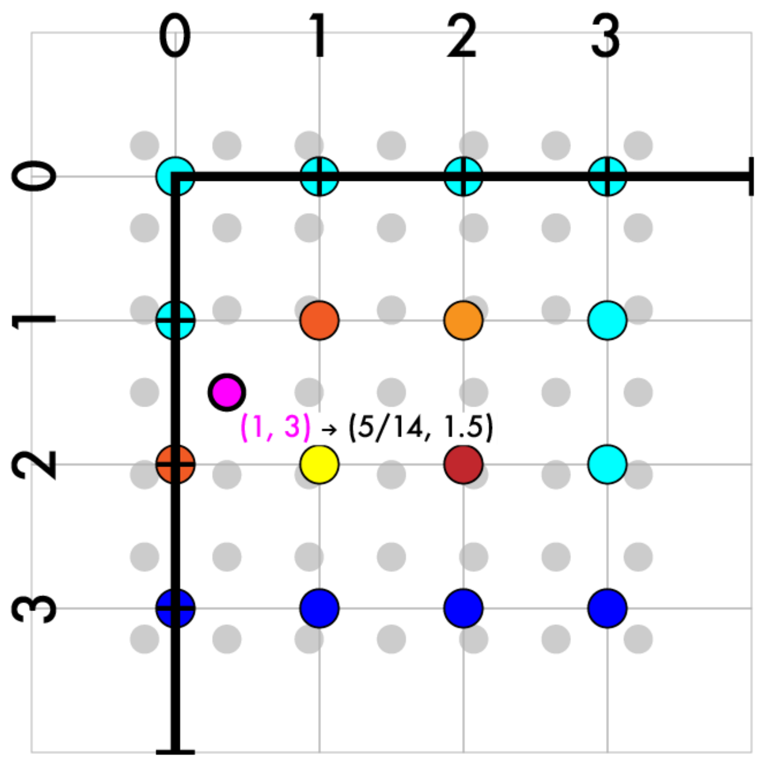
Resize 4x4 -> 7x7

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



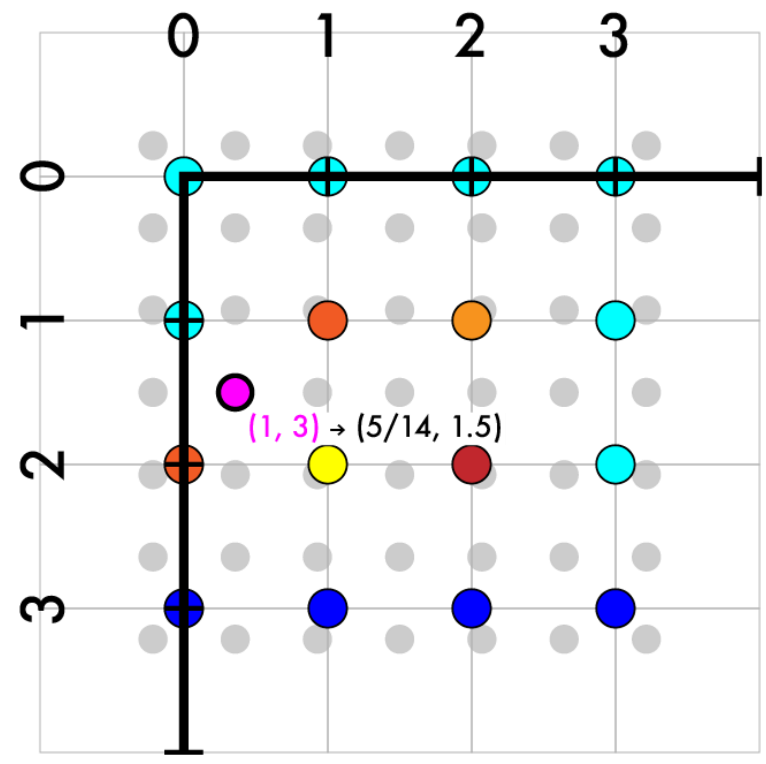
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3)
  - 4/7*1 - 3/14
  - 4/7*3 - 3/14
  - (5/14, 21/14)



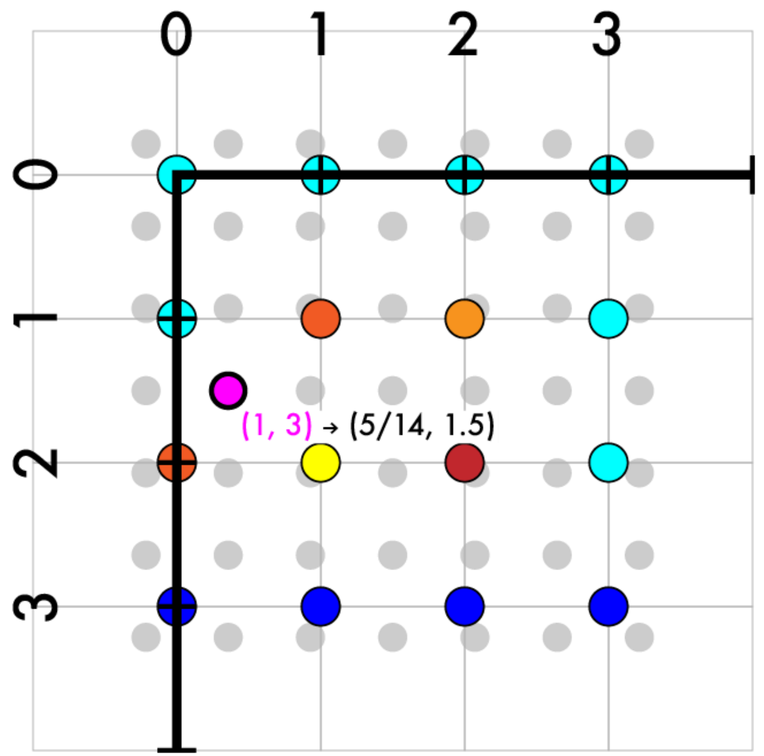
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 21/14)



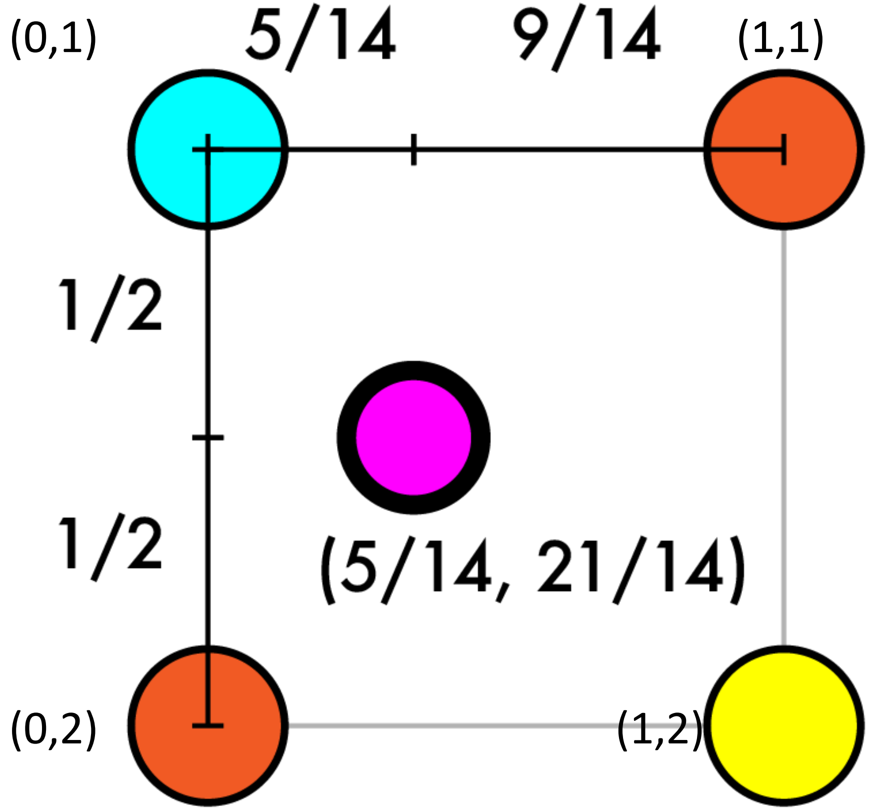
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 21/14)
  - Interpolate old values



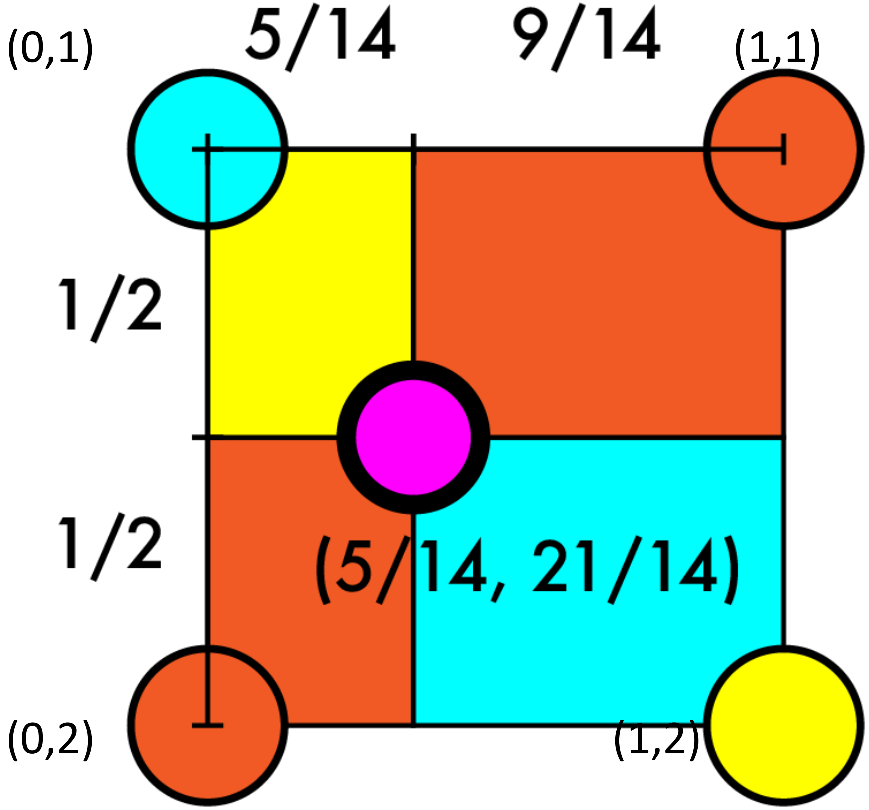
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 21/14)
  - Interpolate old values



Resize 4x4 -> 7x7

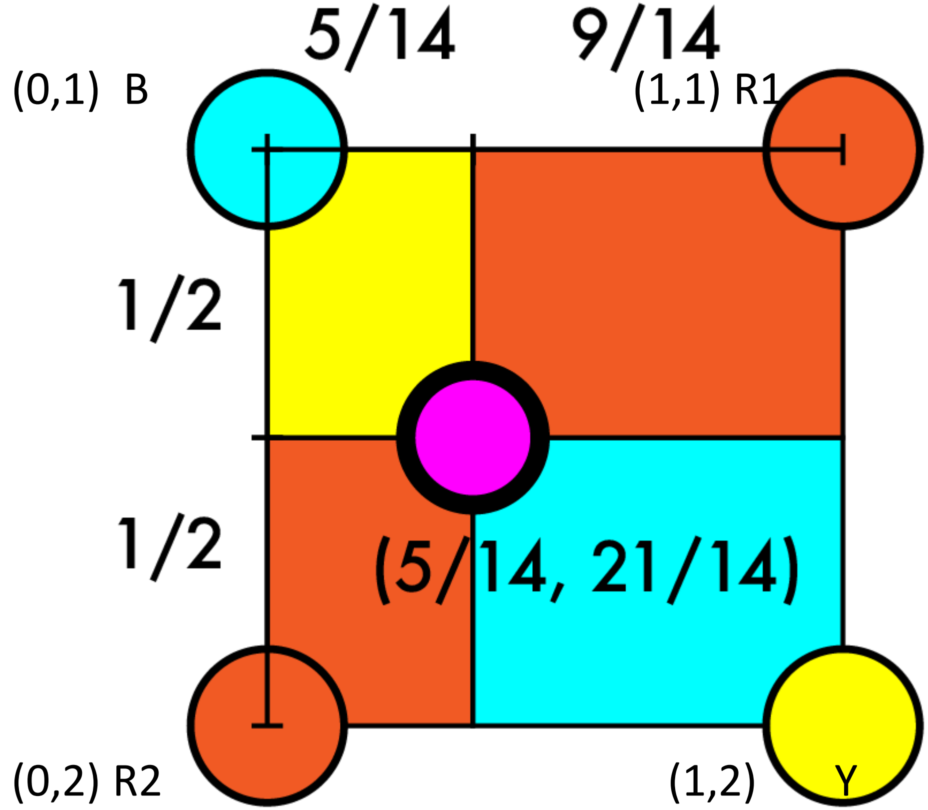
- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 21/14)
  - Interpolate old values
    - Size of opposite rects



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) -> (5/14, 21/14)
  - Interpolate old values
    - Yar = (1/2)(5/14)
    - Bar = (1/2)(9/14)
    - R1ar = (1/2)(5/14)
    - R2ar = (1/2)(9/14)

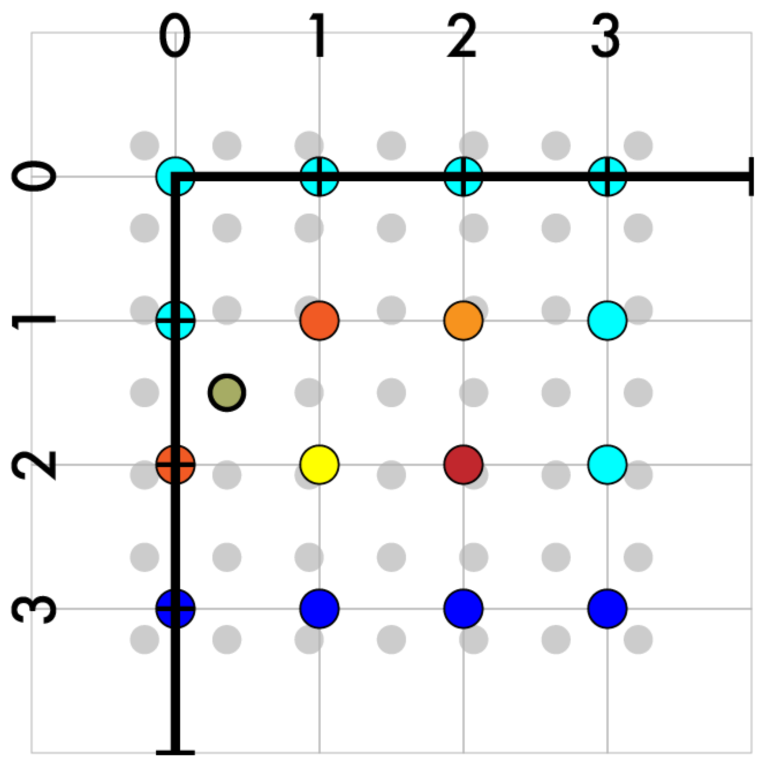
V = Yval*Yar+Bval*Var+R1val*R1ar+R2val*R2ar



- For each channel c, put the interpolated value from that channel in position (1,3,c).

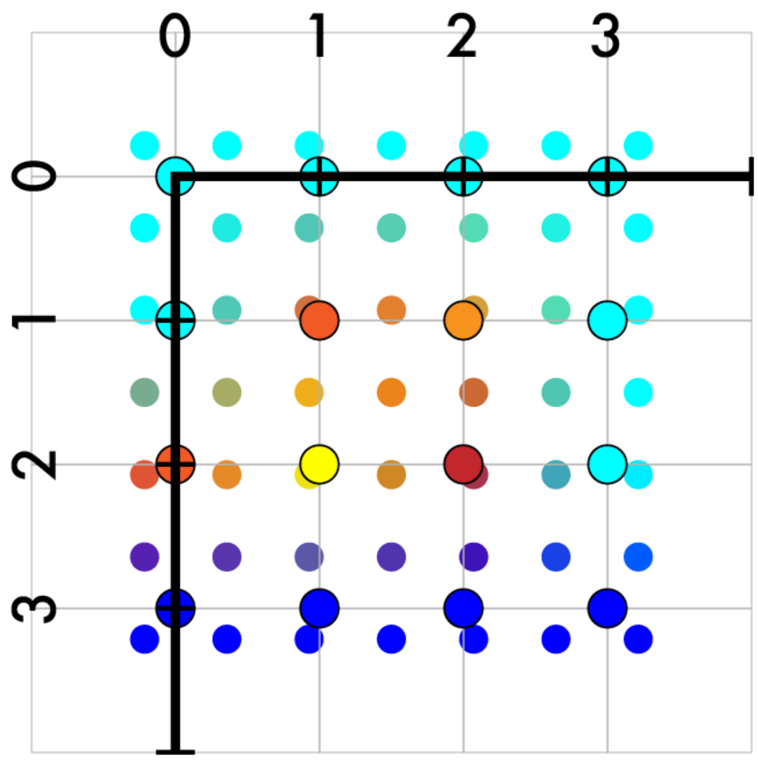
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
    - 4/7 X - 3/14 = Y
- Iterate over new pts
    - Map to old coords
    - (1, 3) -> (5/14, 21/14)
    - Interpolate old values



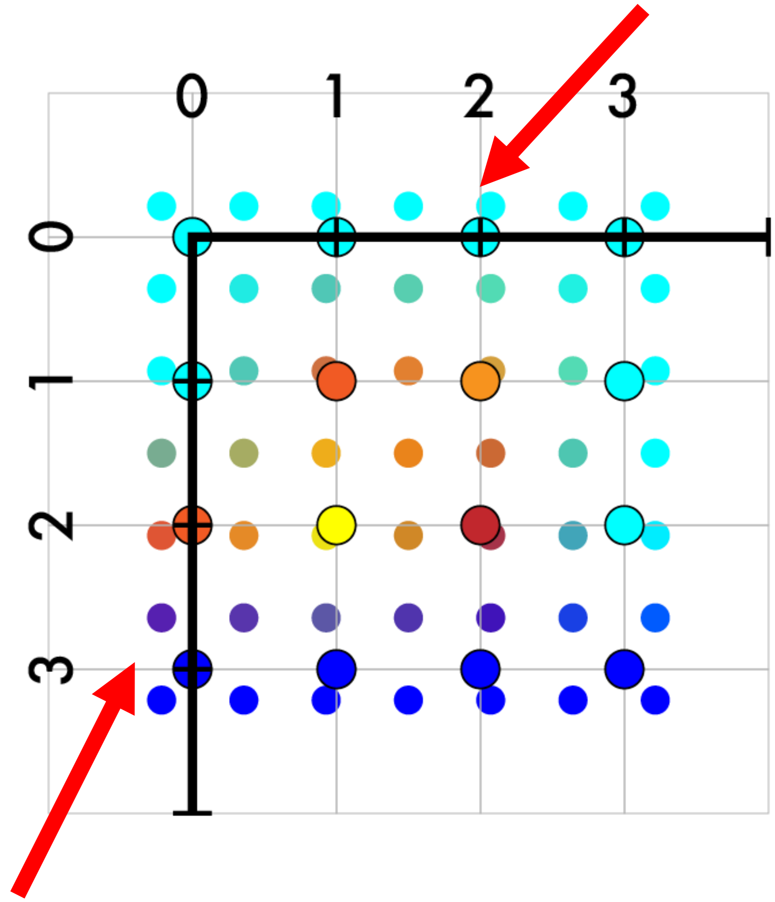
Resize 4x4 -> 7x7

- Create our new image results
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 21/14)
  - Interpolate old values



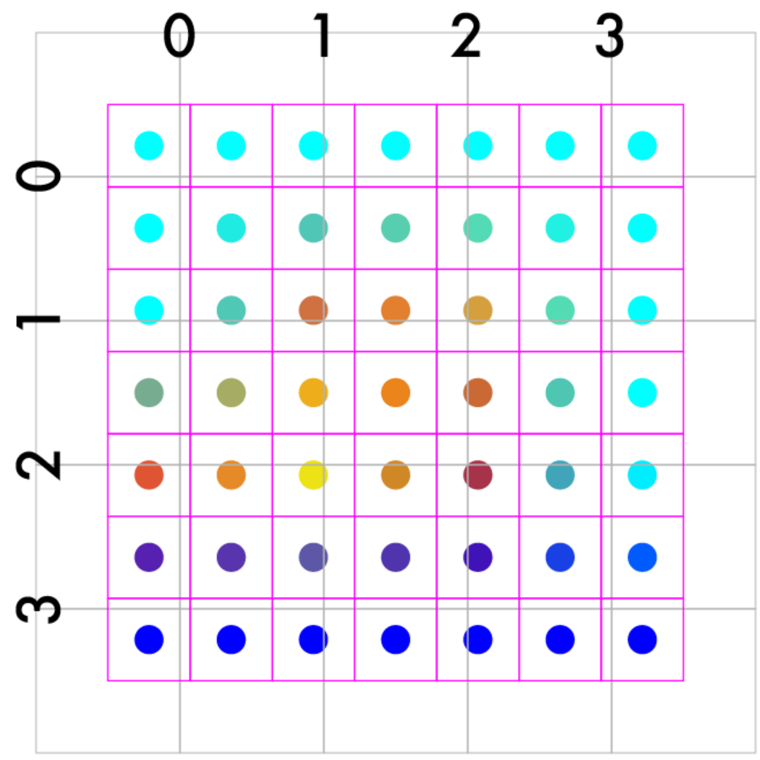
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 21/14)
  - Interpolate old values
- Fill in the rest
  - On outer edges use padding!

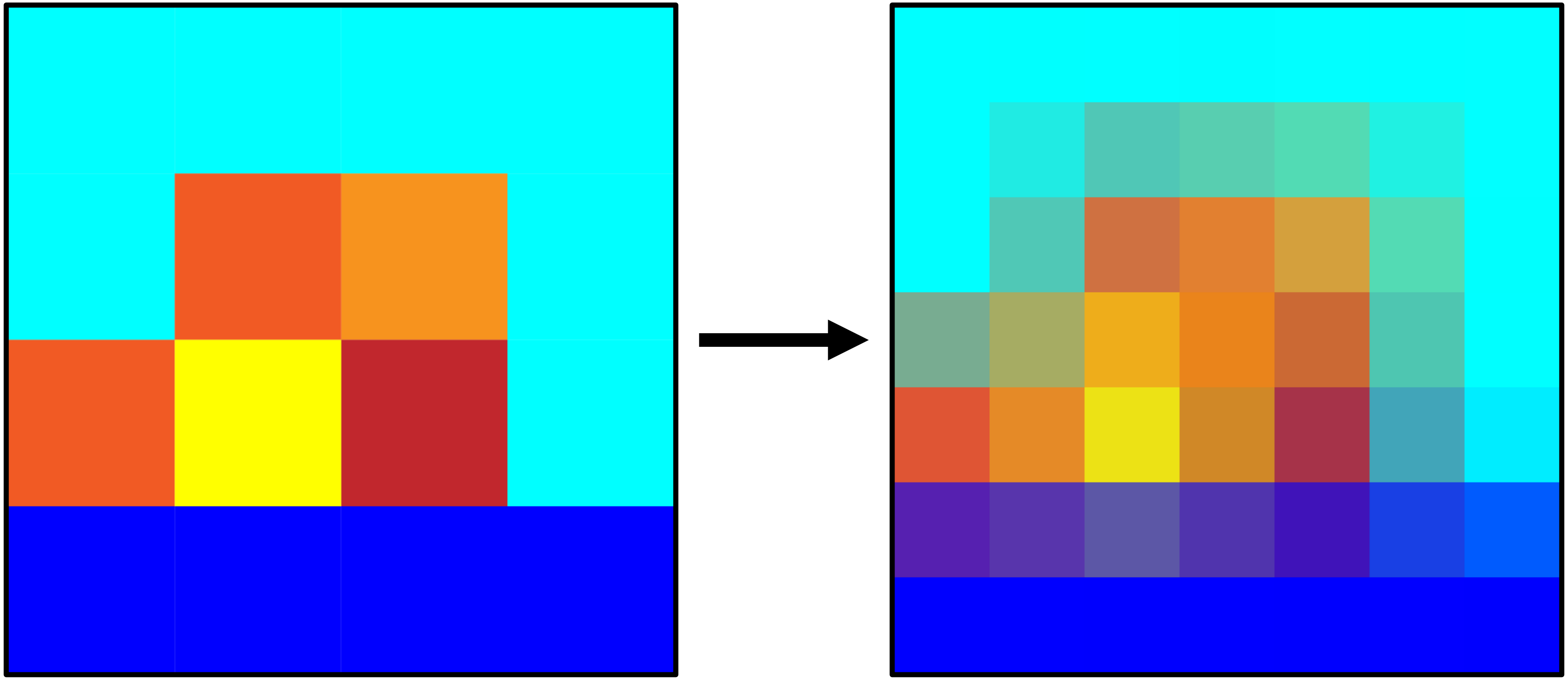


Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 21/14)
  - Interpolate old values
- Final result 7 x 7



We did it!

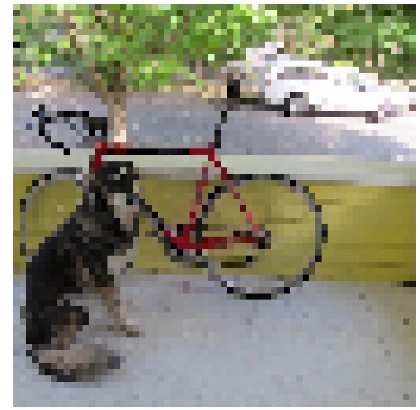


Let’s do something interesting already!!

Want to make image smaller

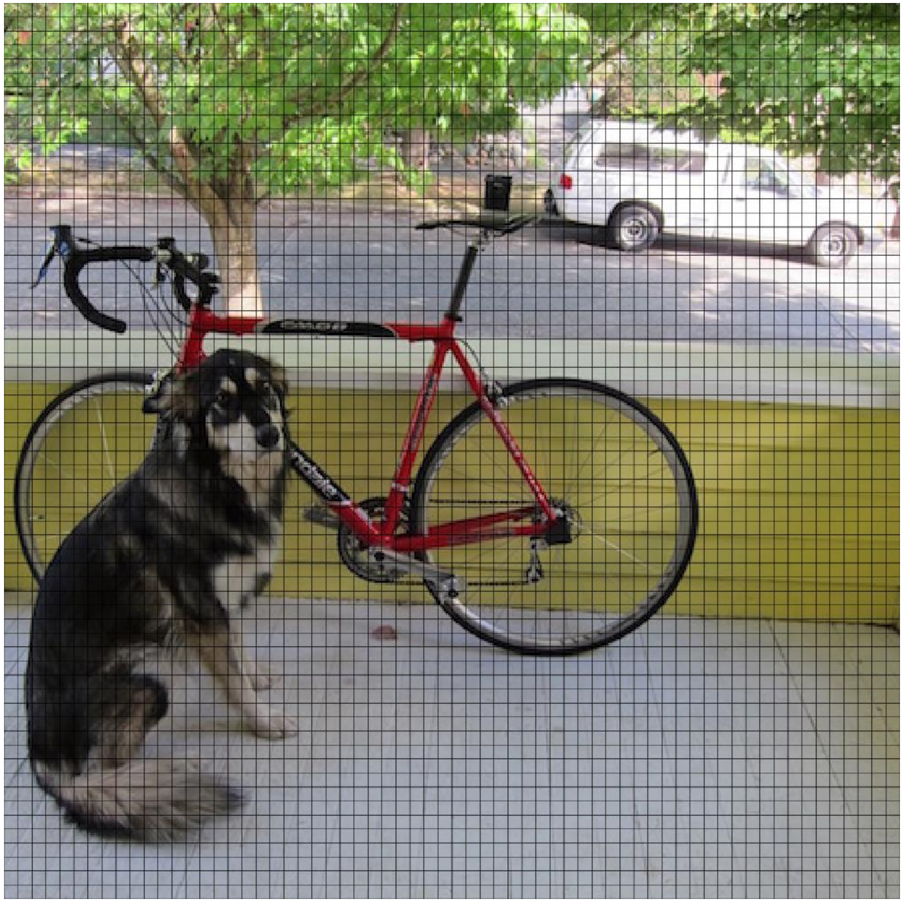
448x448 -> 64x64

448x448 -> 64x64

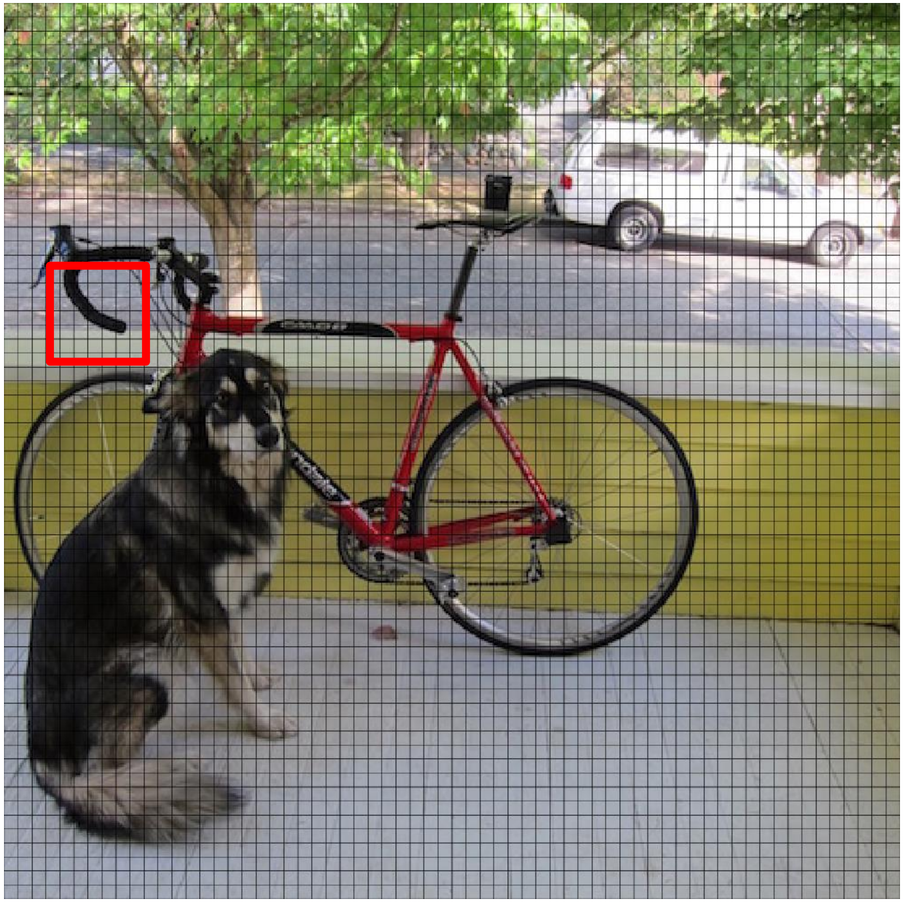


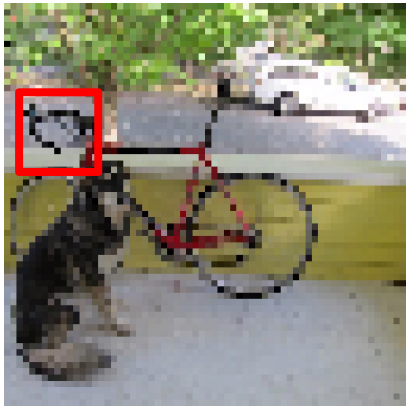
448x448 -> 64x64

448x448 -> 64x64

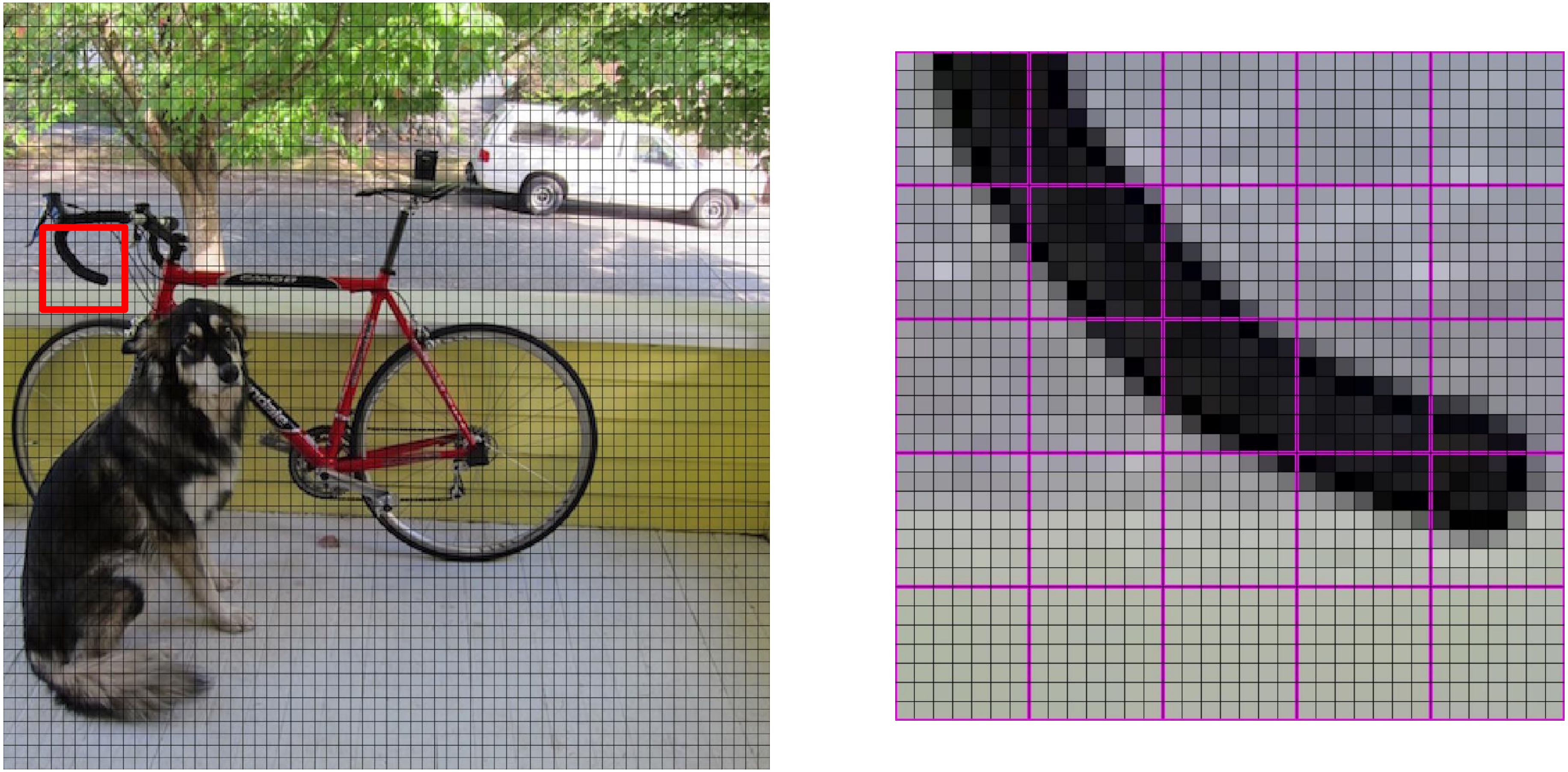


448x448 -> 64x64

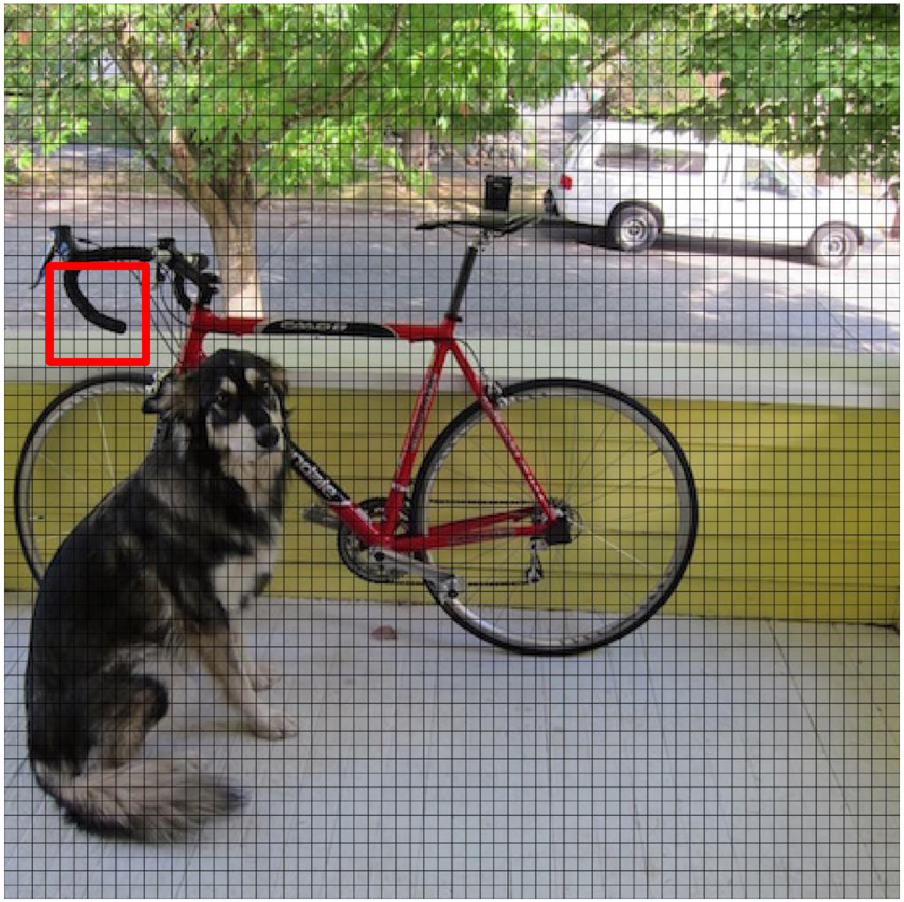


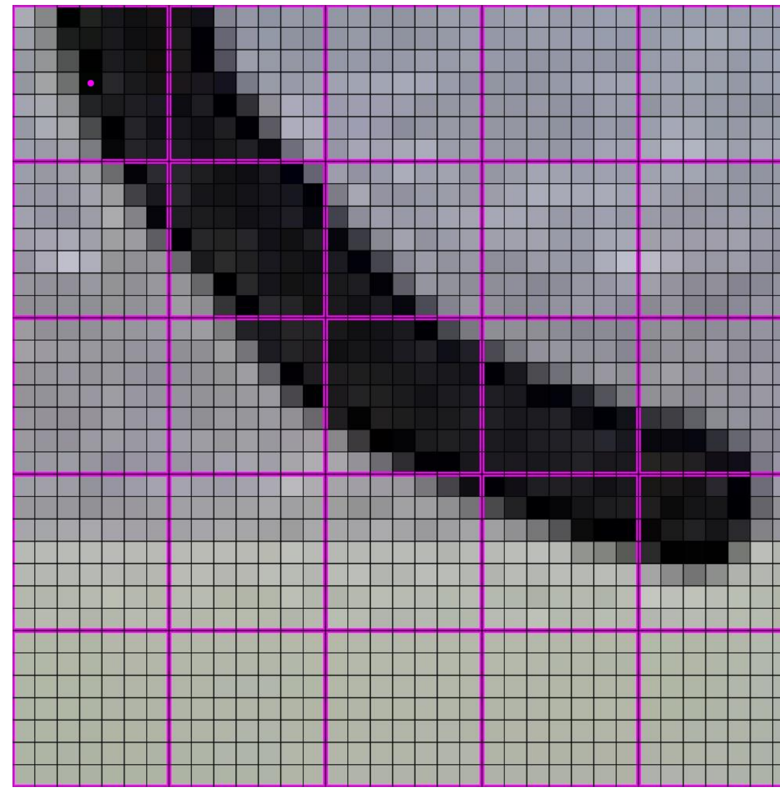


448x448 -> 64x64

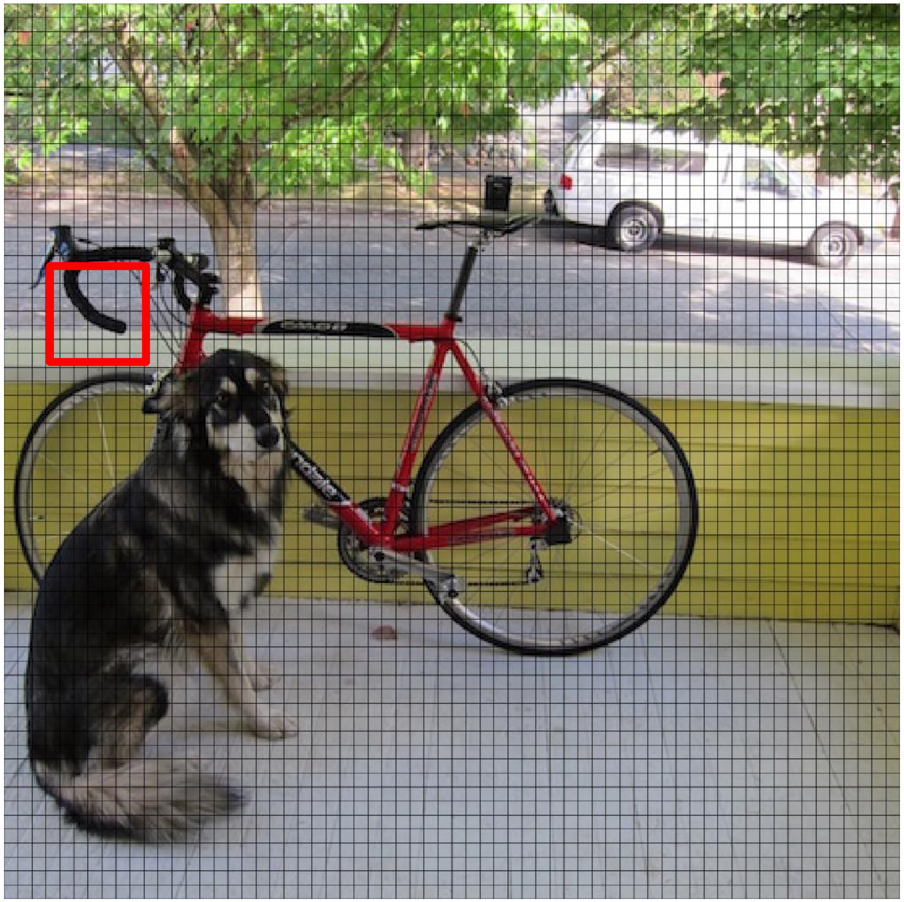


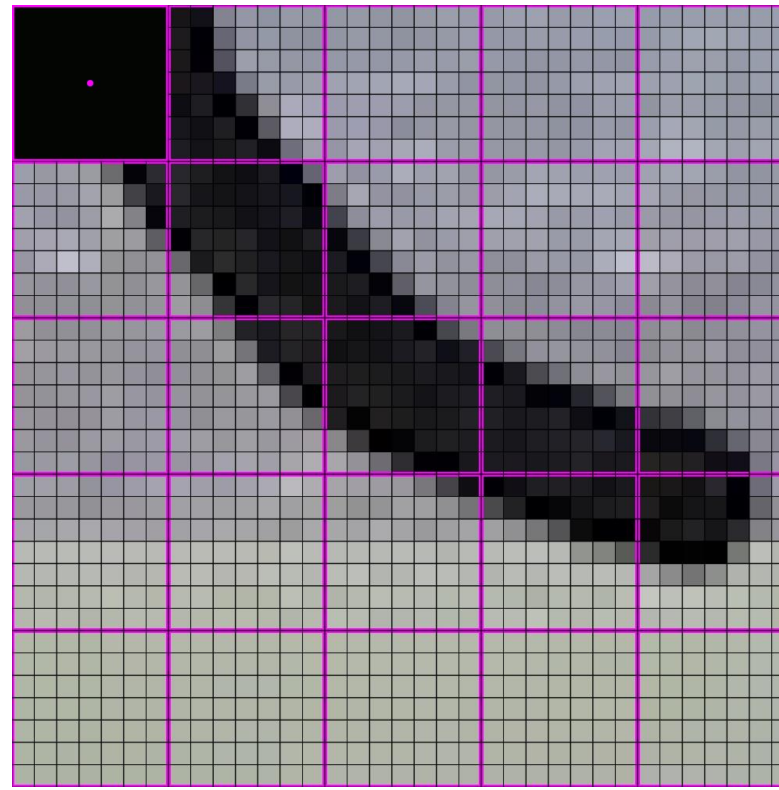
448x448 -> 64x64



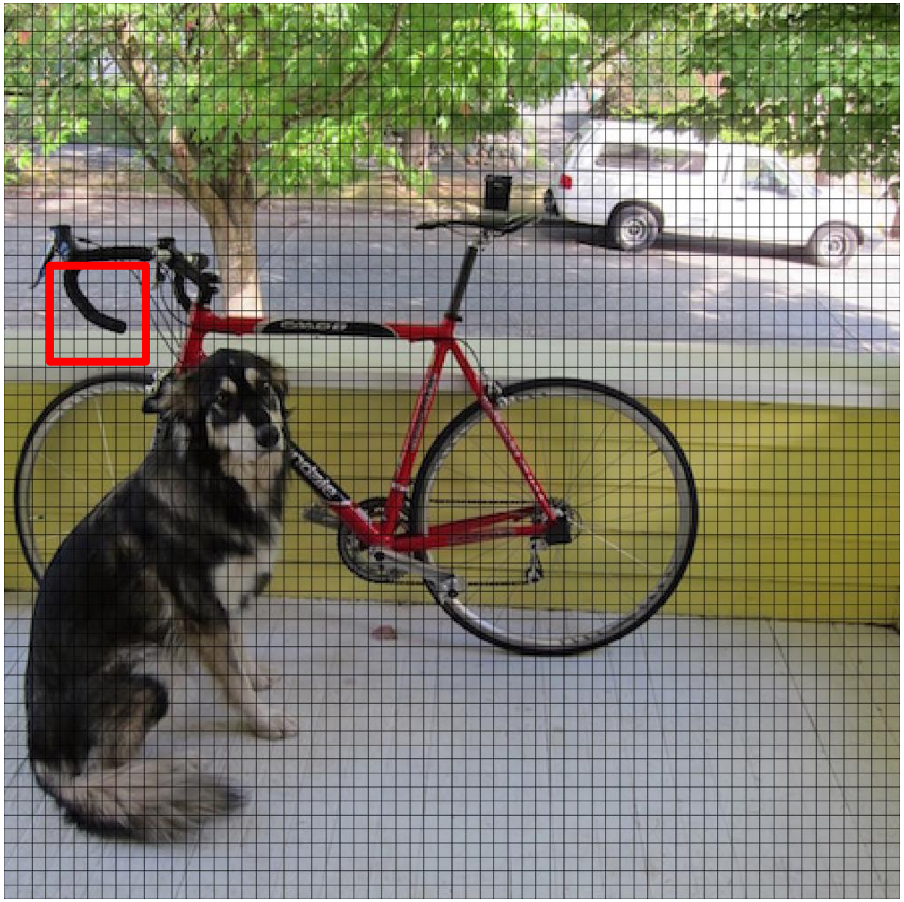


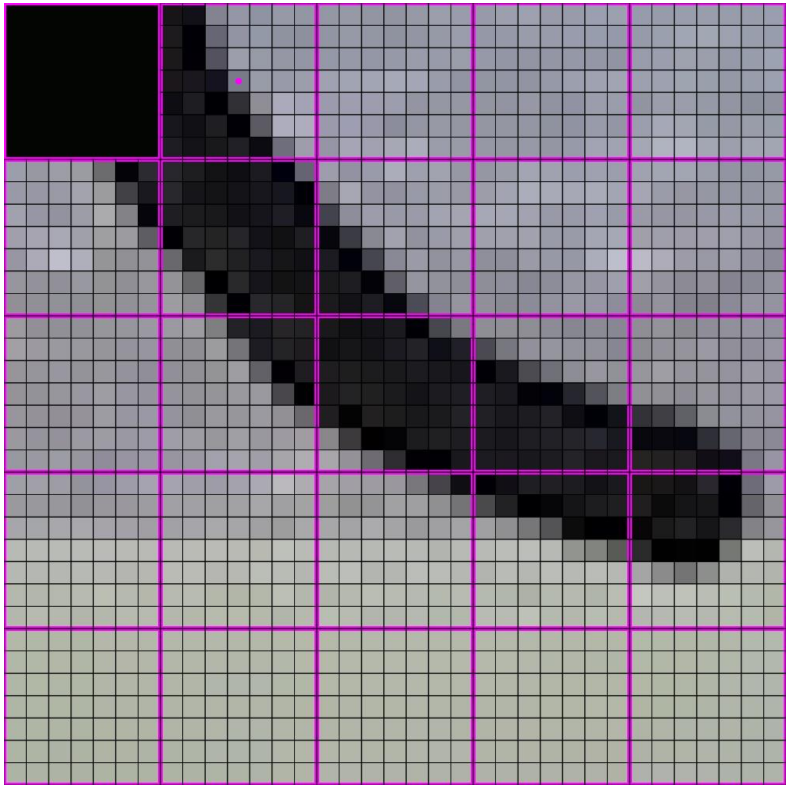
448x448 -> 64x64



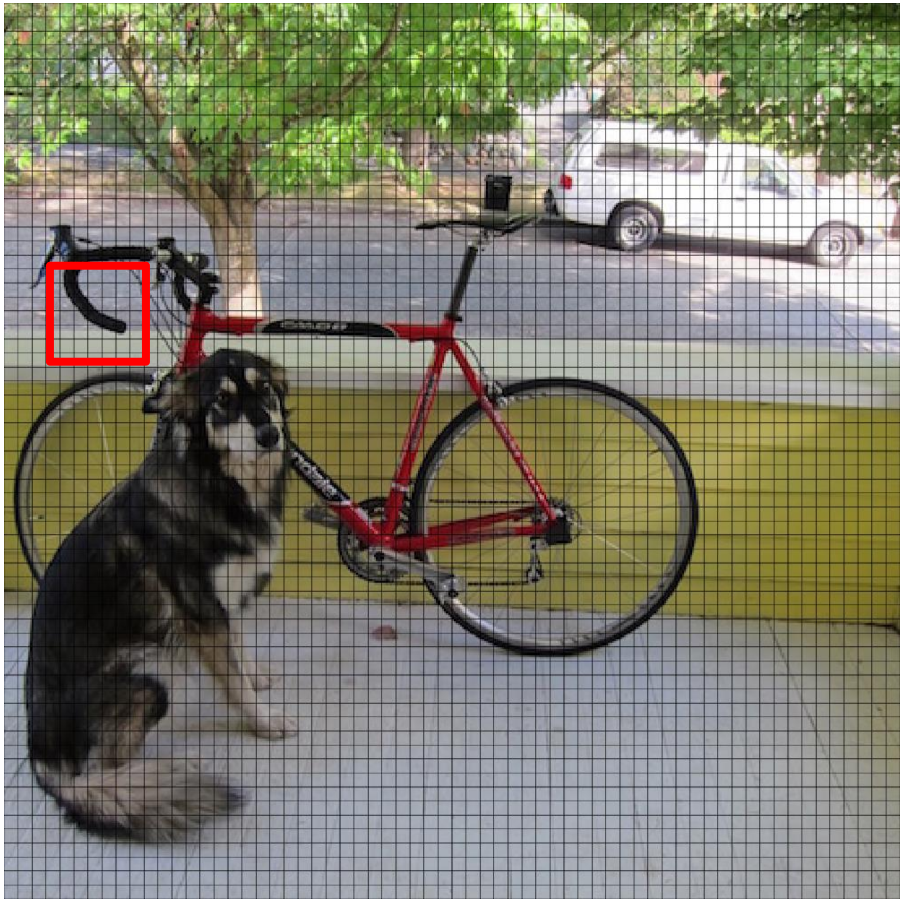


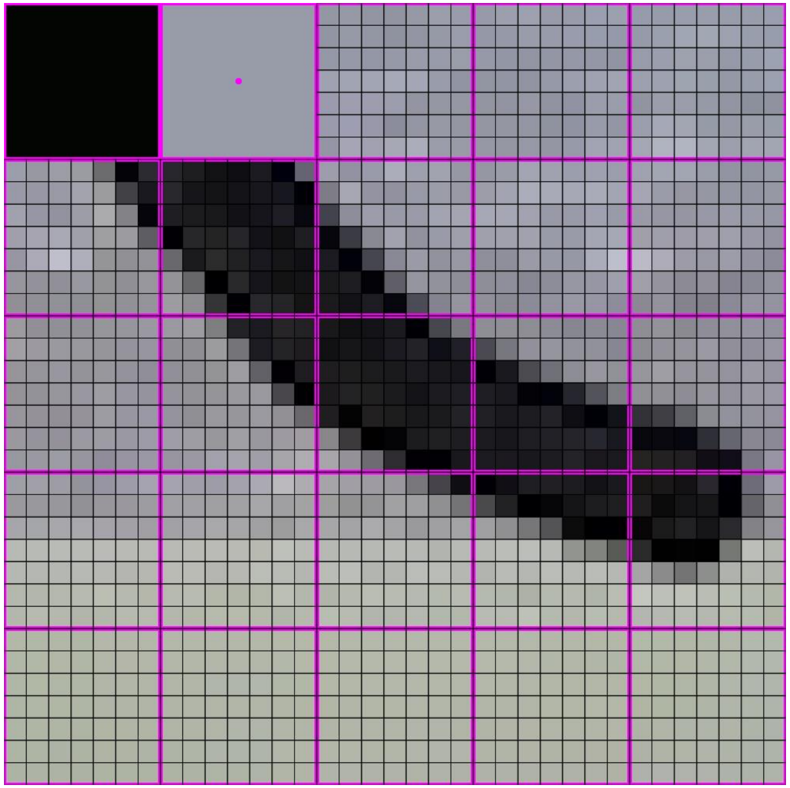
448x448 -> 64x64



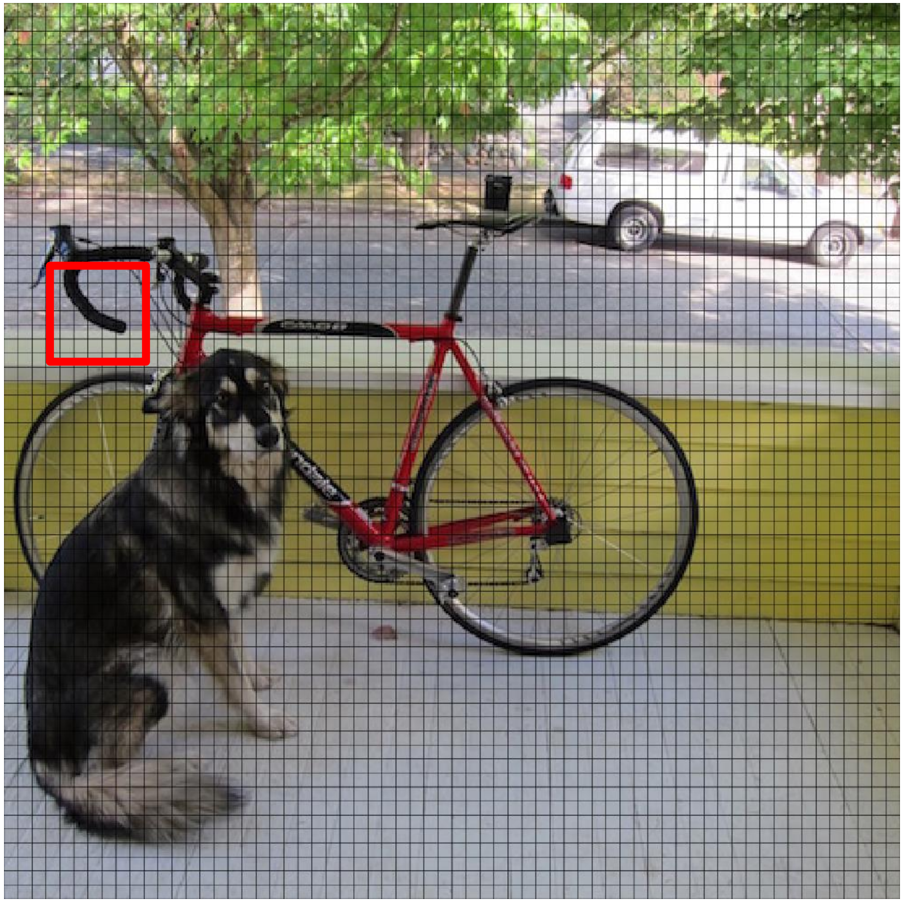


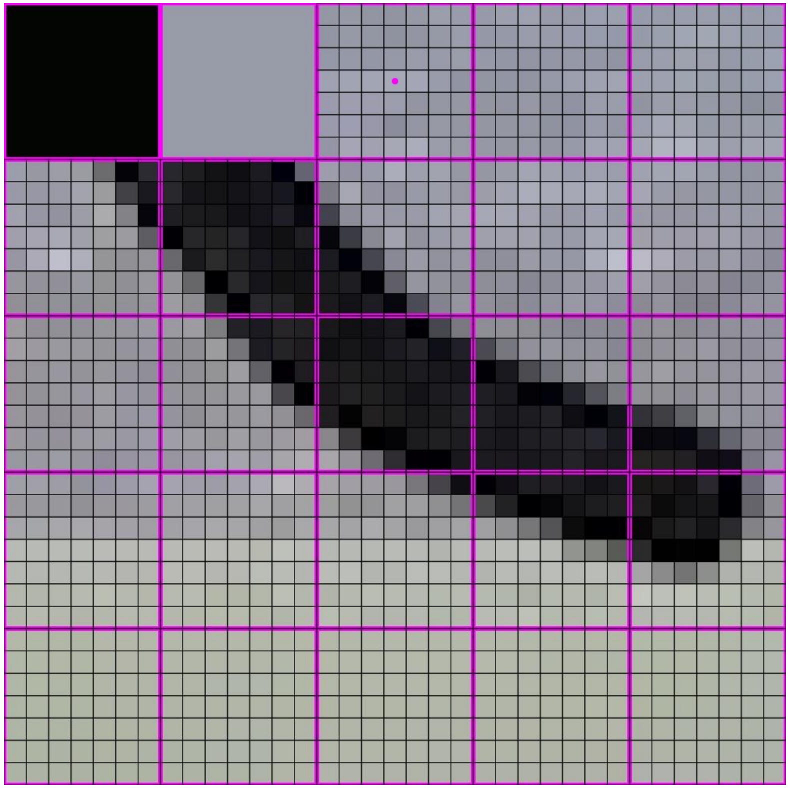
448x448 -> 64x64



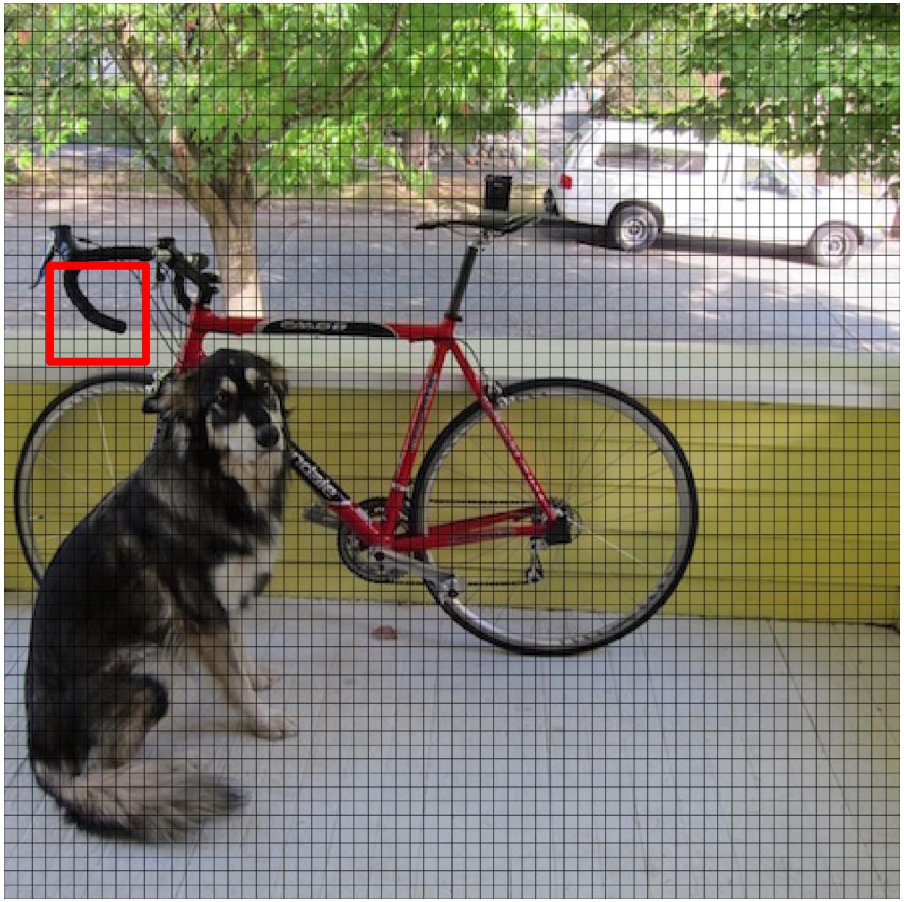


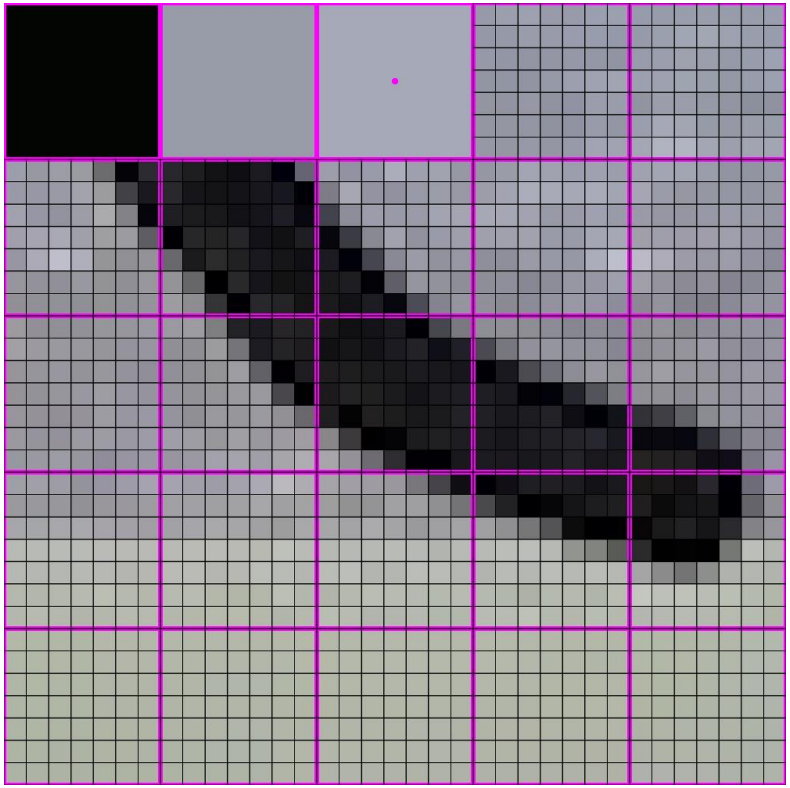
448x448 -> 64x64



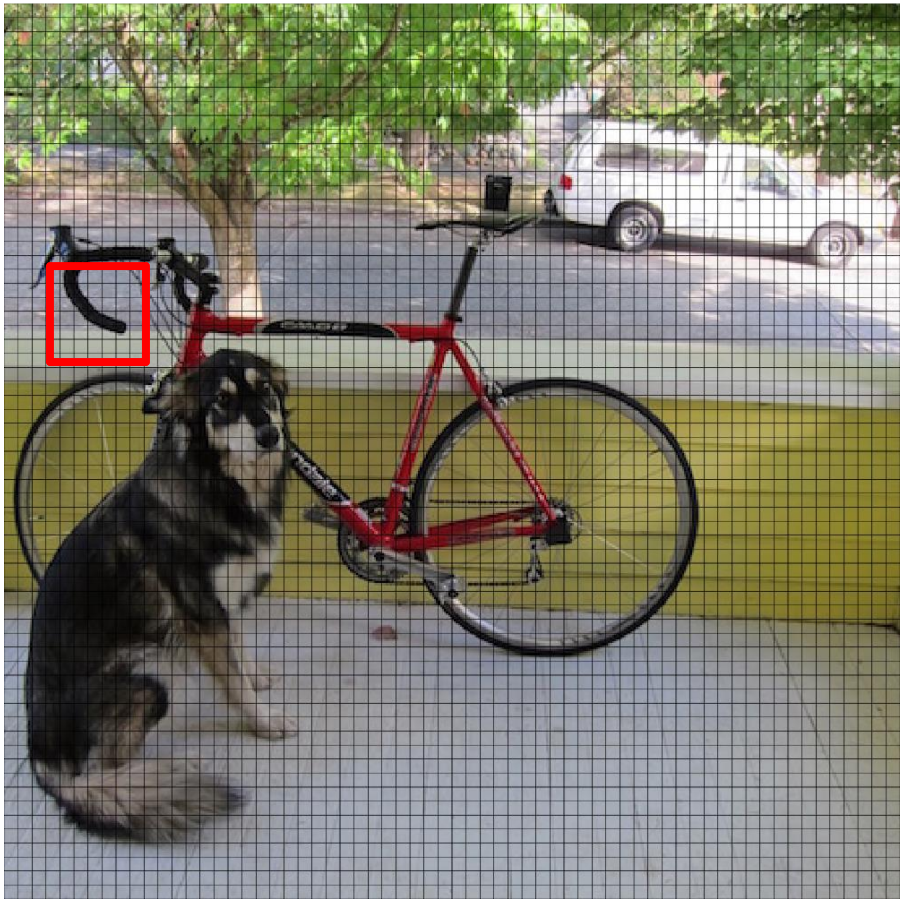


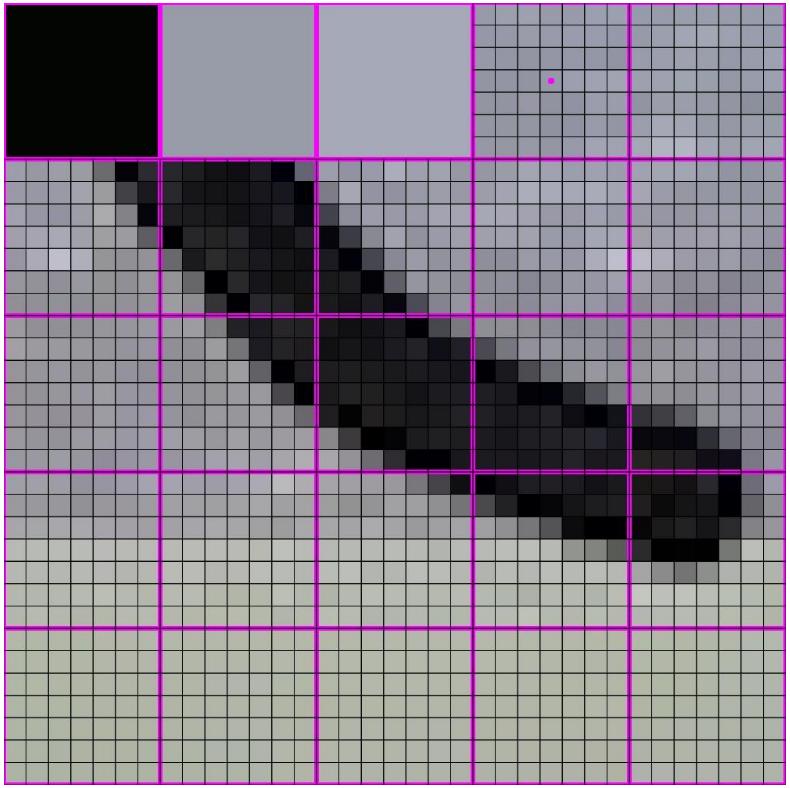
448x448 -> 64x64



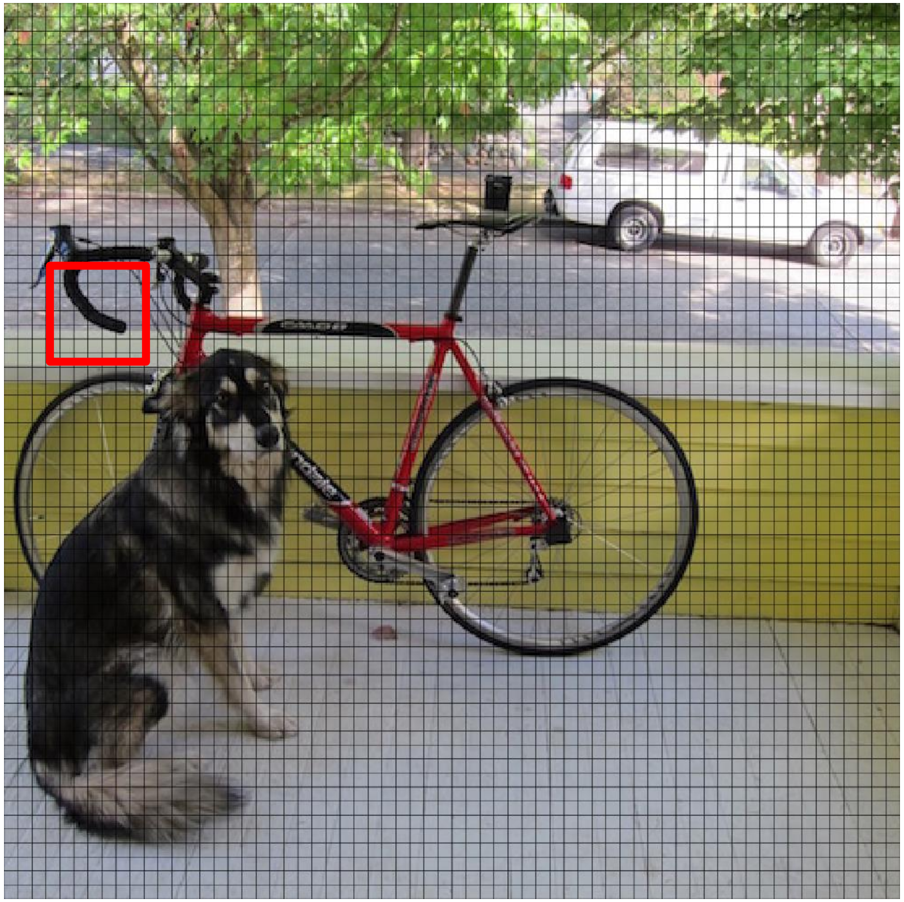


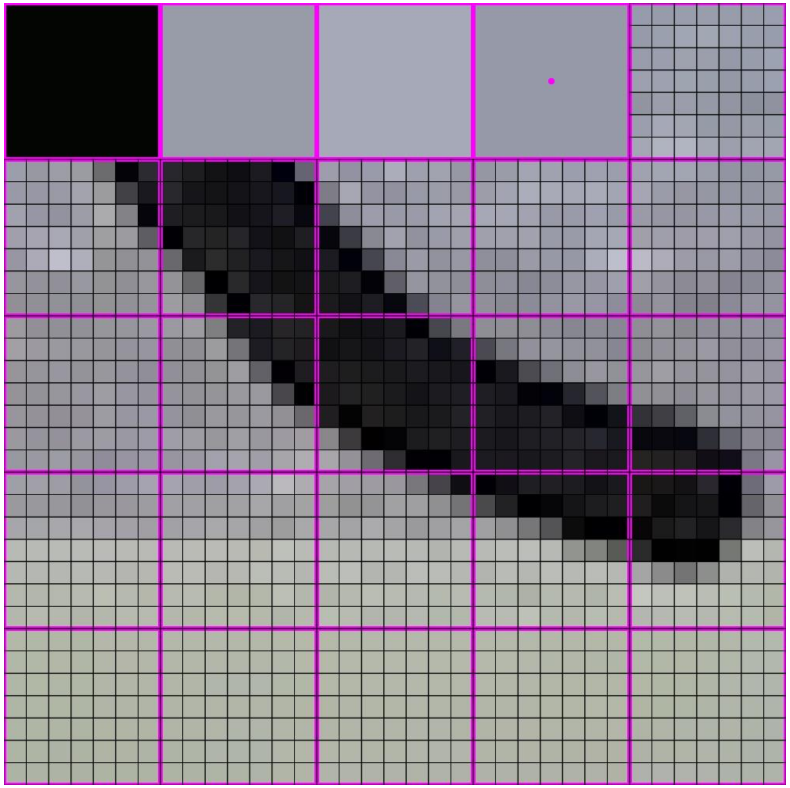
448x448 -> 64x64



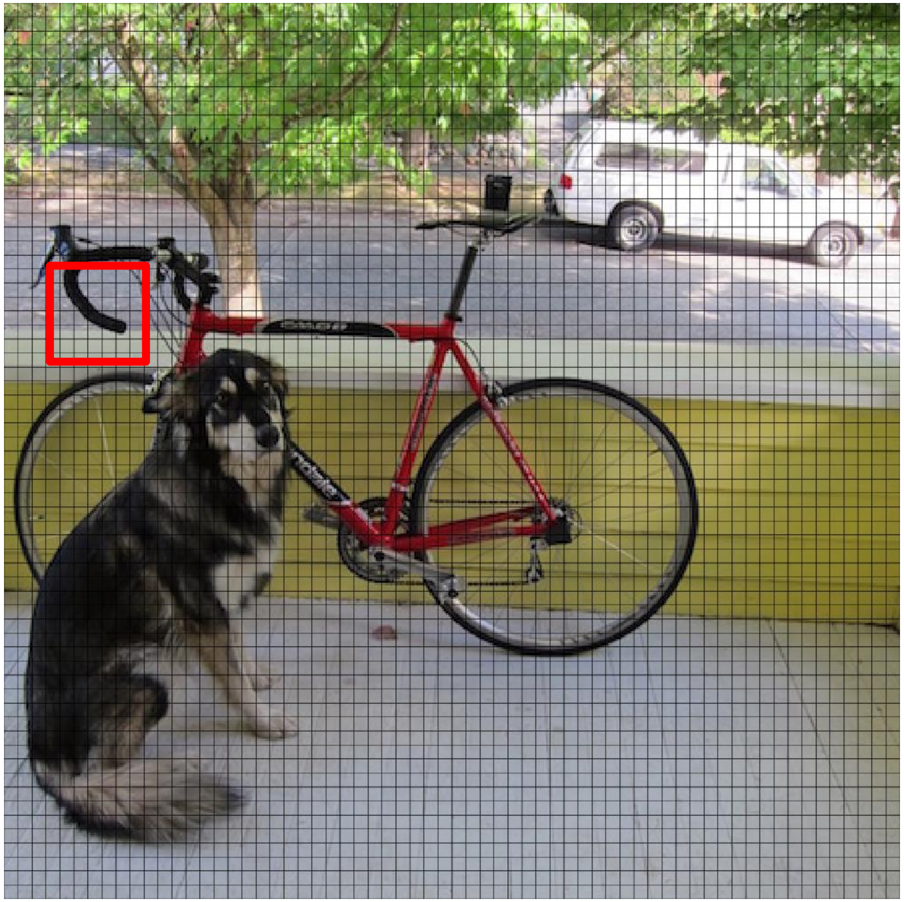


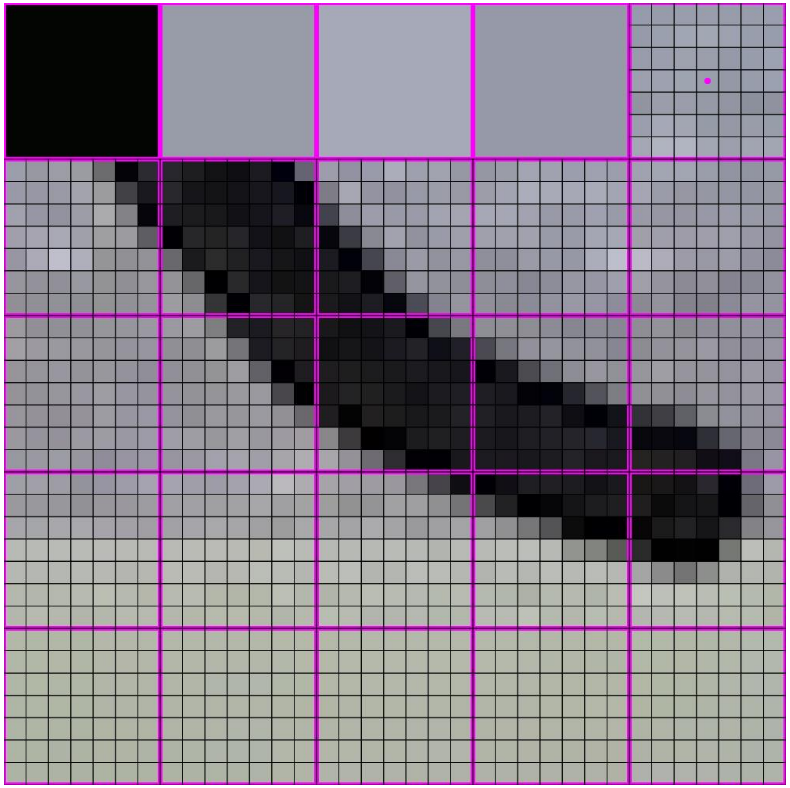
448x448 -> 64x64



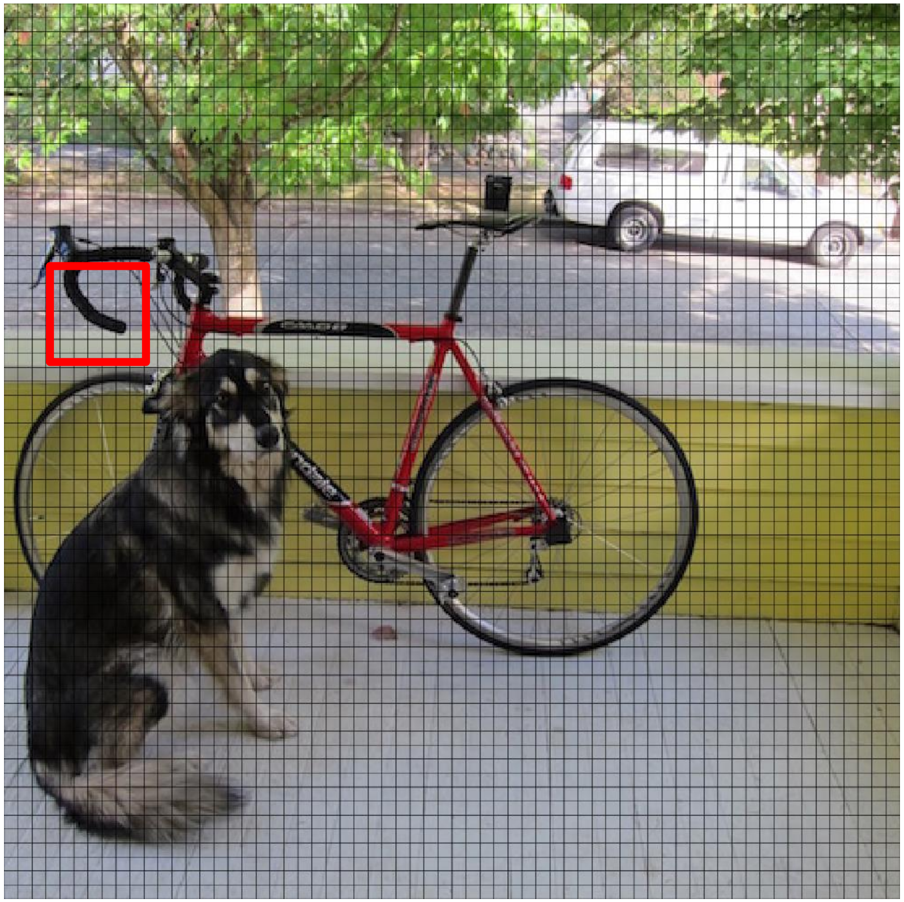


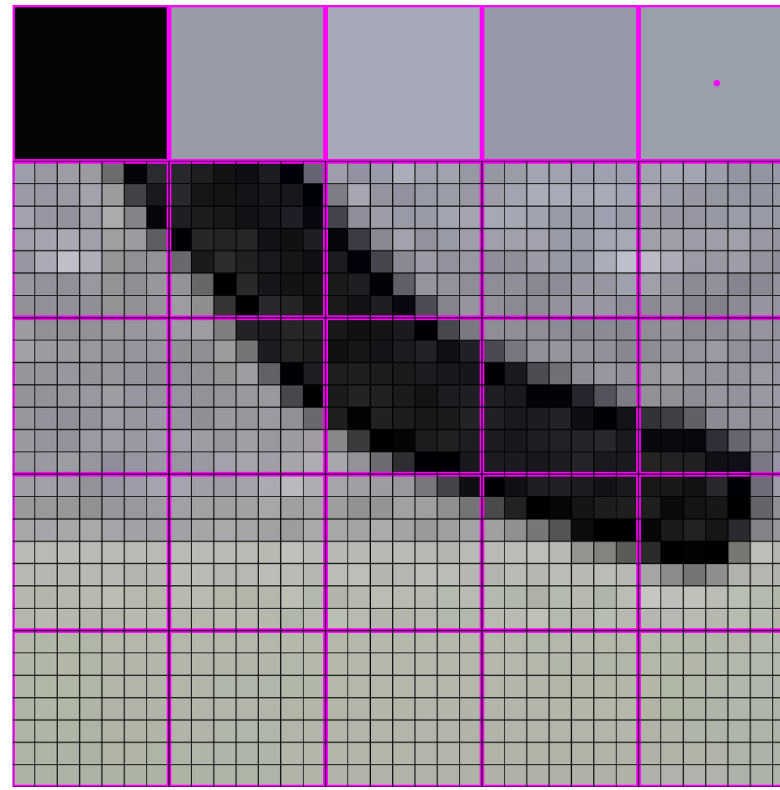
448x448 -> 64x64



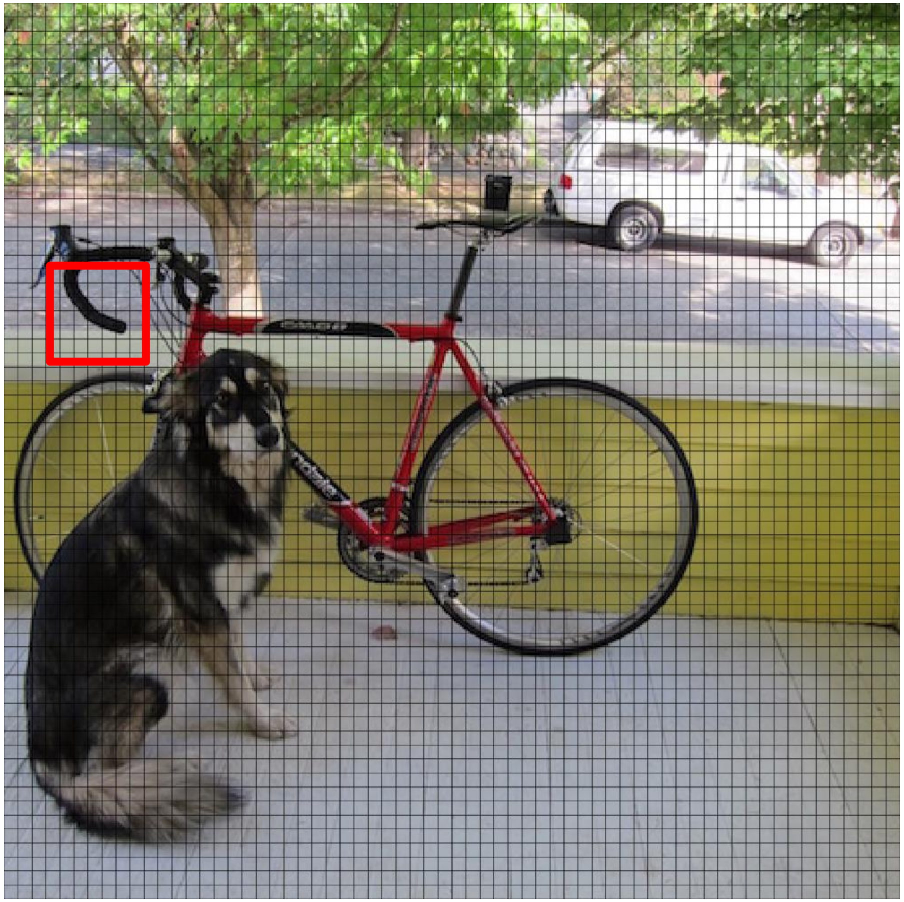


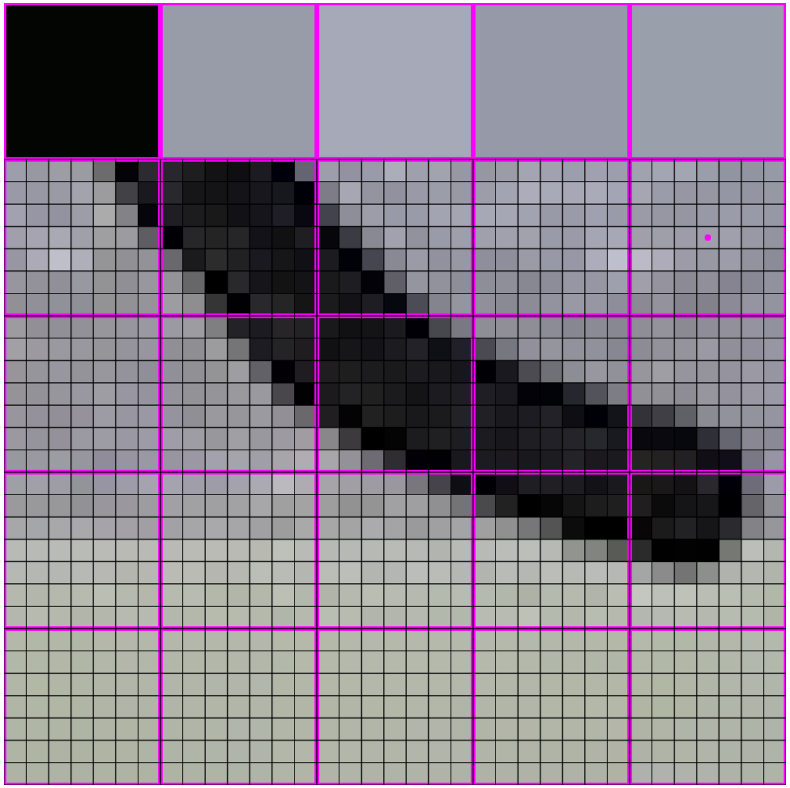
448x448 -> 64x64



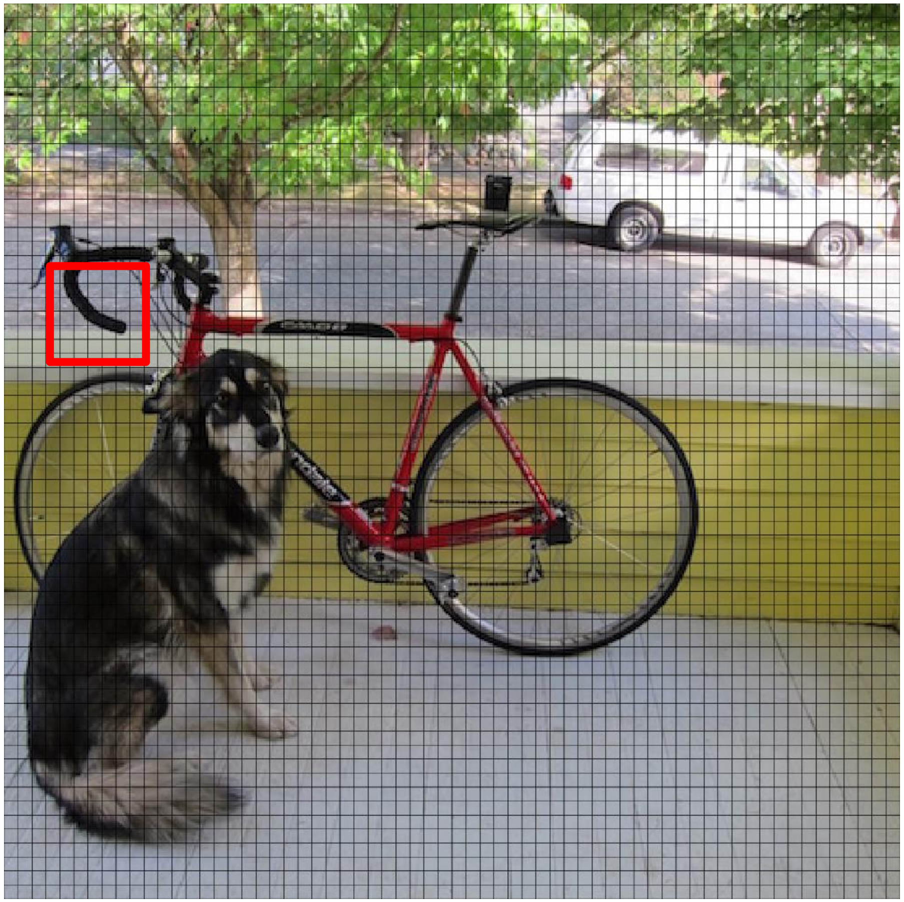


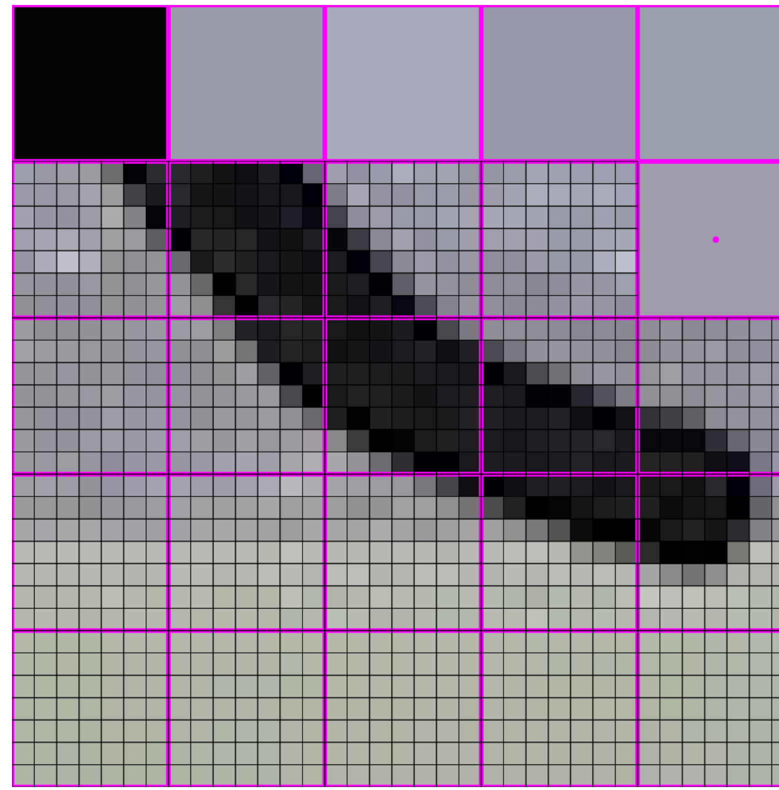
448x448 -> 64x64



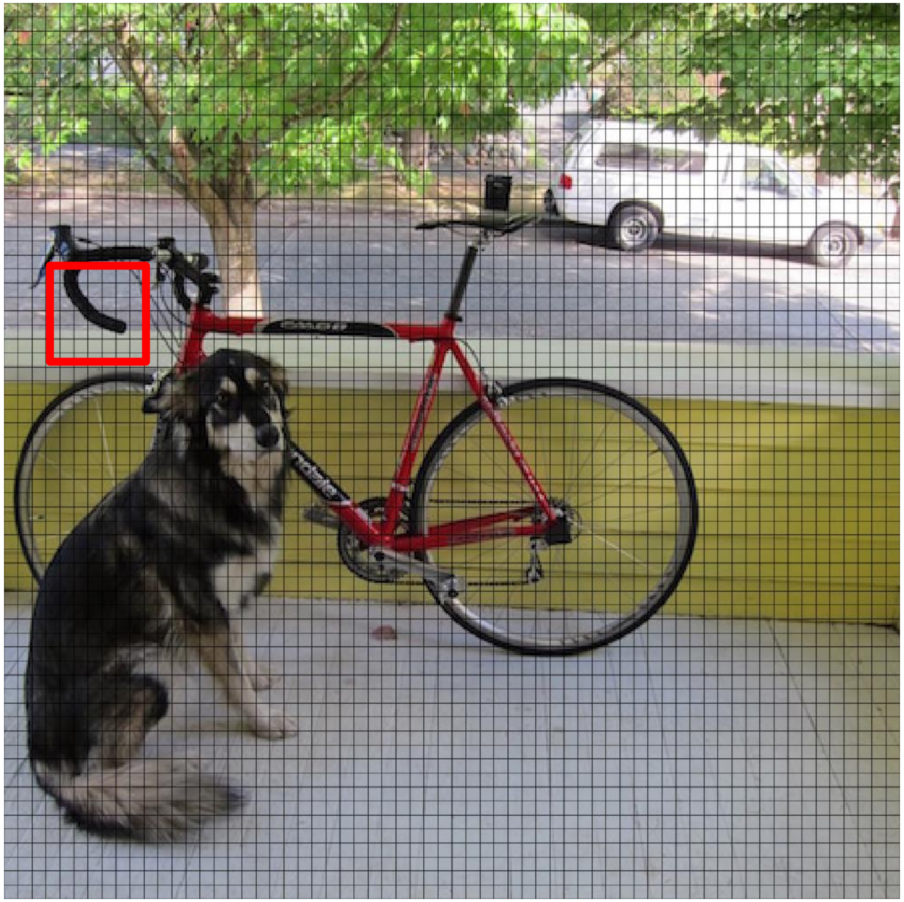


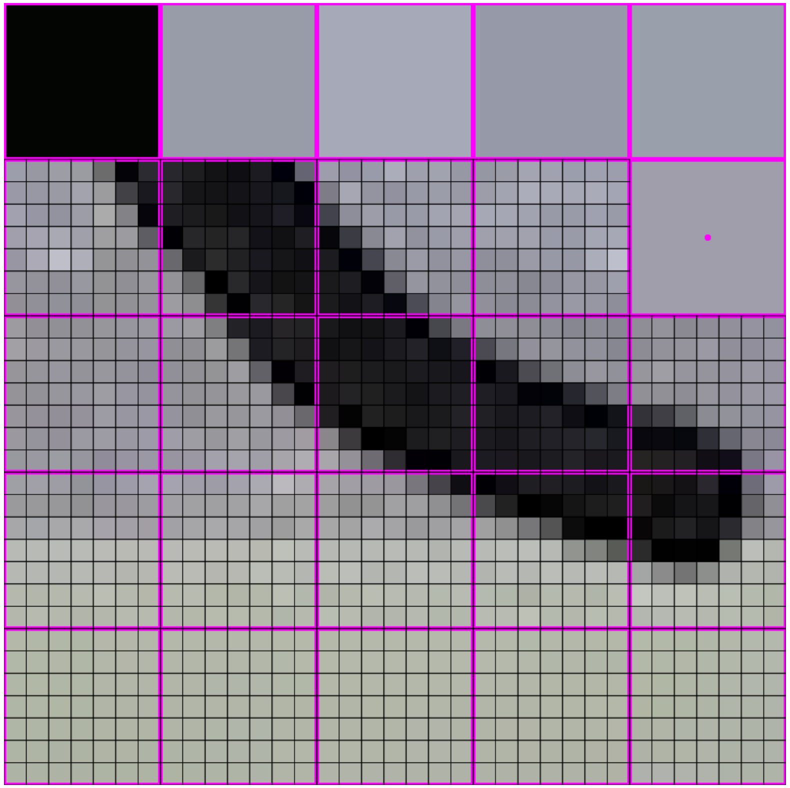
448x448 -> 64x64



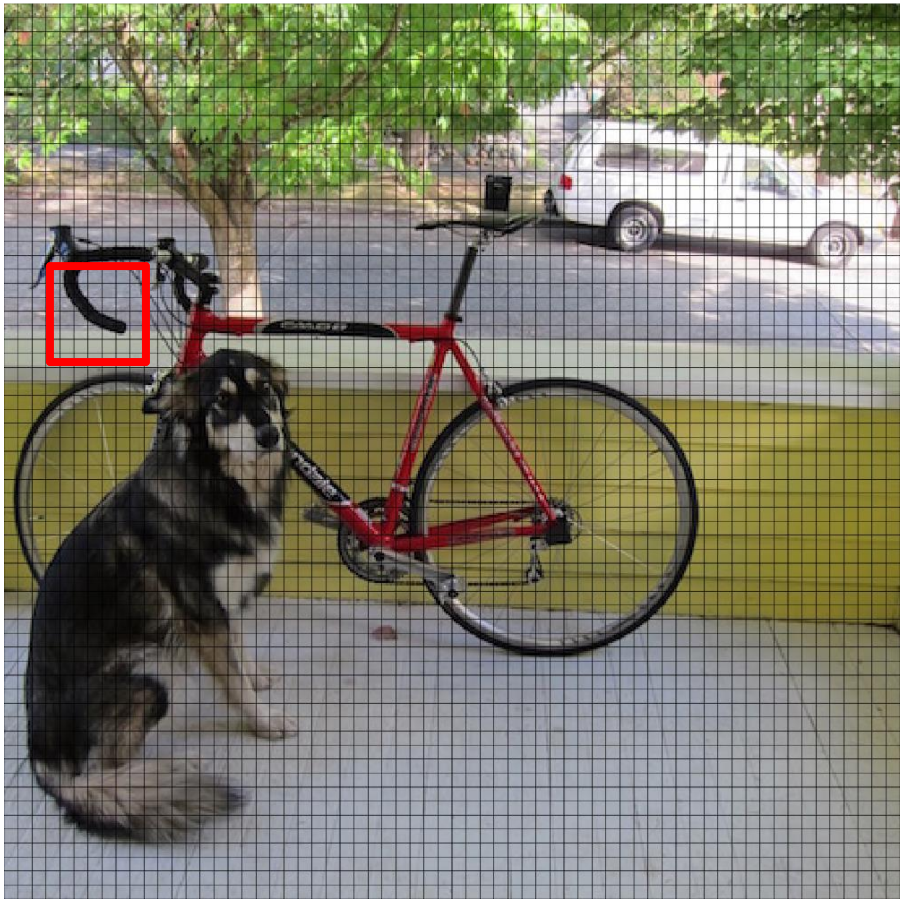


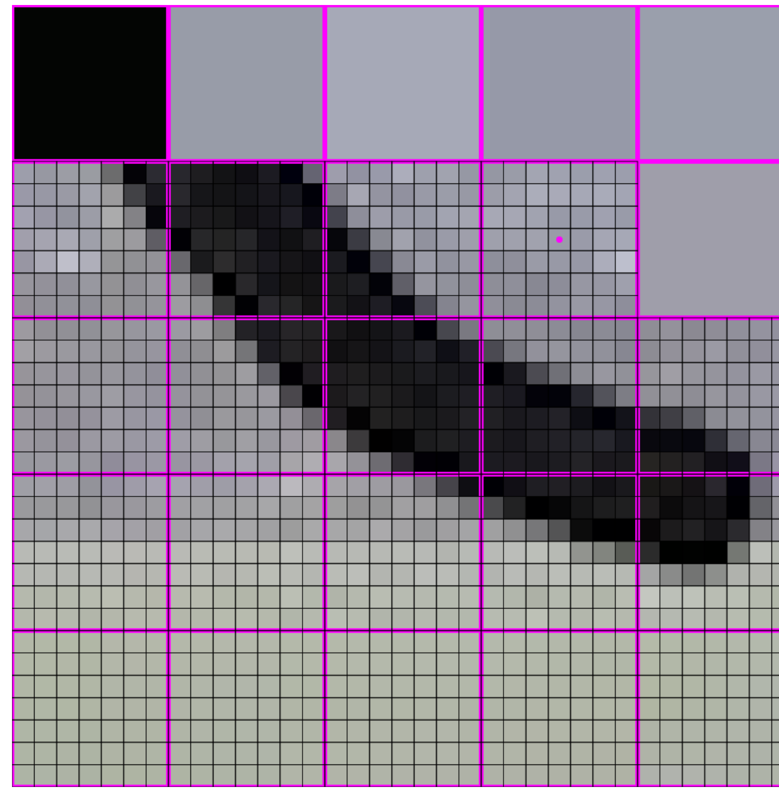
448x448 -> 64x64



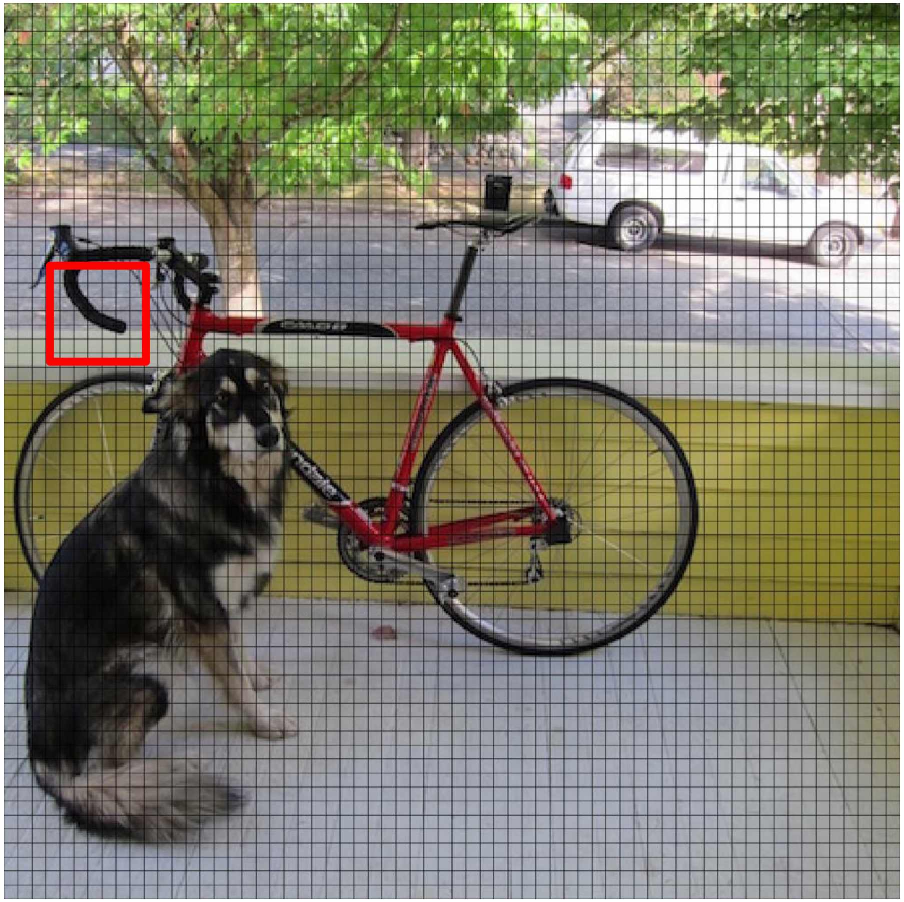


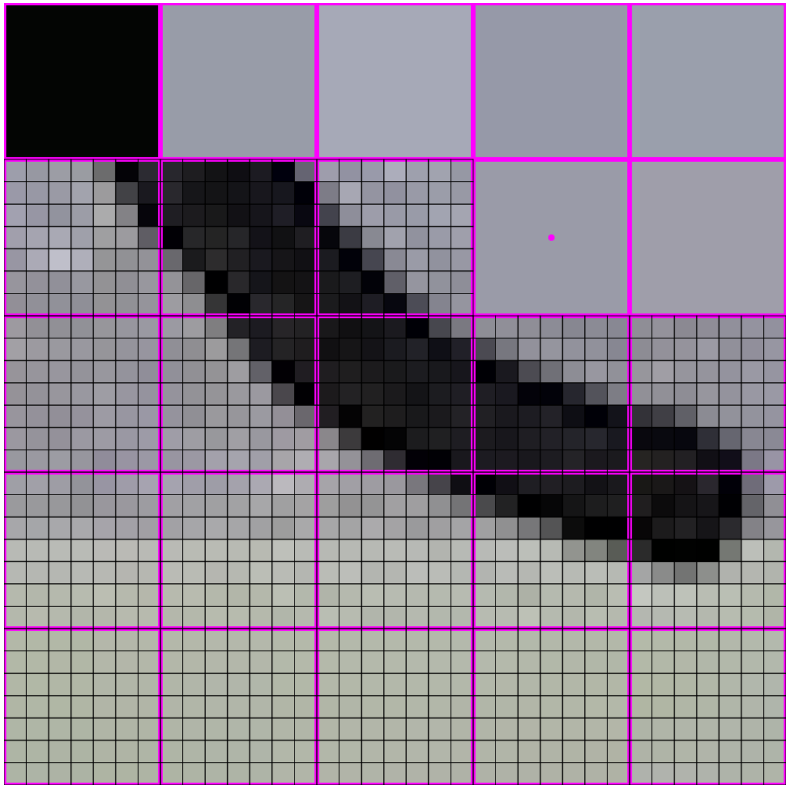
448x448 -> 64x64



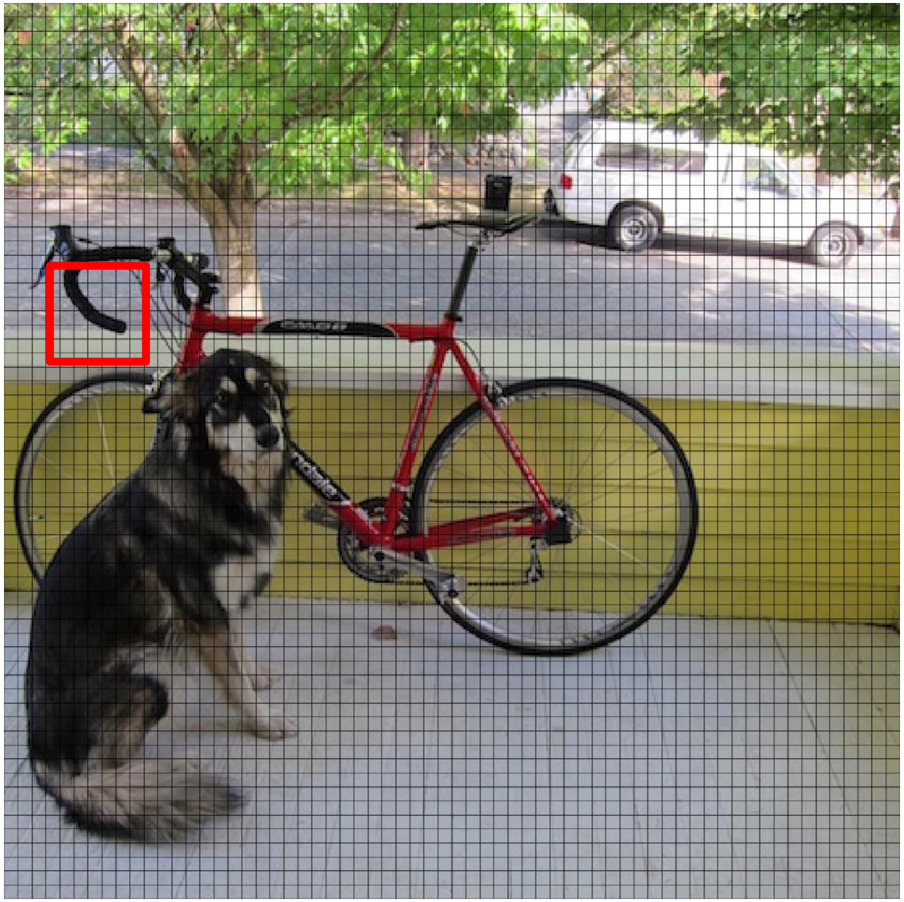


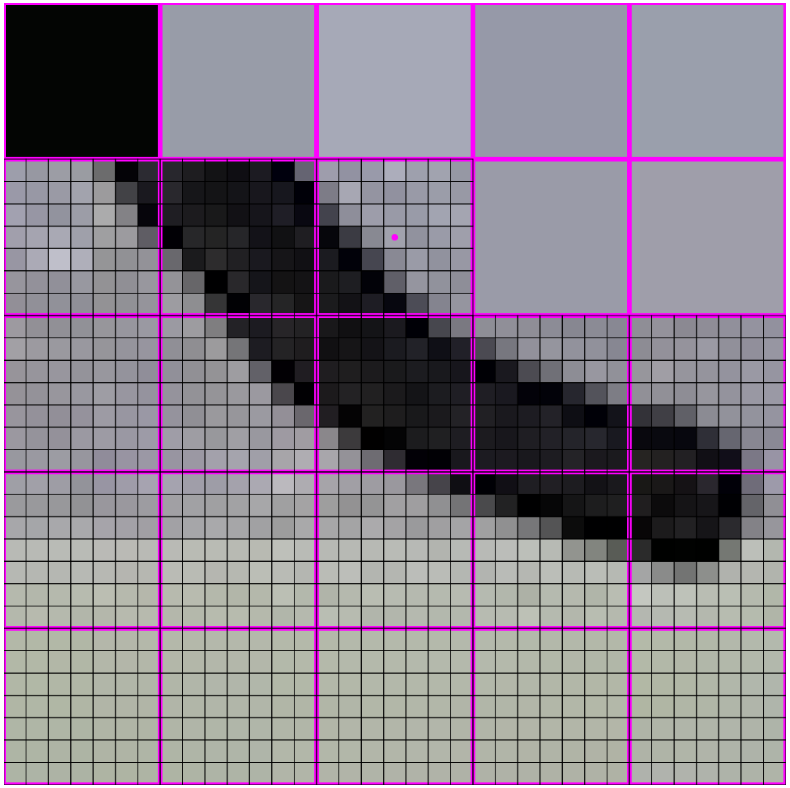
448x448 -> 64x64



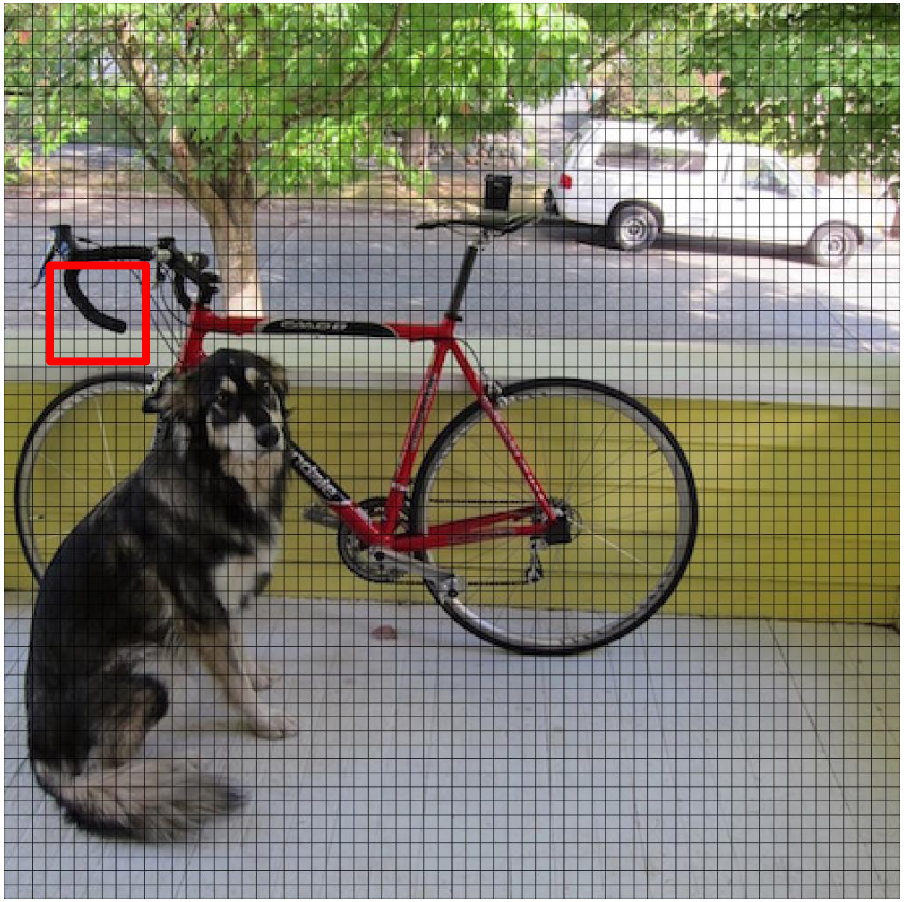


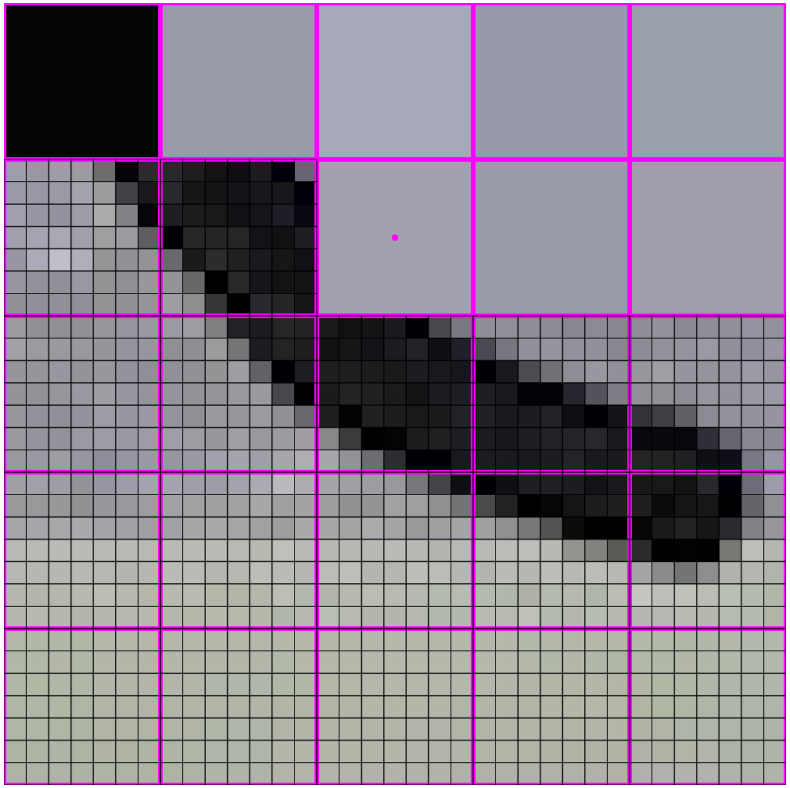
448x448 -> 64x64



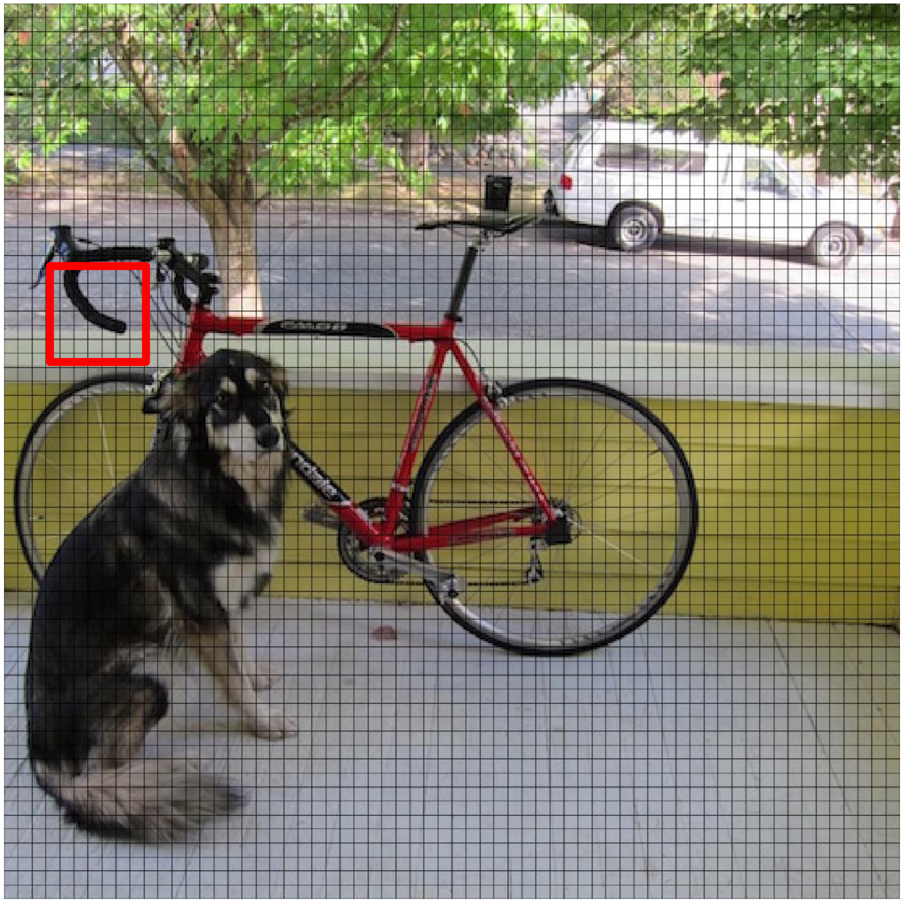


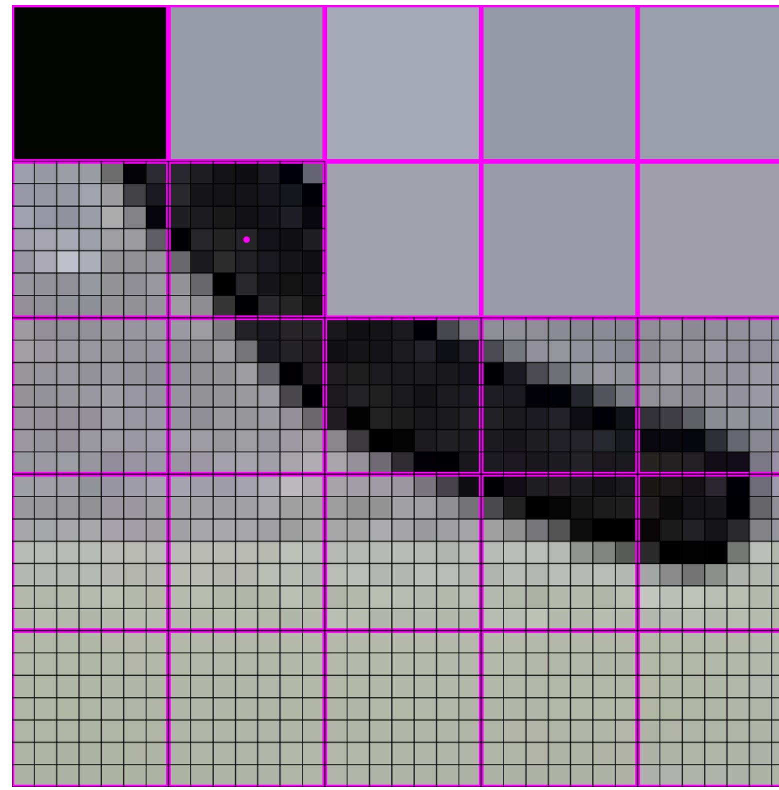
448x448 -> 64x64



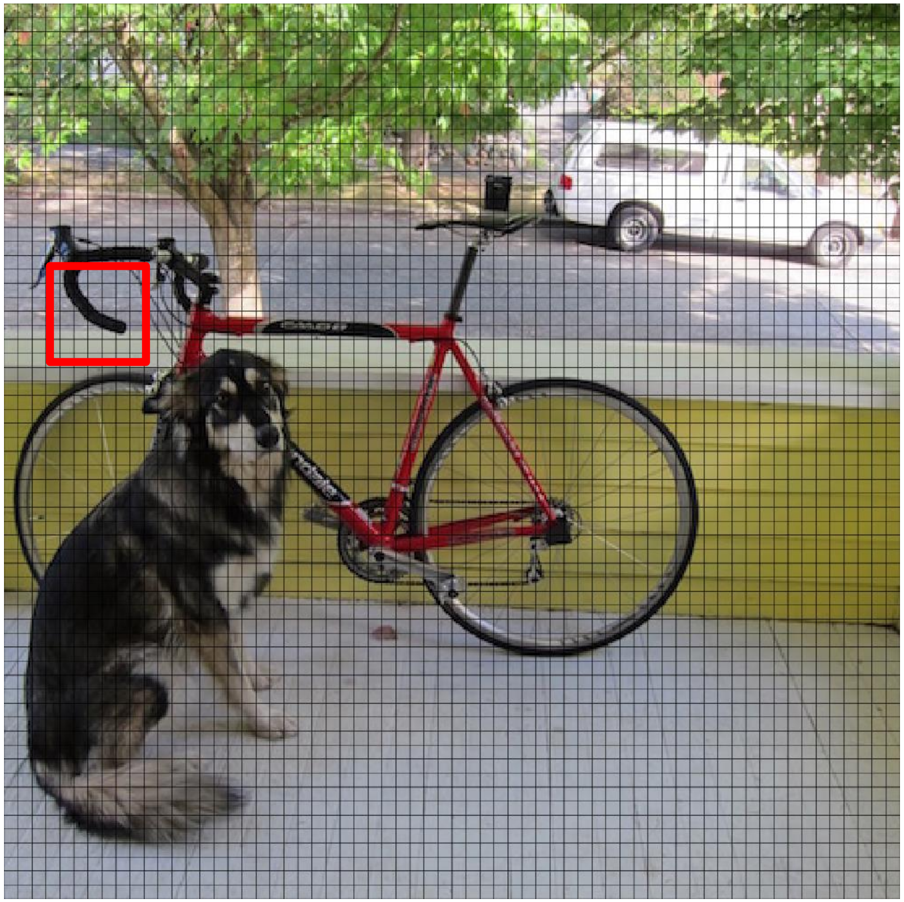


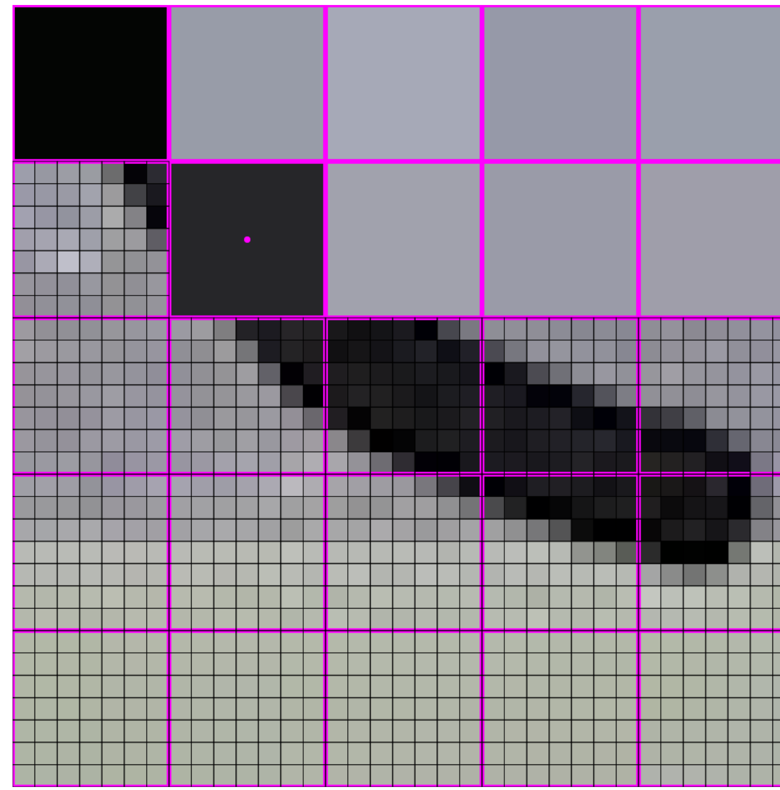
448x448 -> 64x64



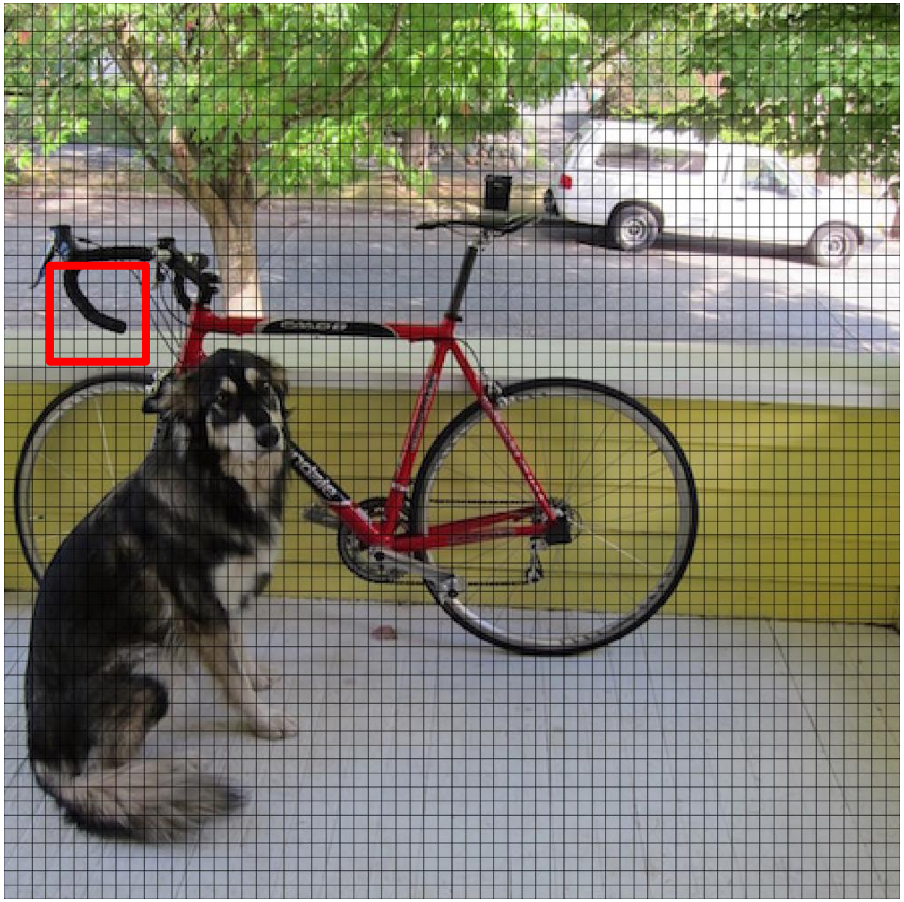


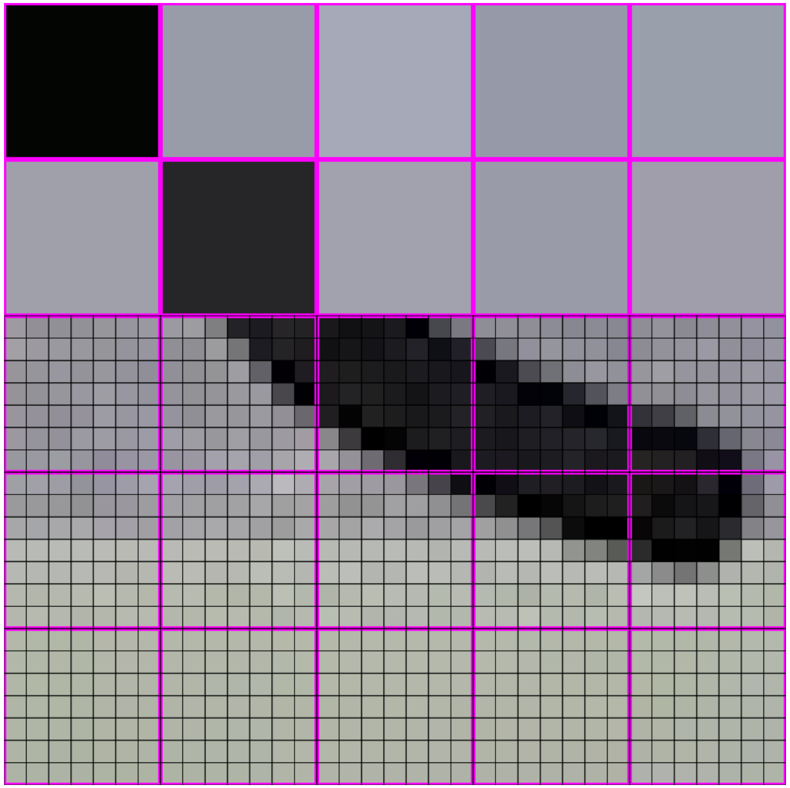
448x448 -> 64x64



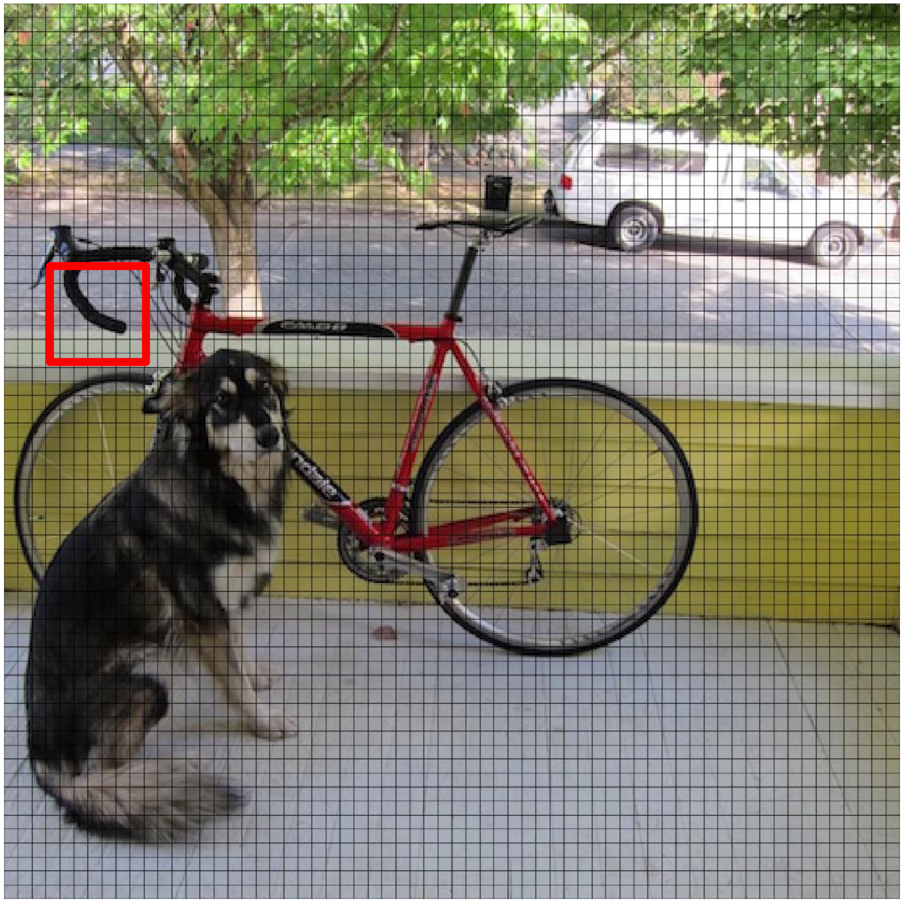


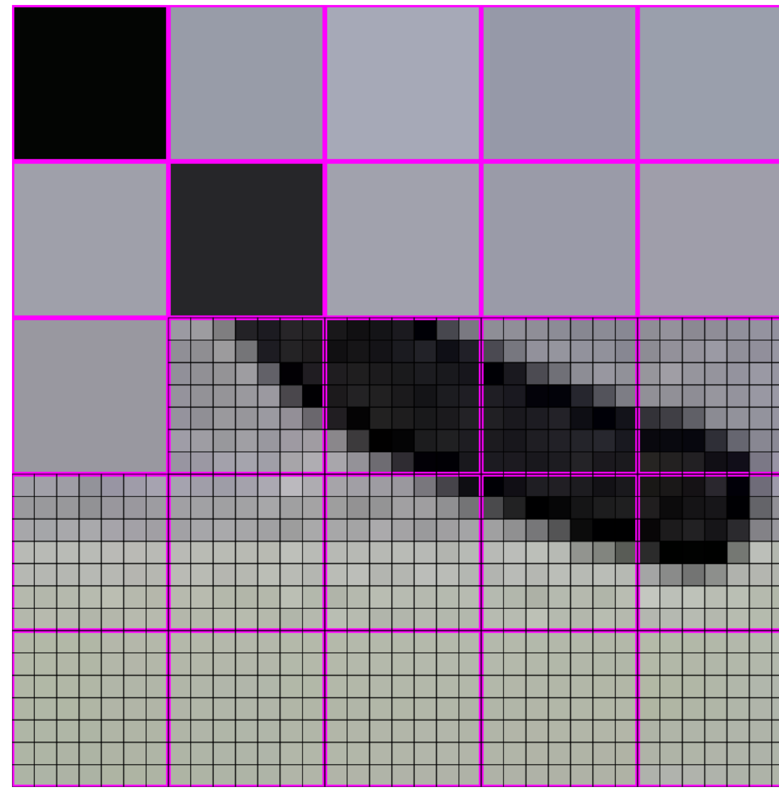
448x448 -> 64x64



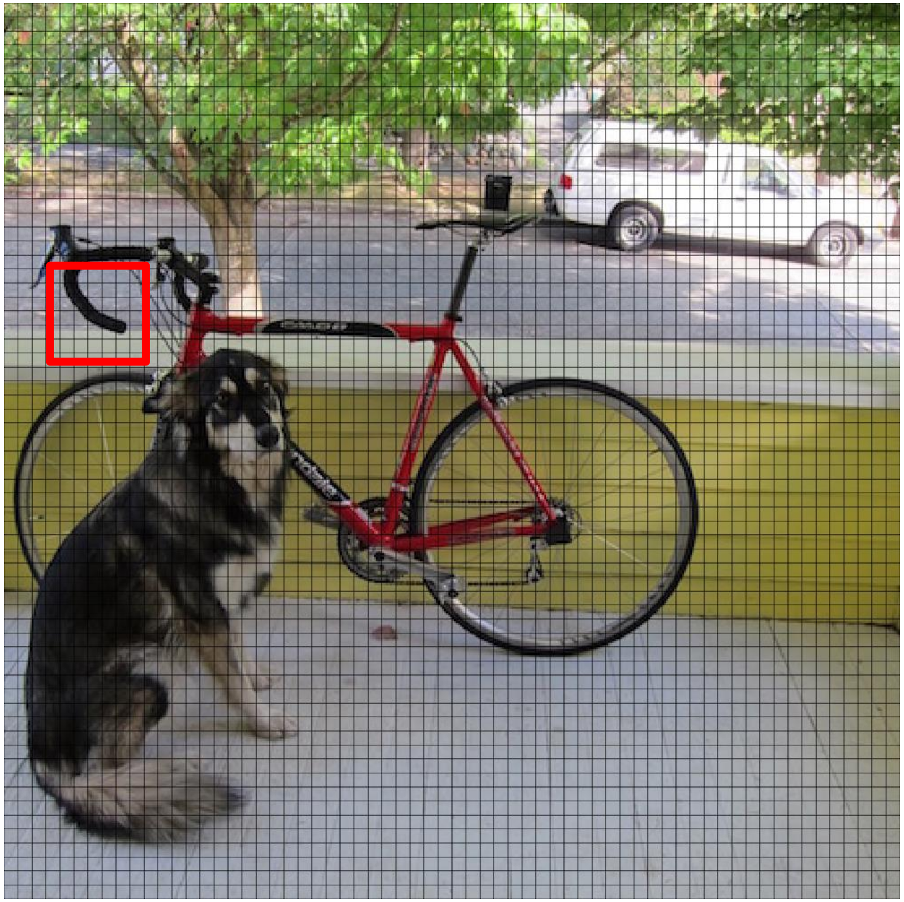


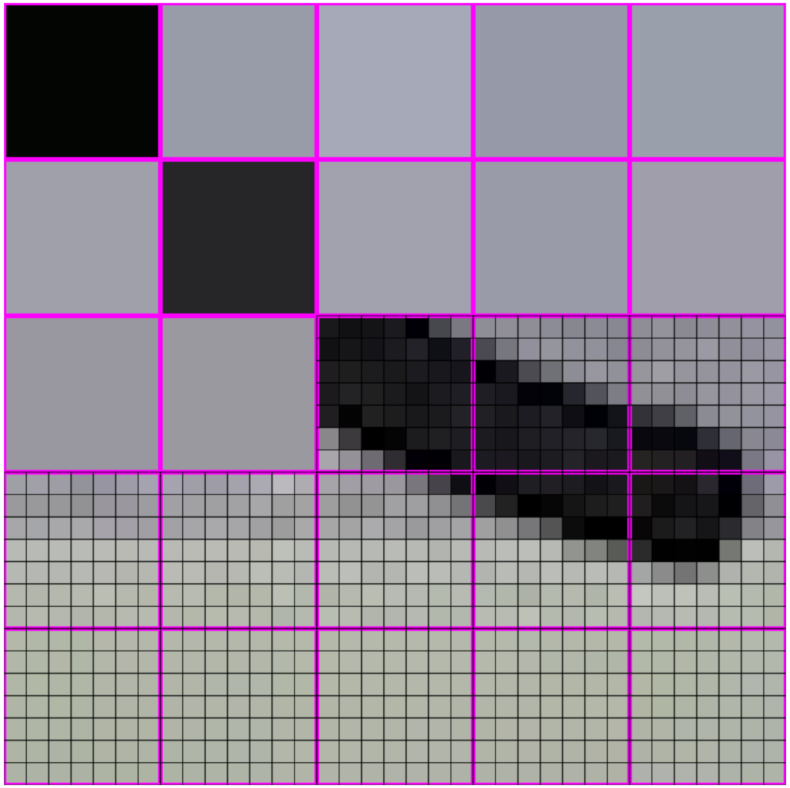
448x448 -> 64x64



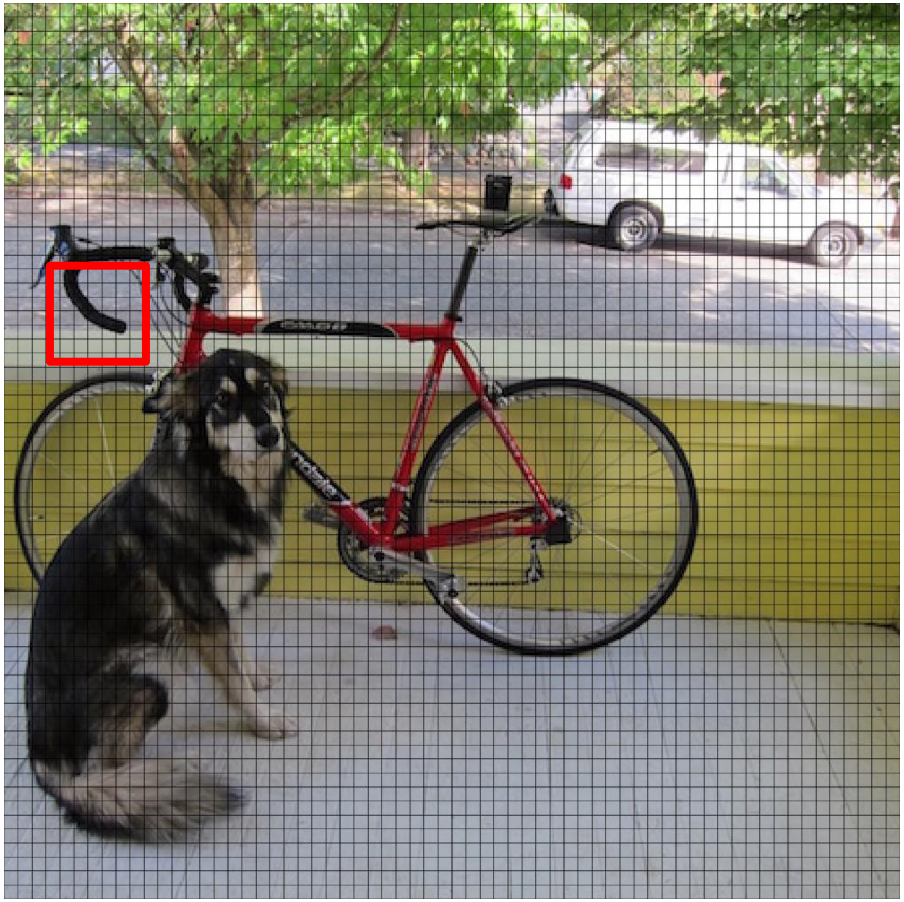


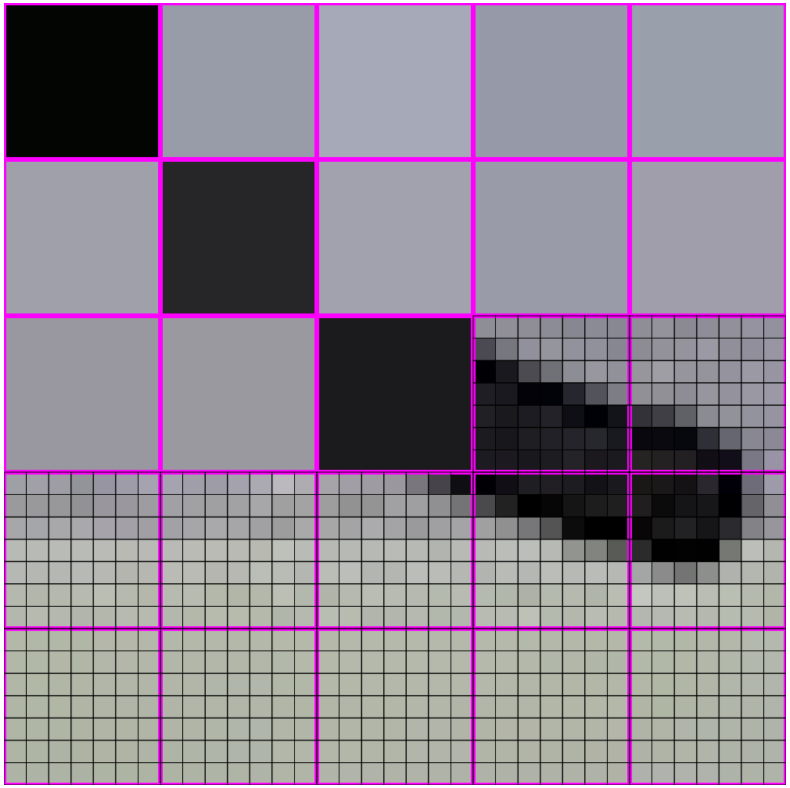
448x448 -> 64x64



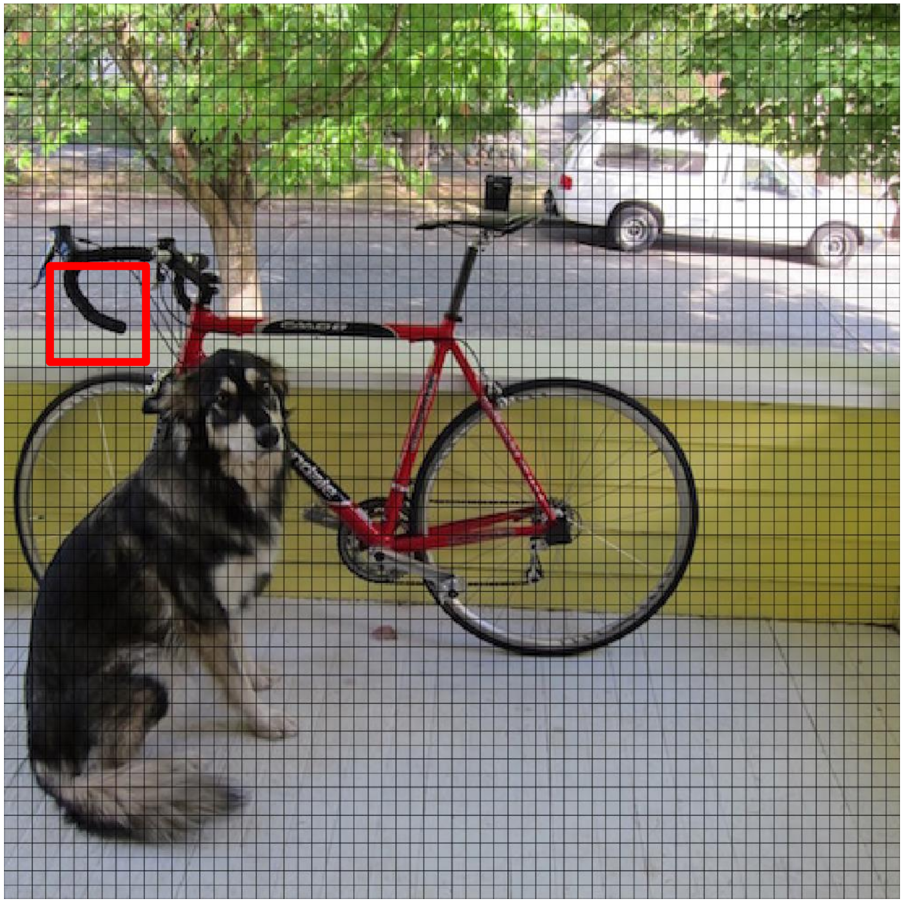


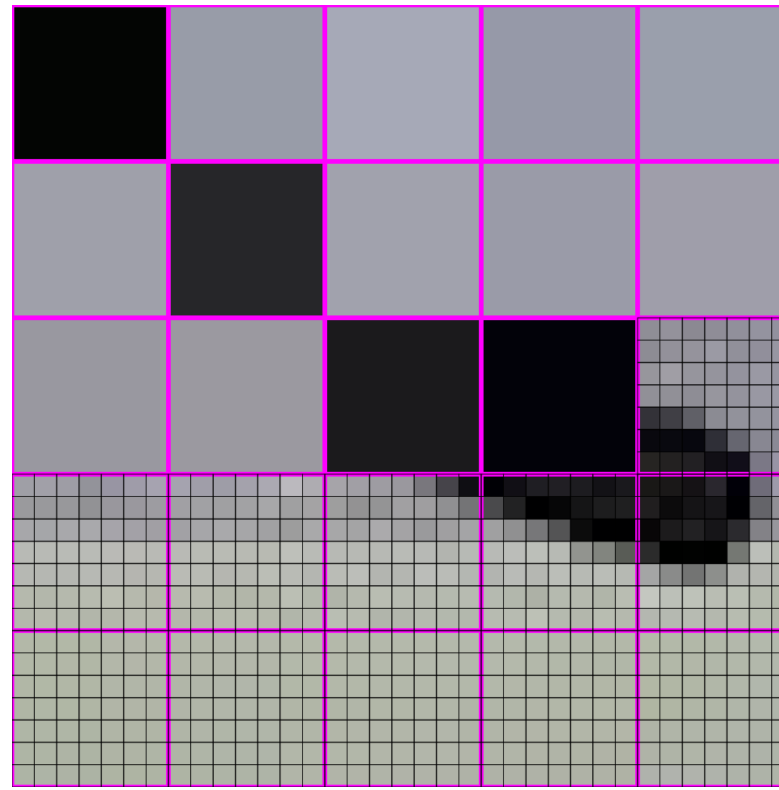
448x448 -> 64x64



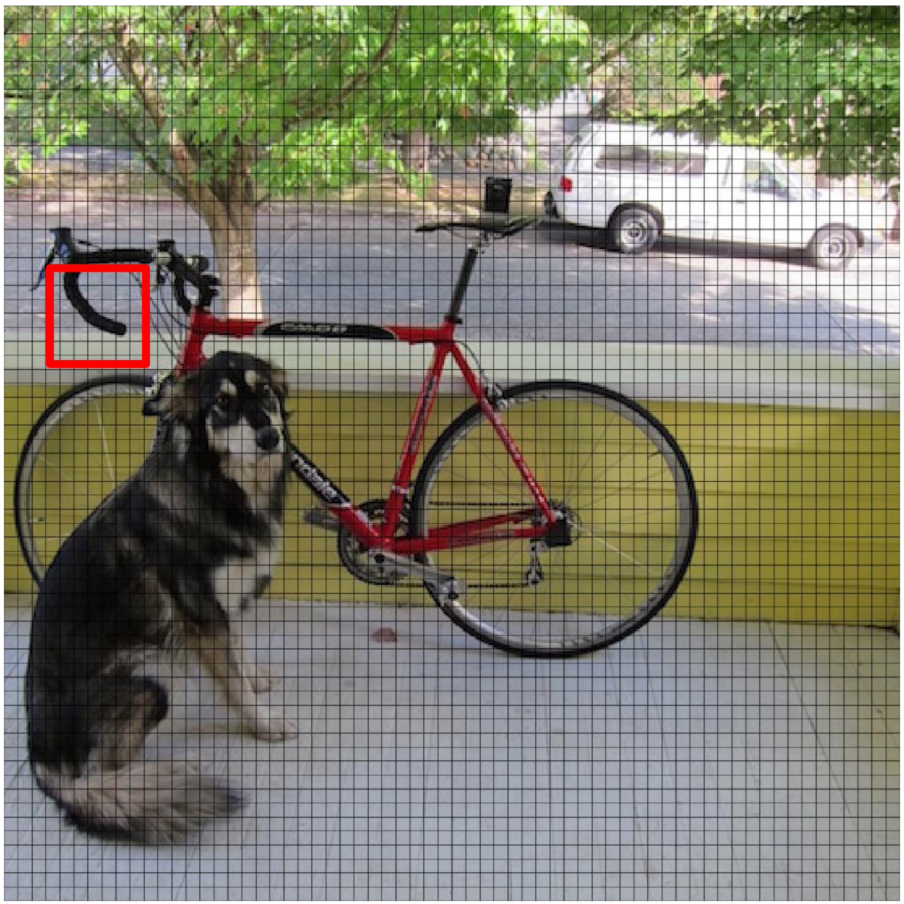


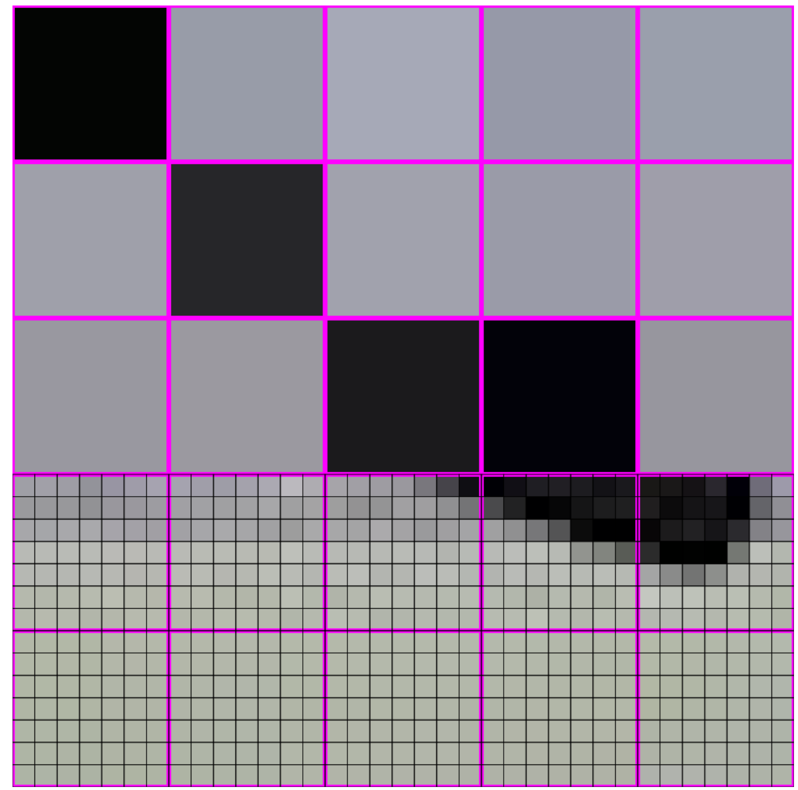
448x448 -> 64x64



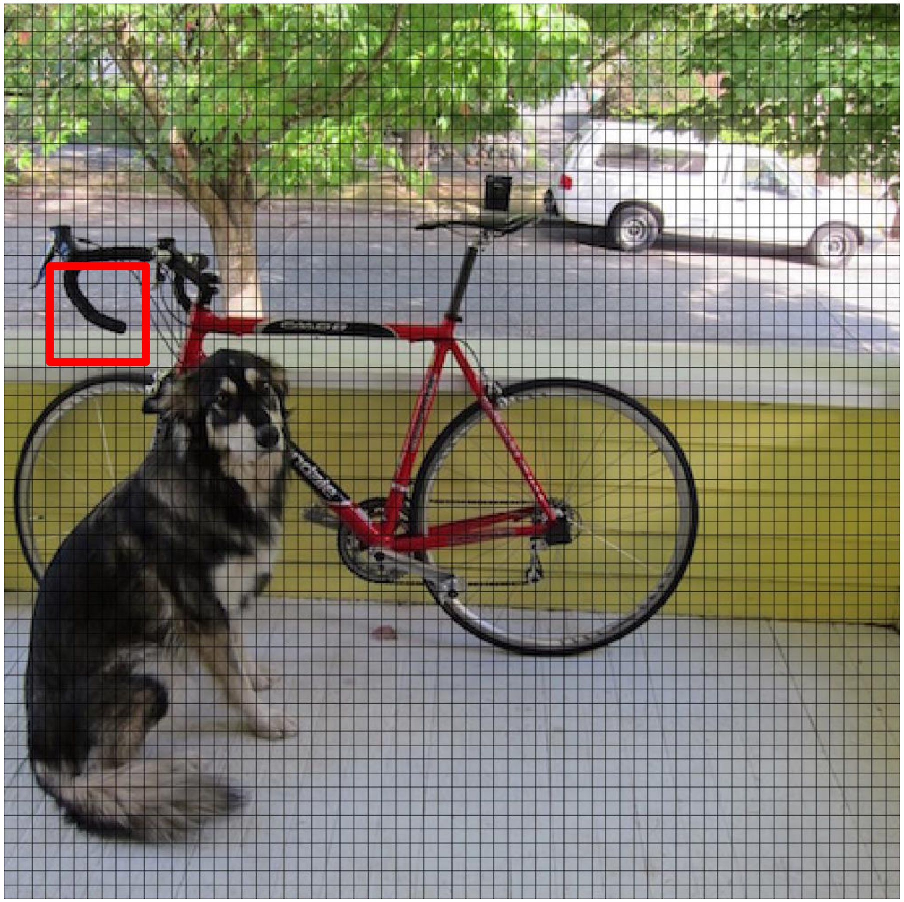


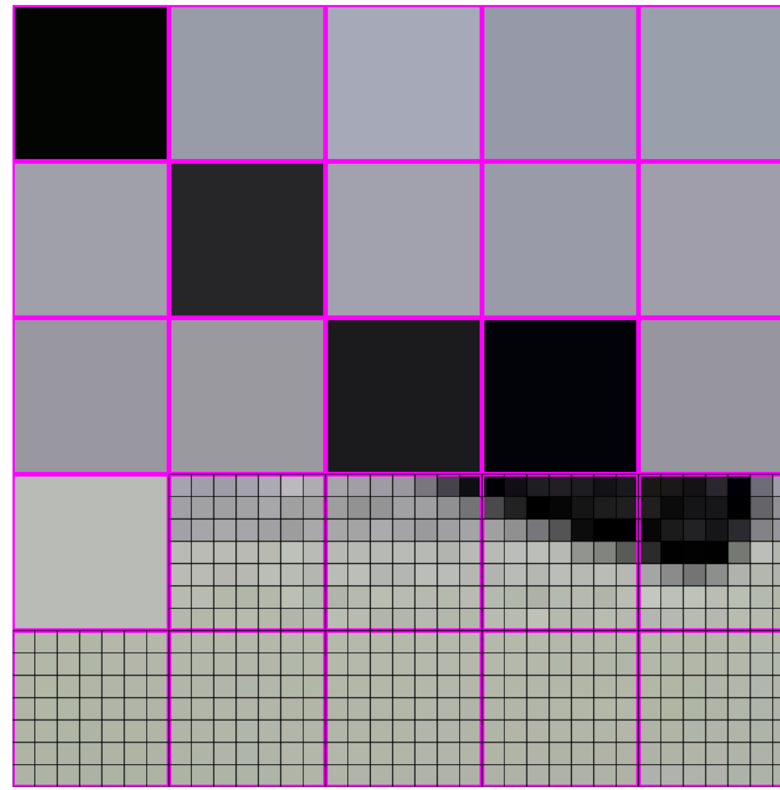
448x448 -> 64x64



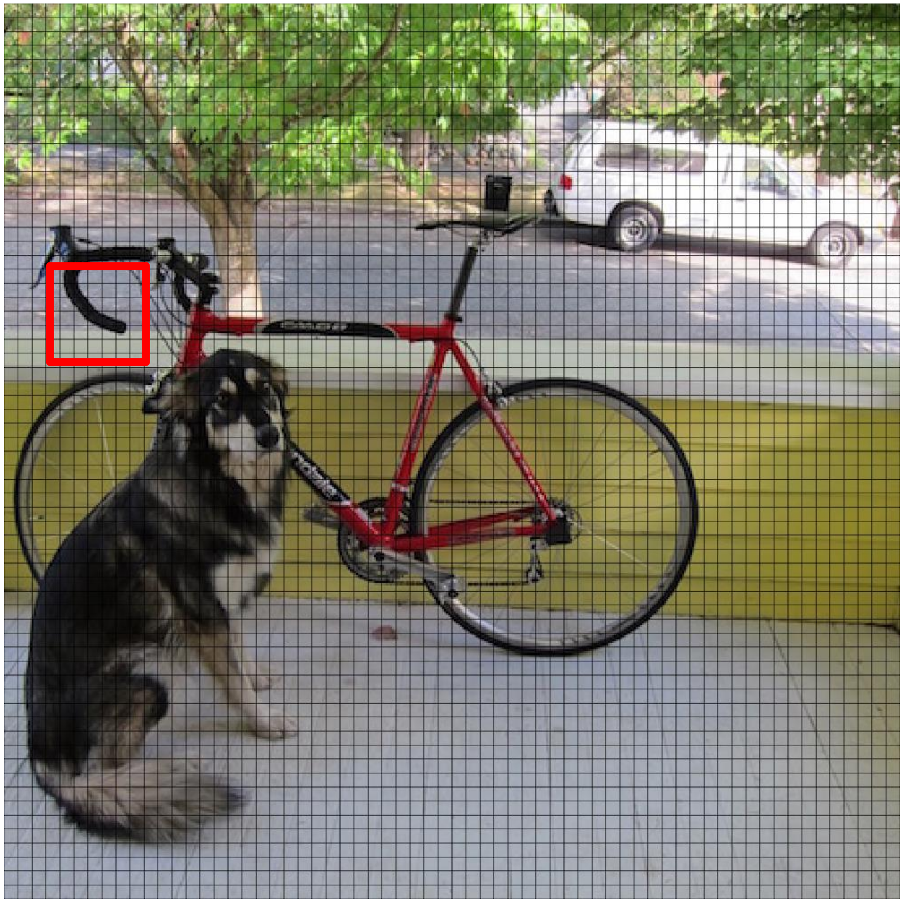


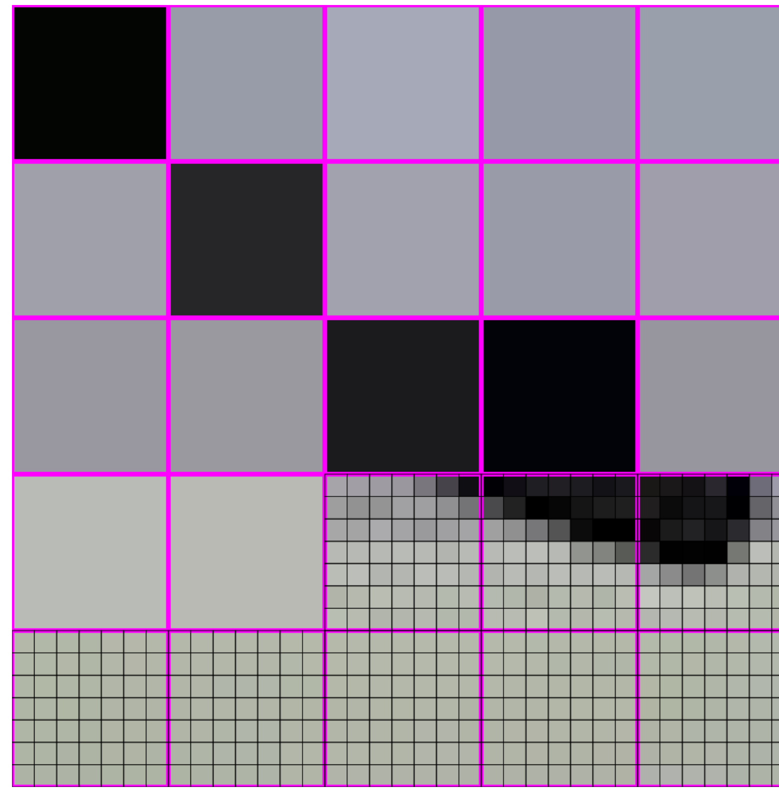
448x448 -> 64x64



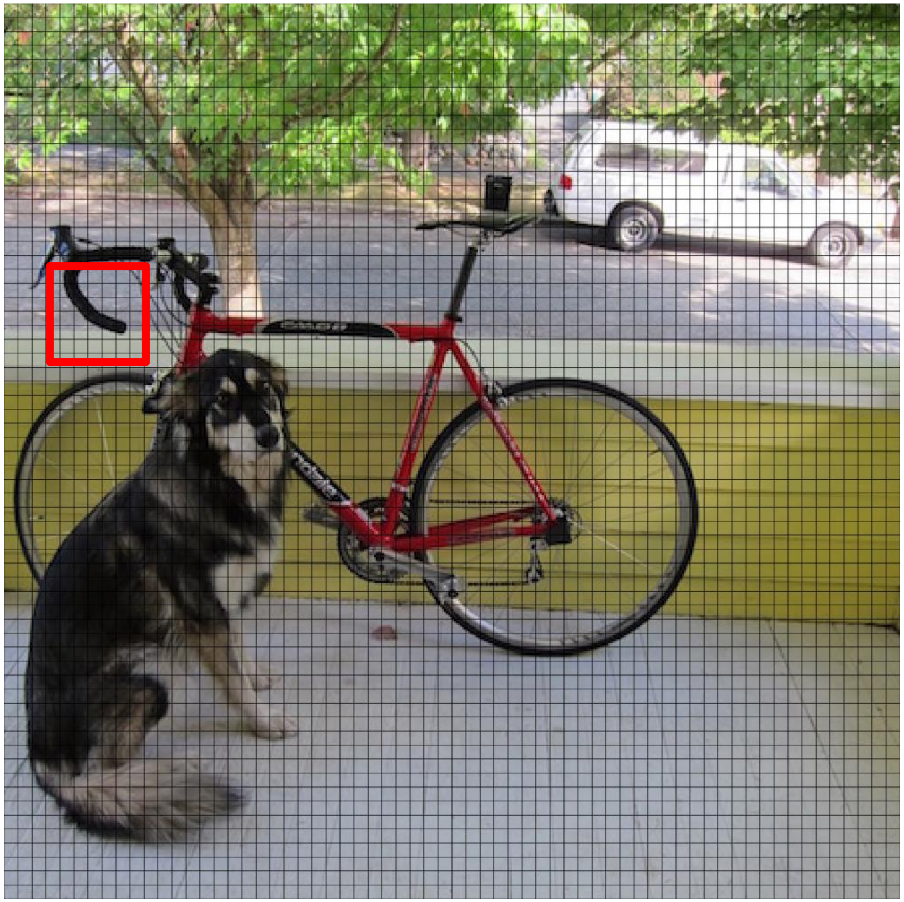


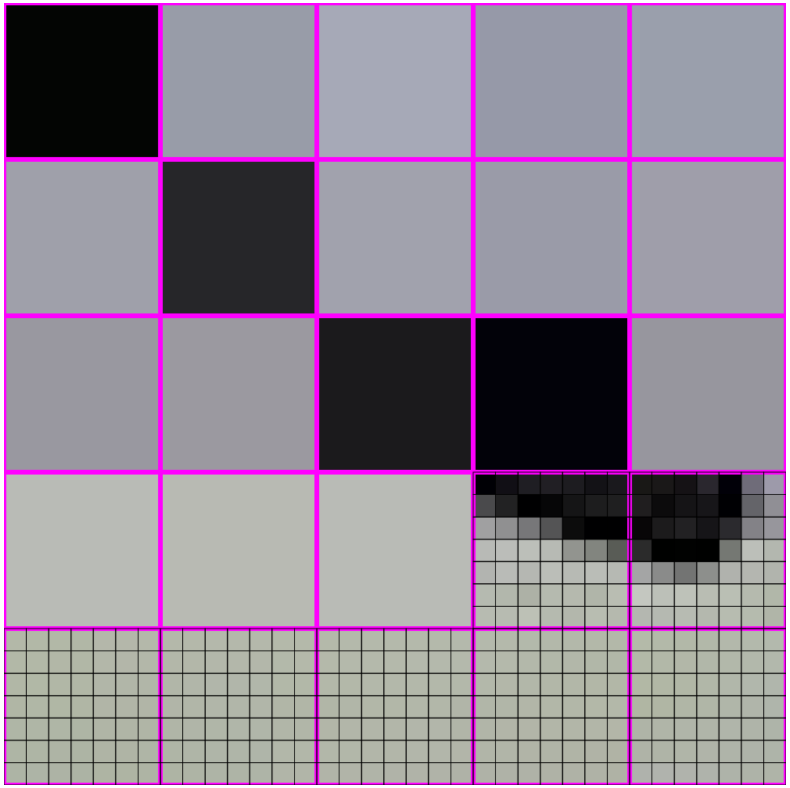
448x448 -> 64x64



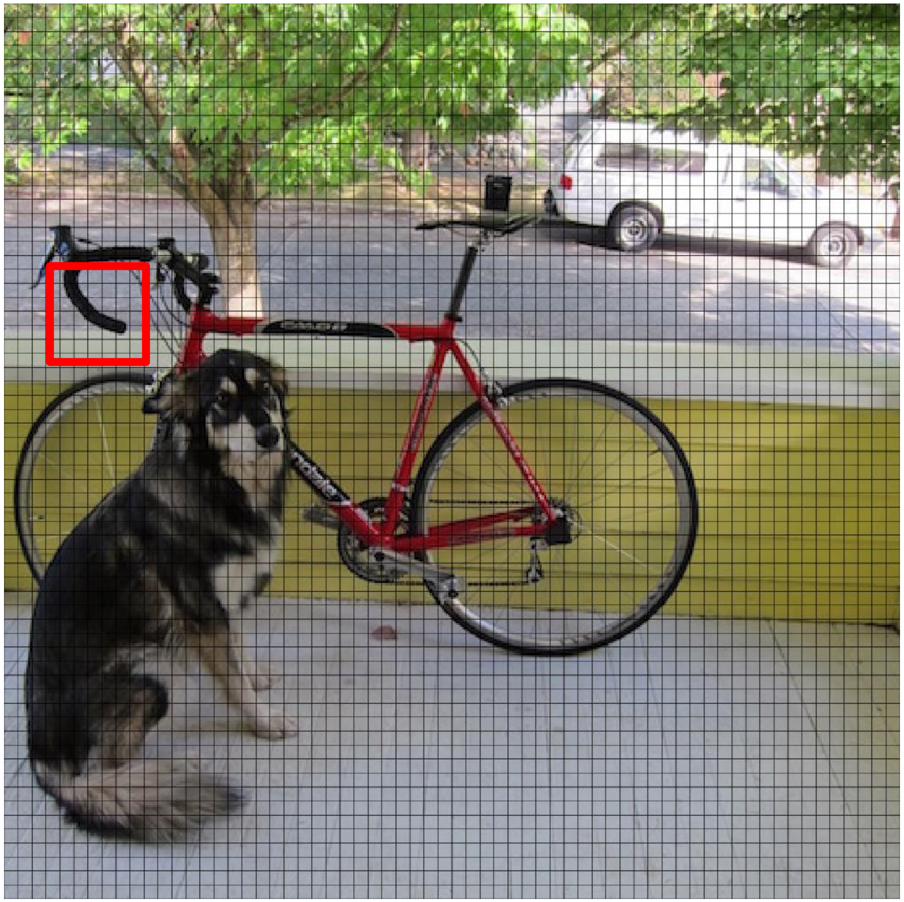


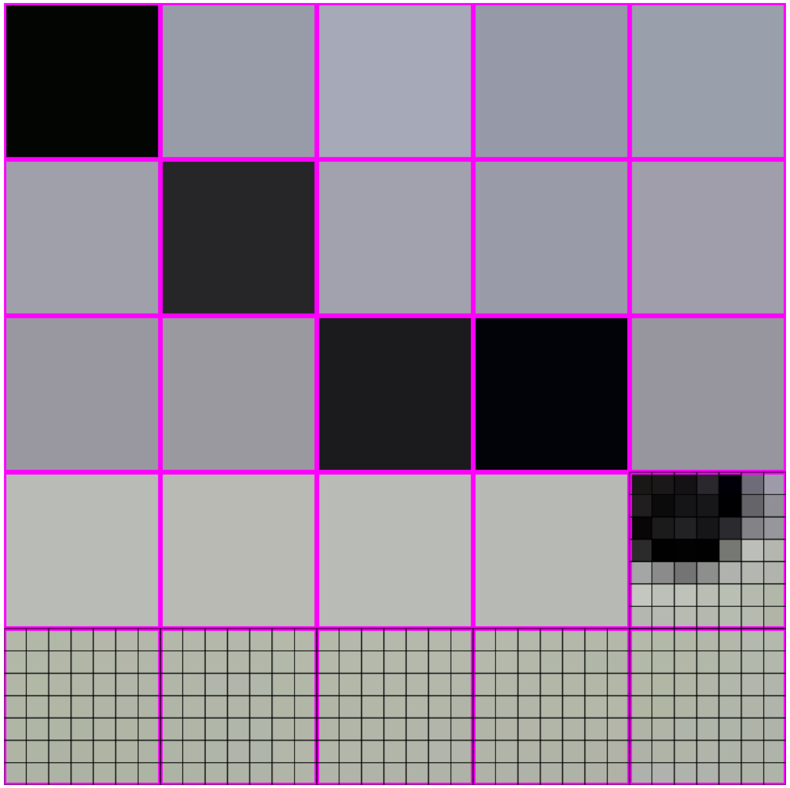
448x448 -> 64x64



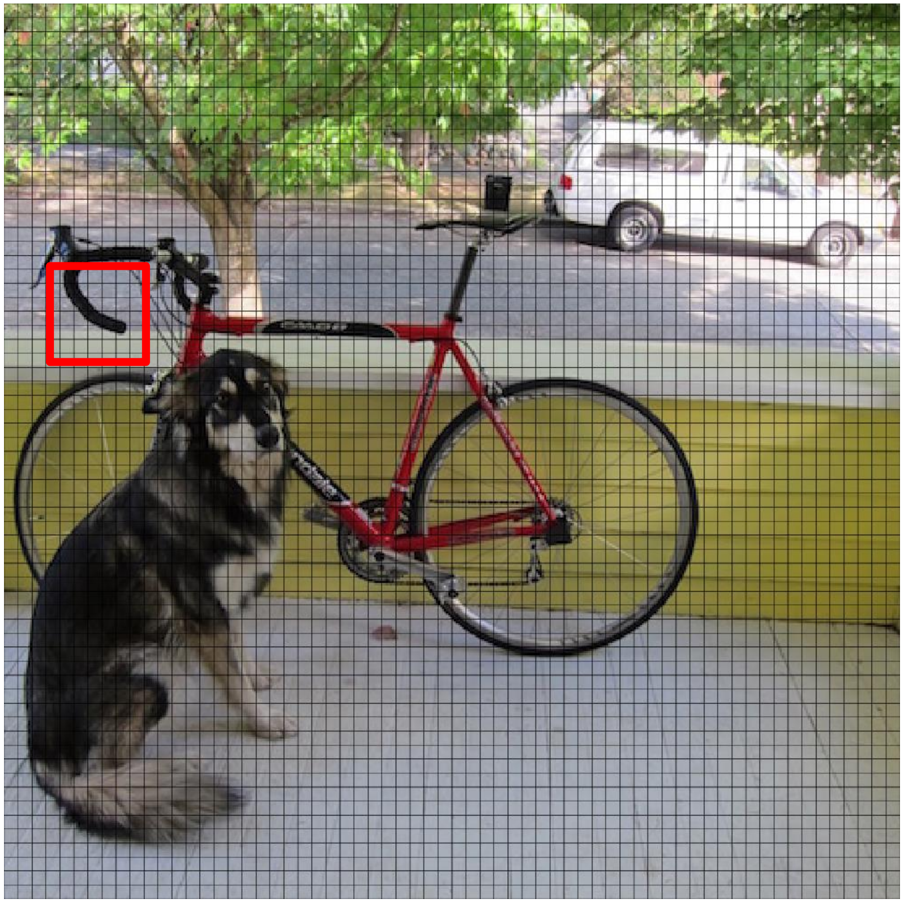


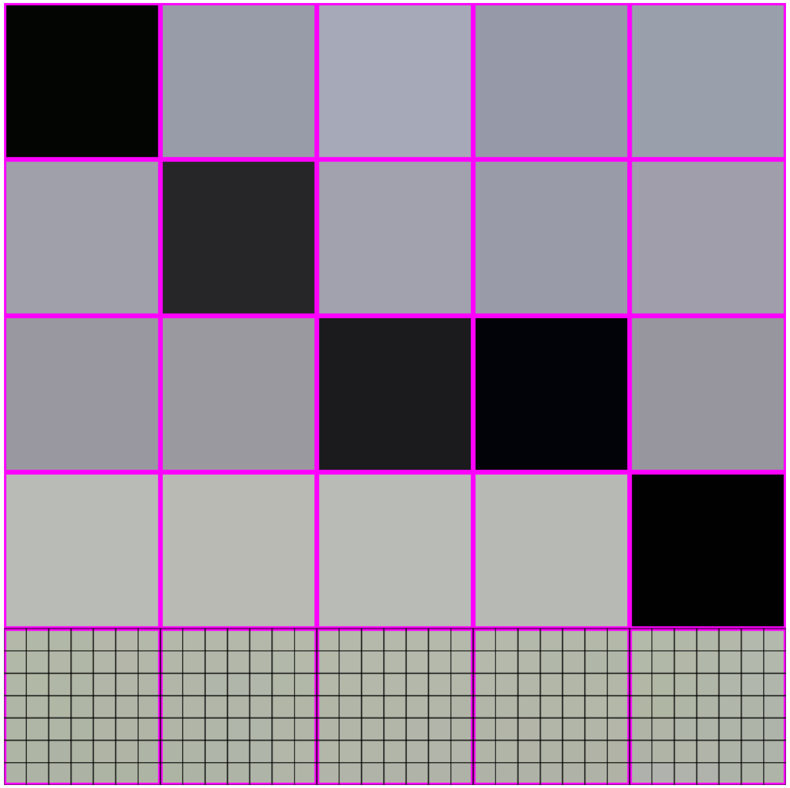
448x448 -> 64x64



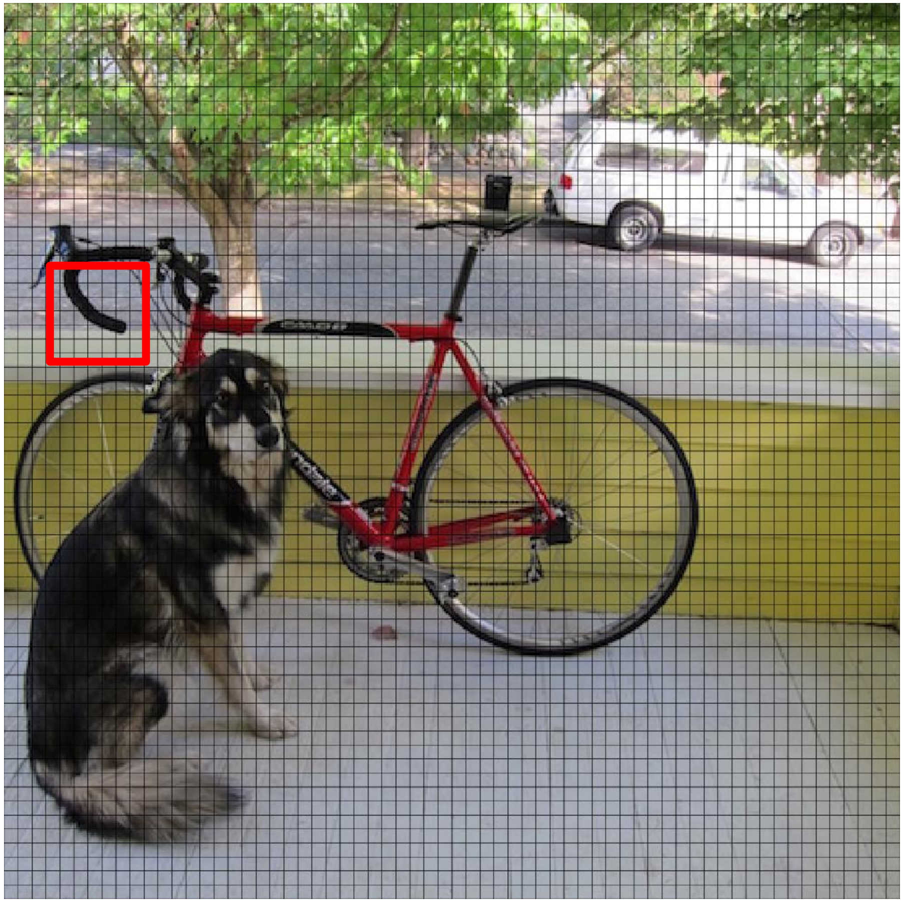


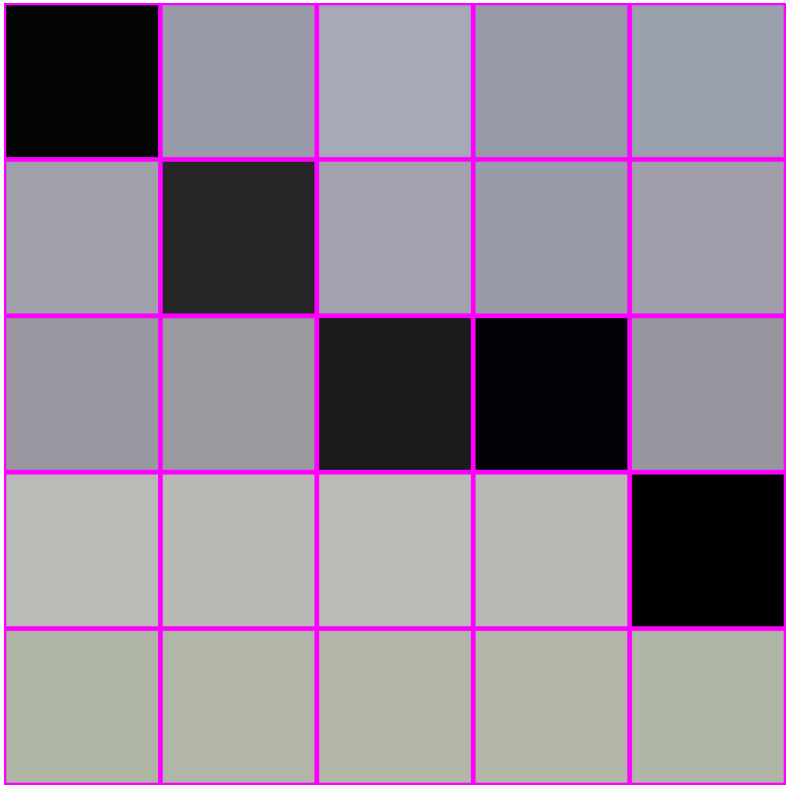
448x448 -> 64x64



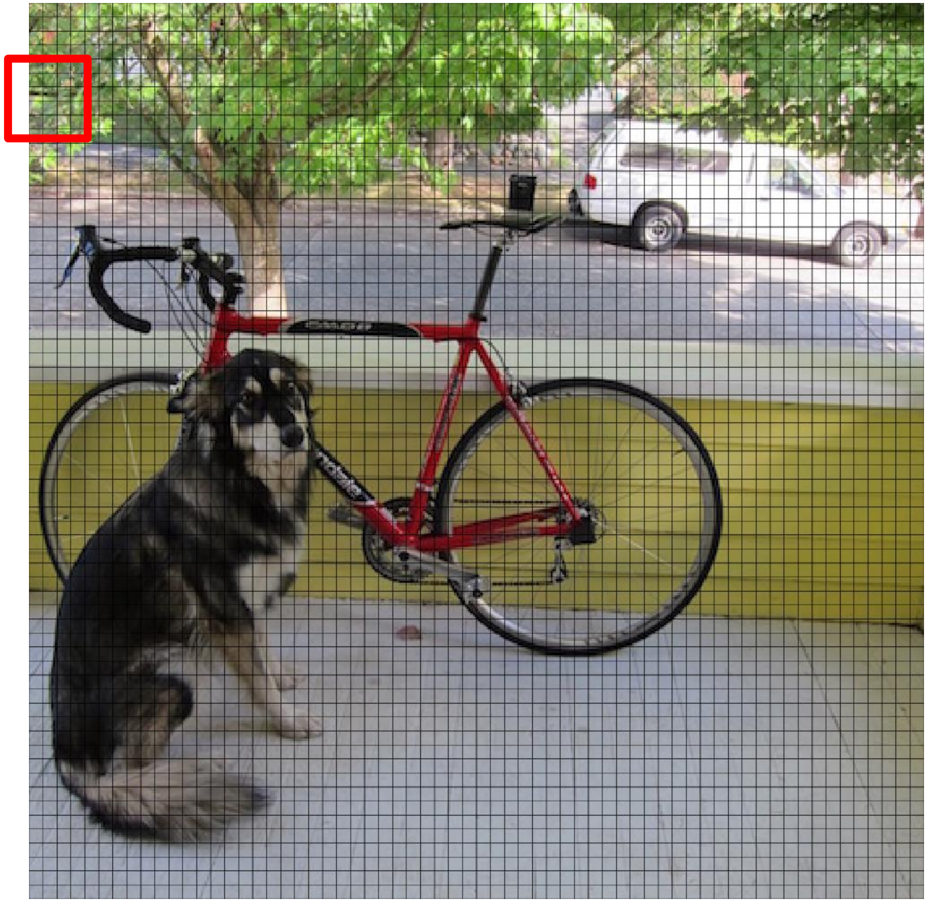


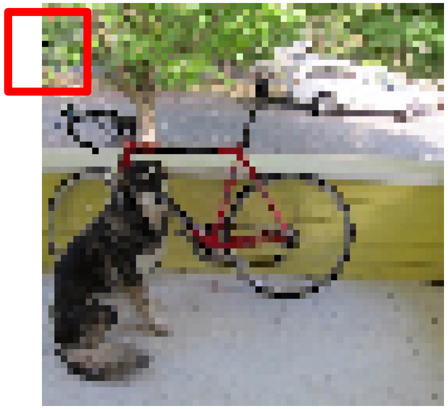
448x448 -> 64x64



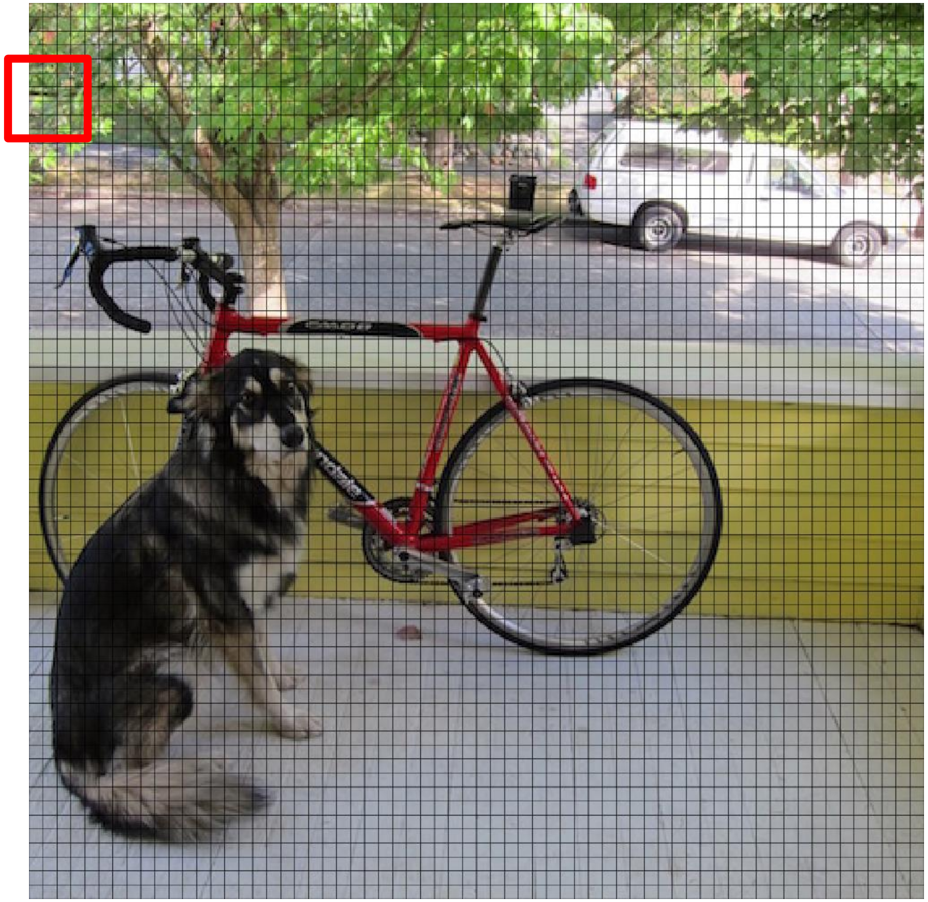


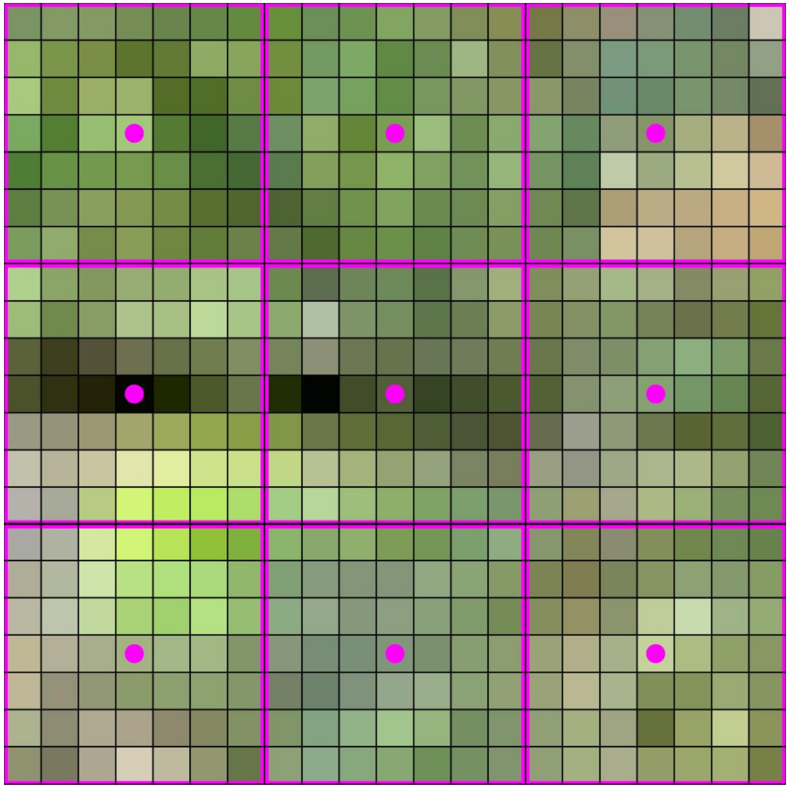
448x448 -> 64x64



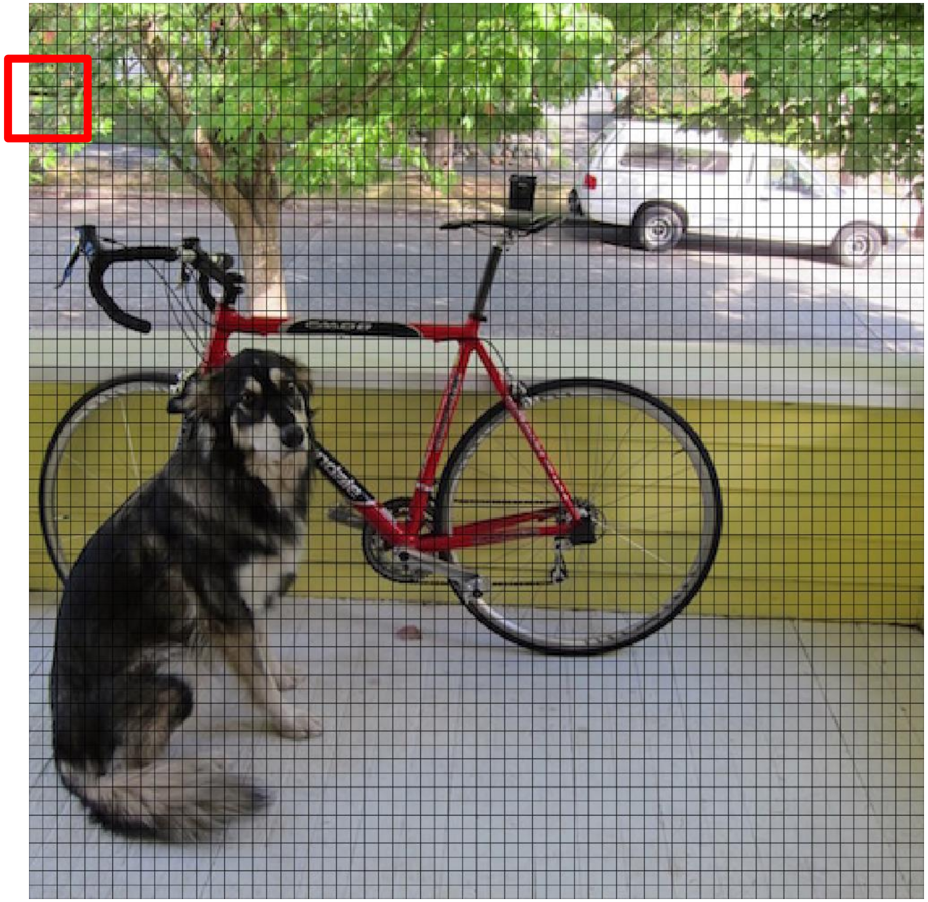


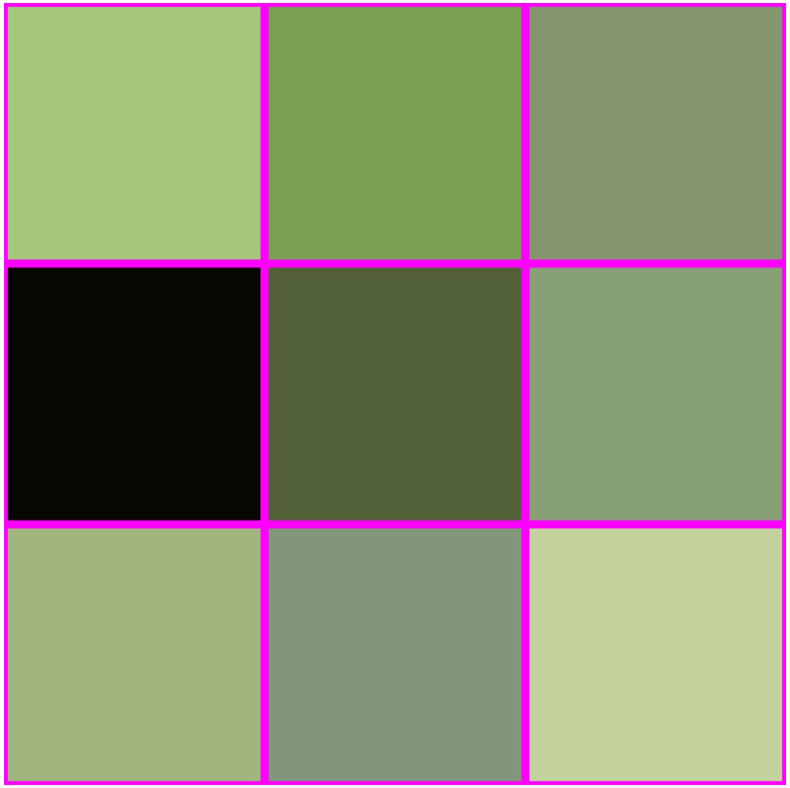
448x448 -> 64x64



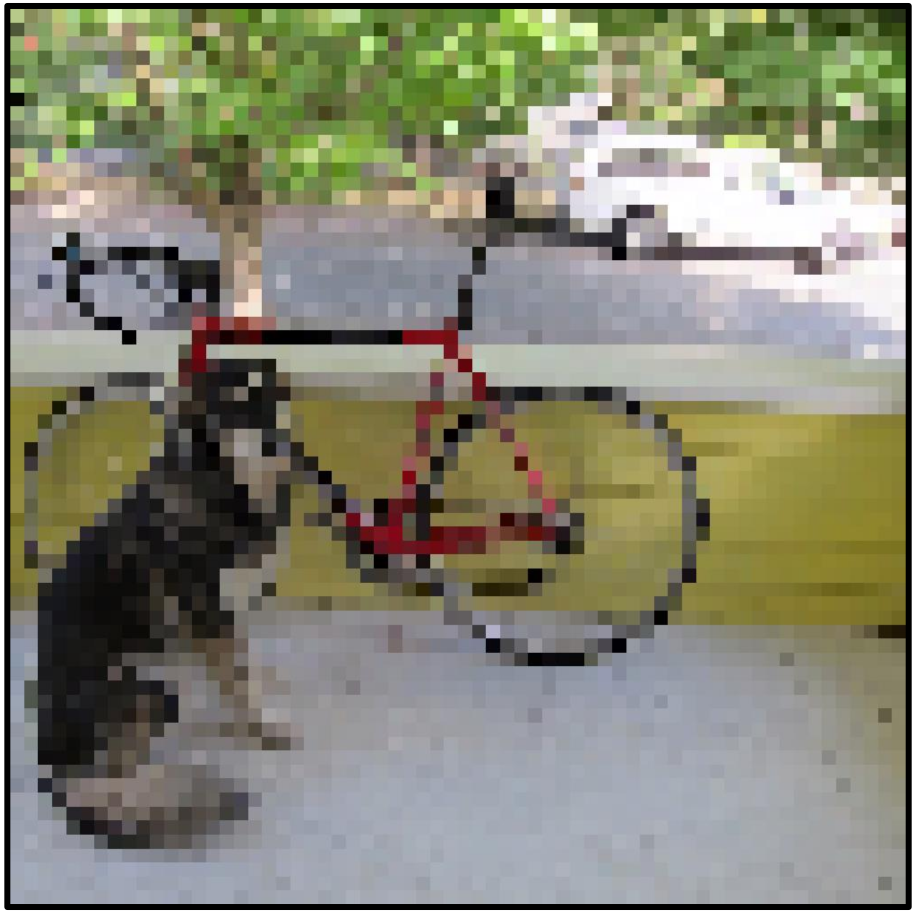


448x448 -> 64x64





IS THIS ALL THERE IS??



THERE IS A BETTER WAY!

Next Time: Filtering