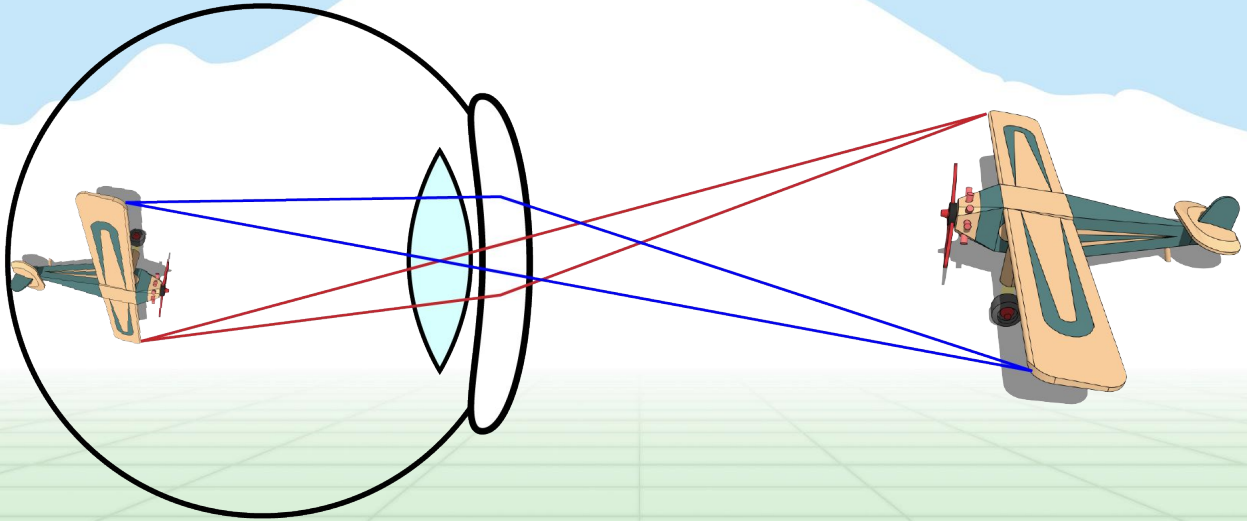
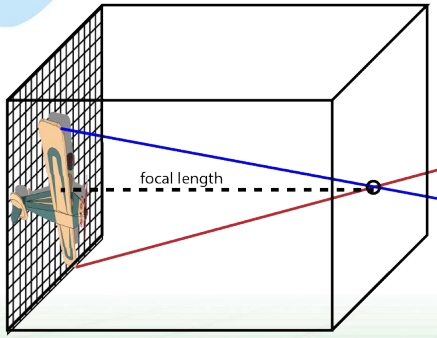

The Ancient
Secrets
Computer Vision

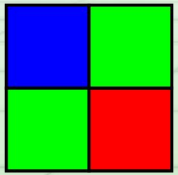

Previously
On
the
Ancient Secrets
of Computer Vision

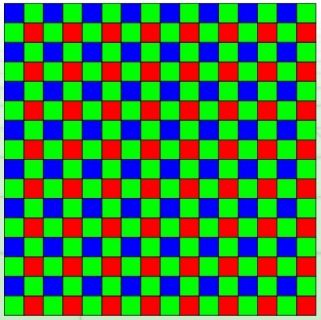
Eyes: projection onto retina

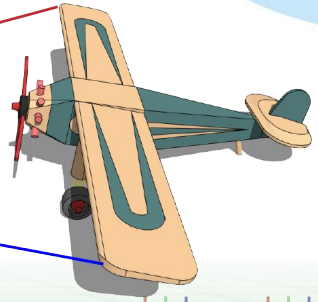


Camera: projection onto sensor array









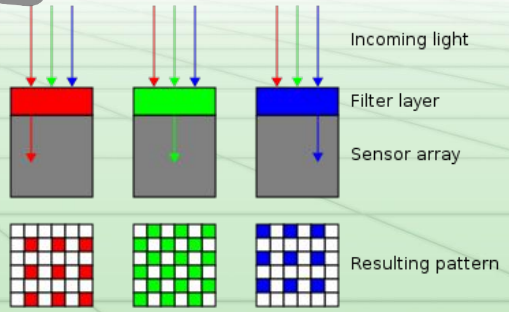
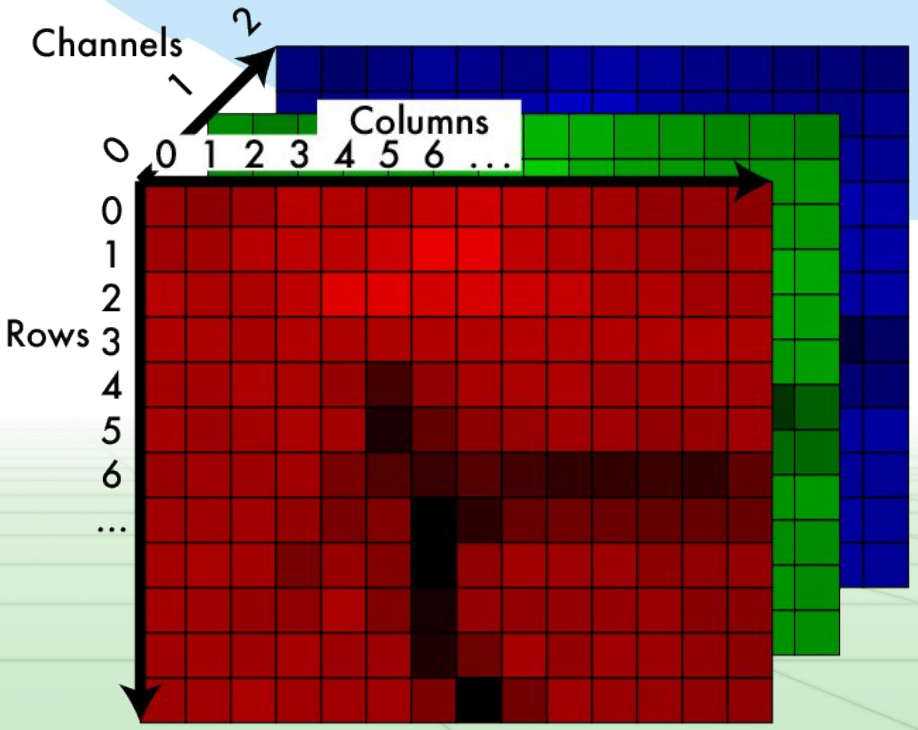


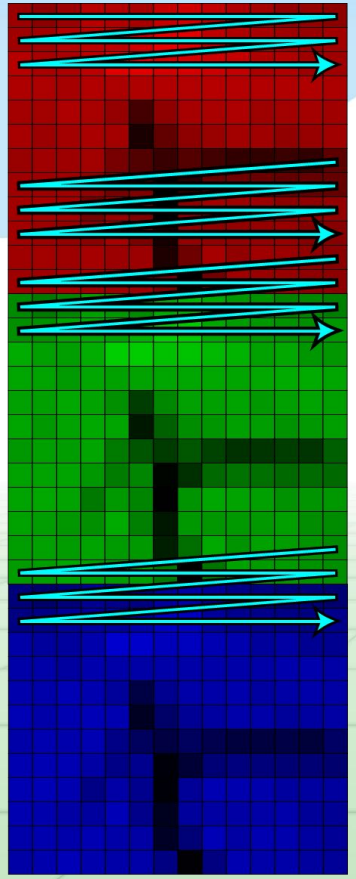
Image: tensor of light values

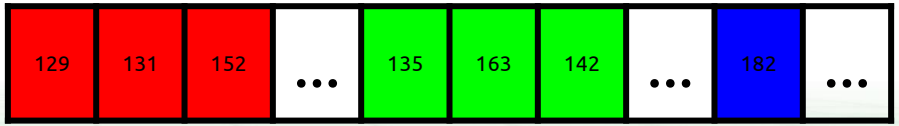
Addressing pixels:

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

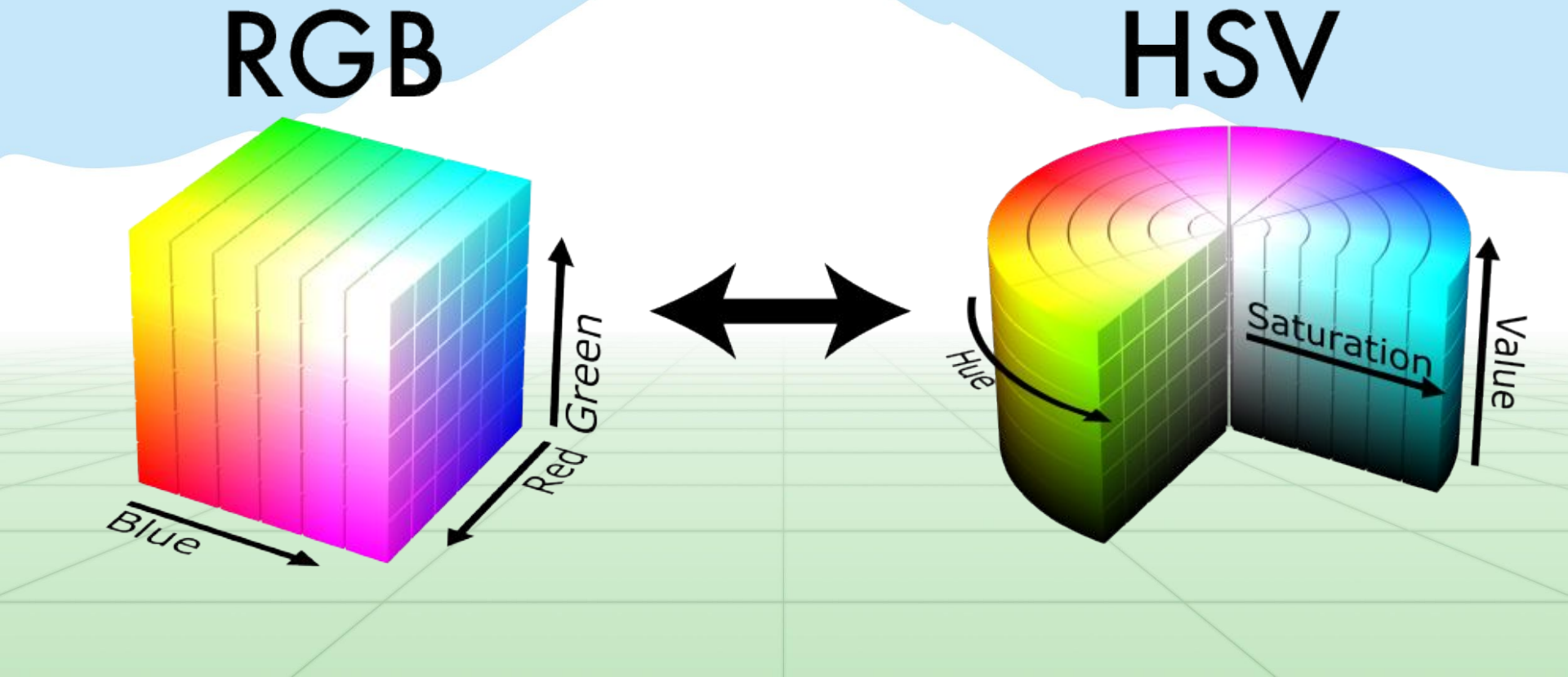


CHW for storing in 1d array

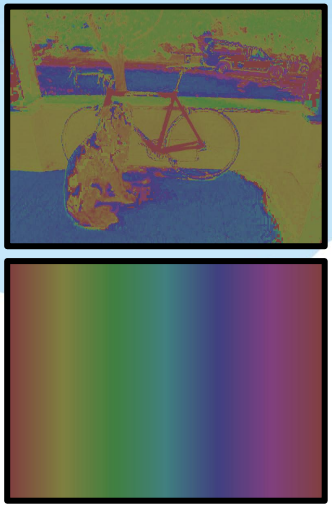




Other colorspaces are fun!



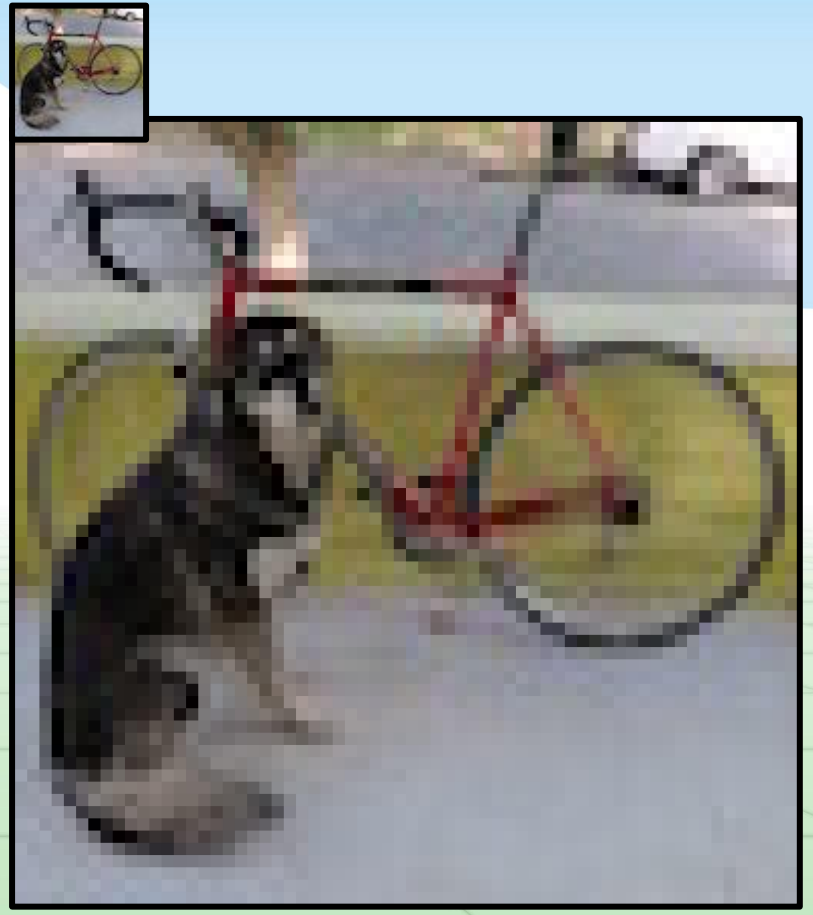
Or pattern...



Nearest neighbor: what it sounds like

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

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

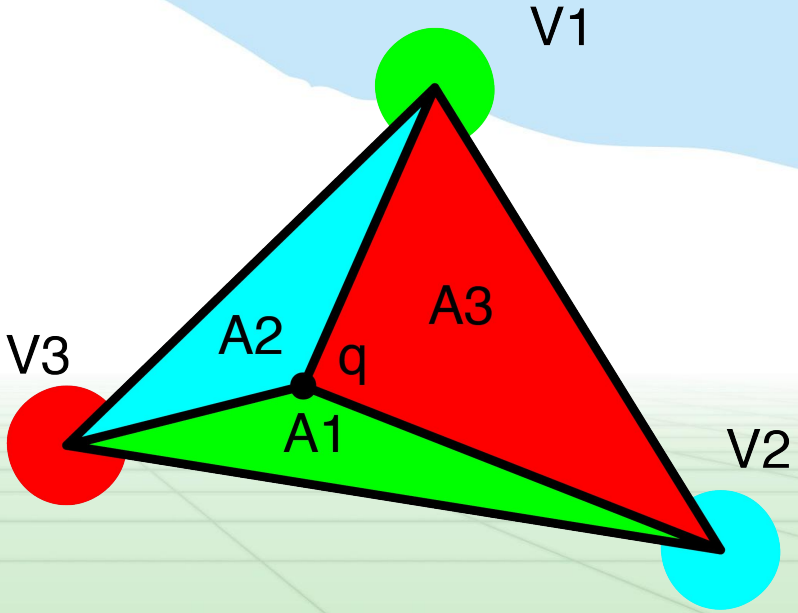


Triangle interpolation: for less structured image

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

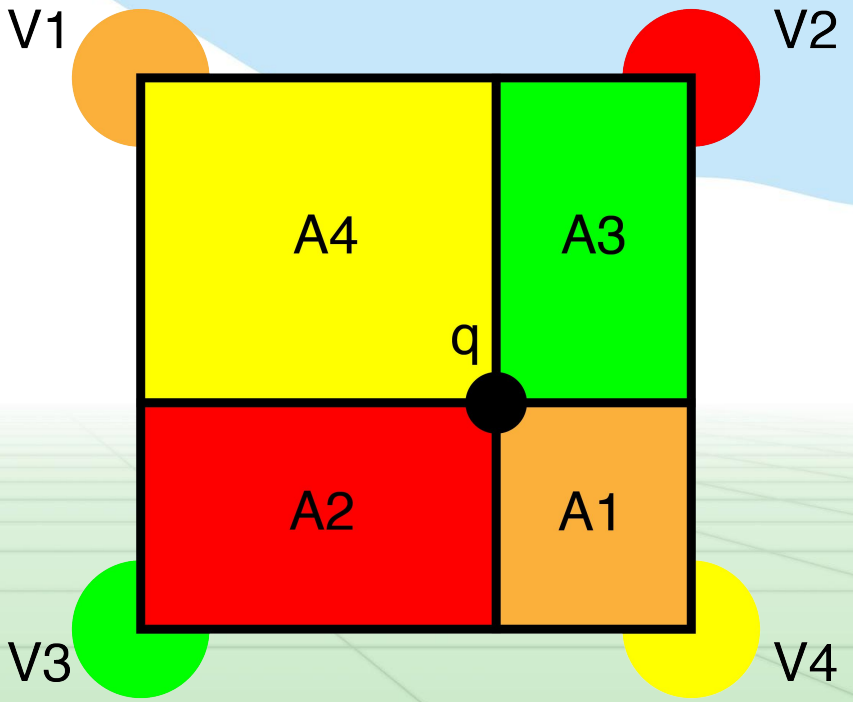
This time find the closest pixels in a box

Same plan, weighted sum based on area of opposite rectangle

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

Still need to normalize!

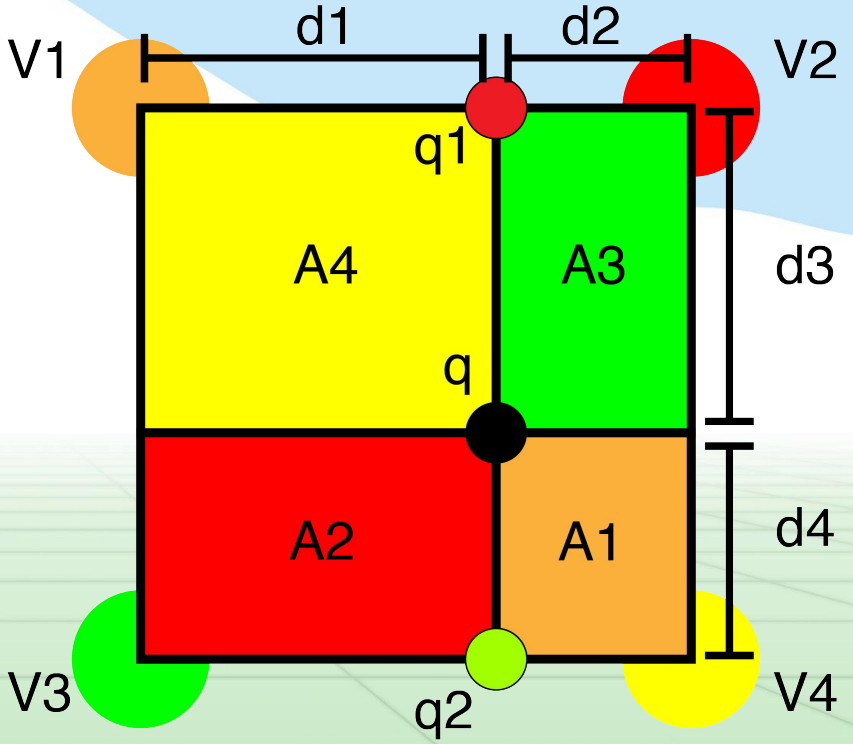
Or do we?



Bilinear interpolation: for grids, pretty good

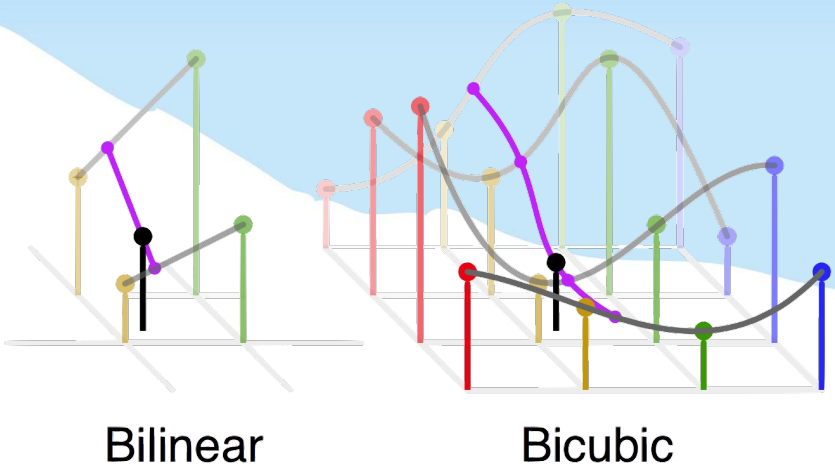
Alternatively, linear interpolation of linear interpolates

```
q1 = V1*d2 + V2*d1
q2 = V3*d2 + V4*d1
q = q1*d4 + q2*d3
```



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…

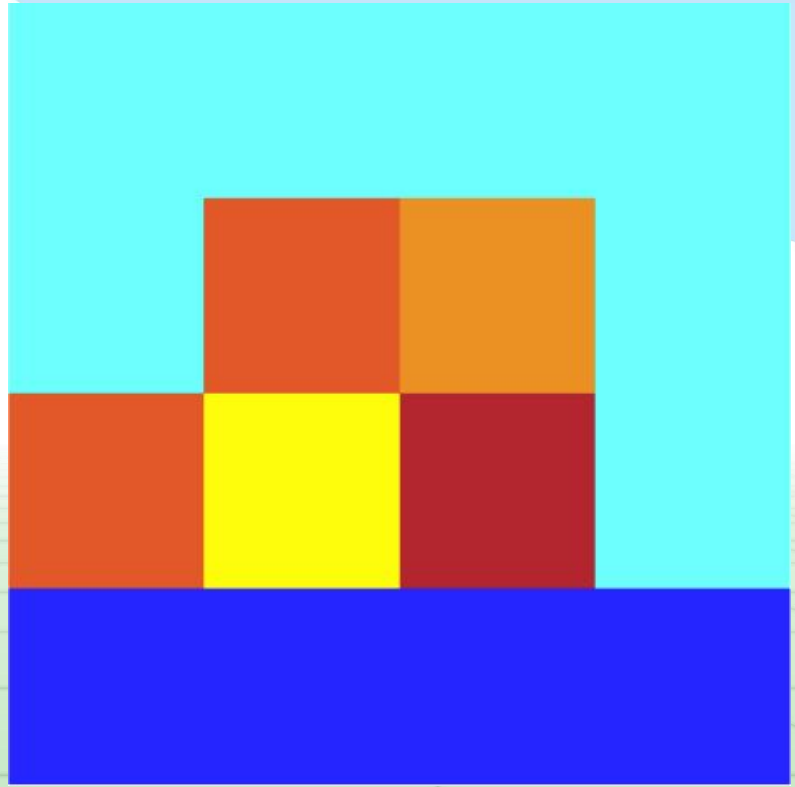
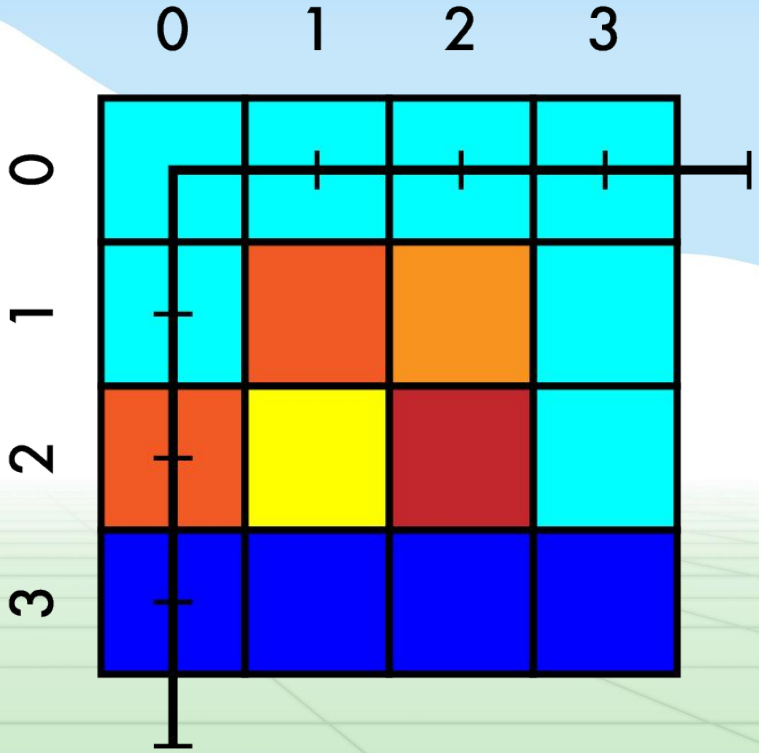


Image resizing!

Say we want to increase the size of an image…

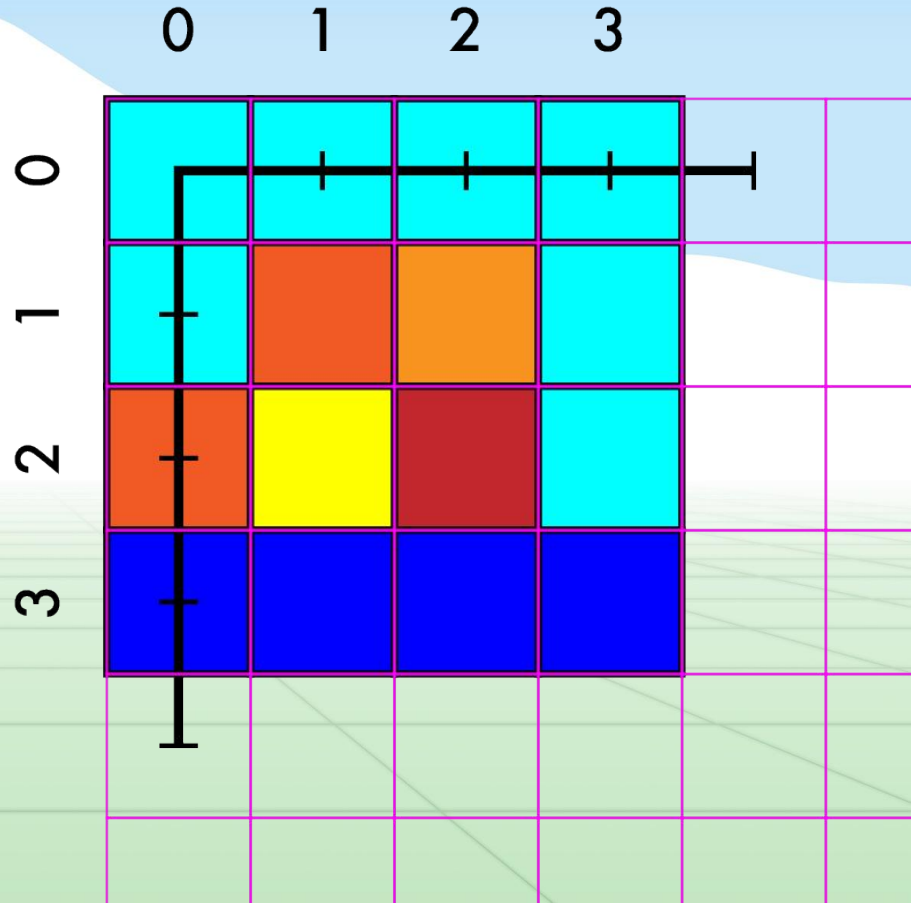
This is a beautiful image of a sunset… it’s just very small…

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



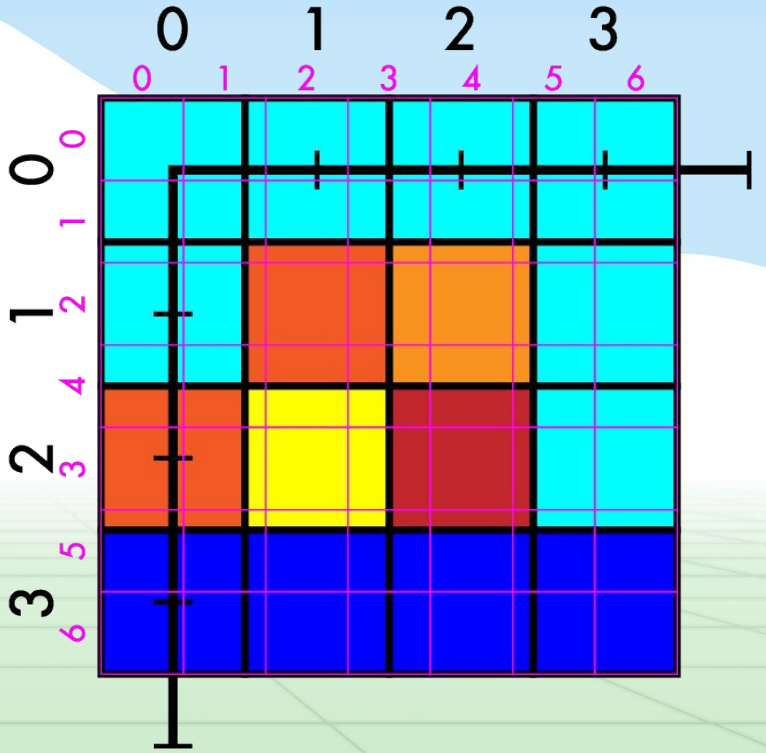
Resize 4x4 -> 7x7

- Create our new image



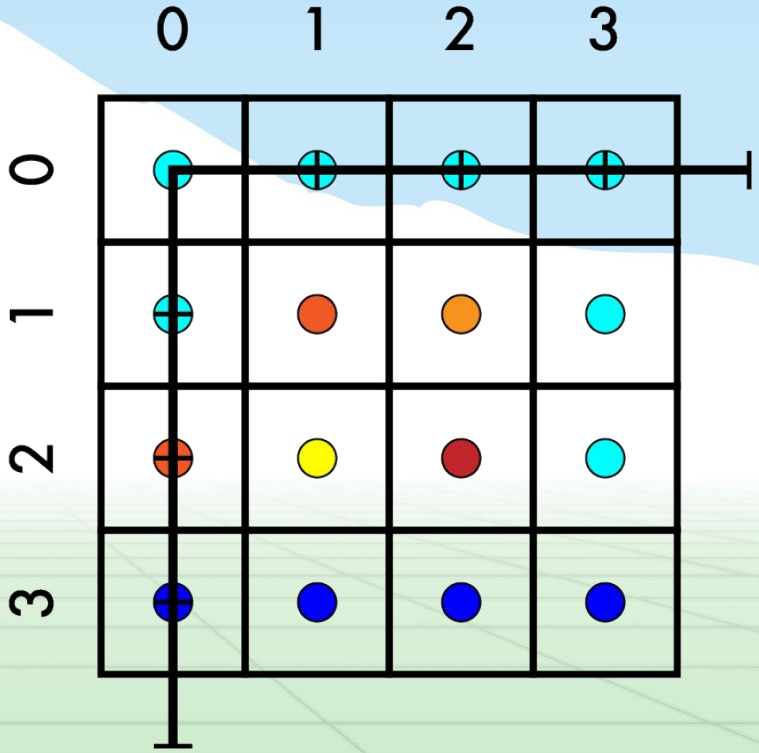
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates



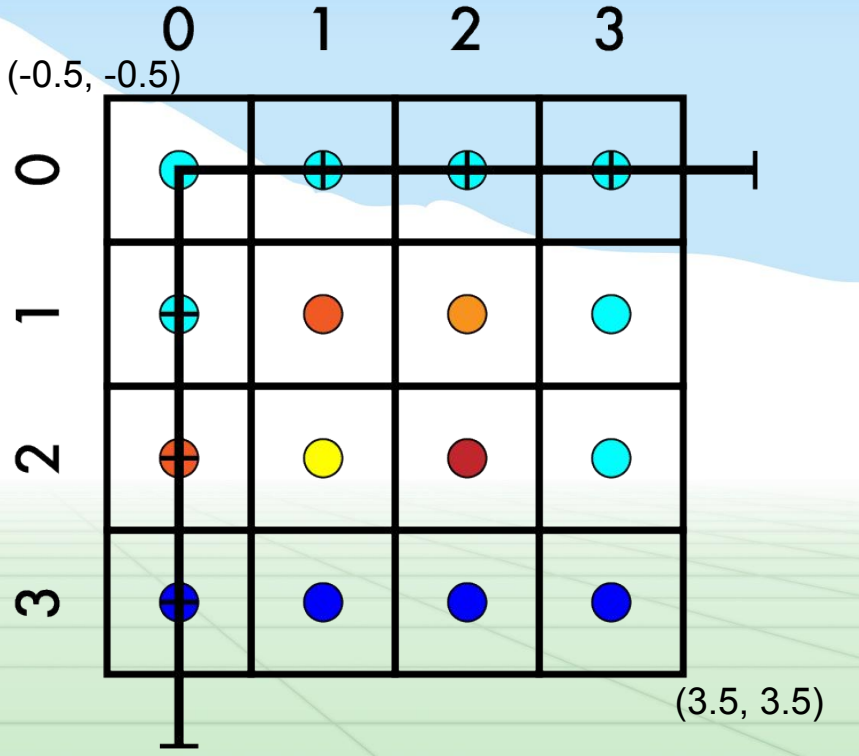
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates



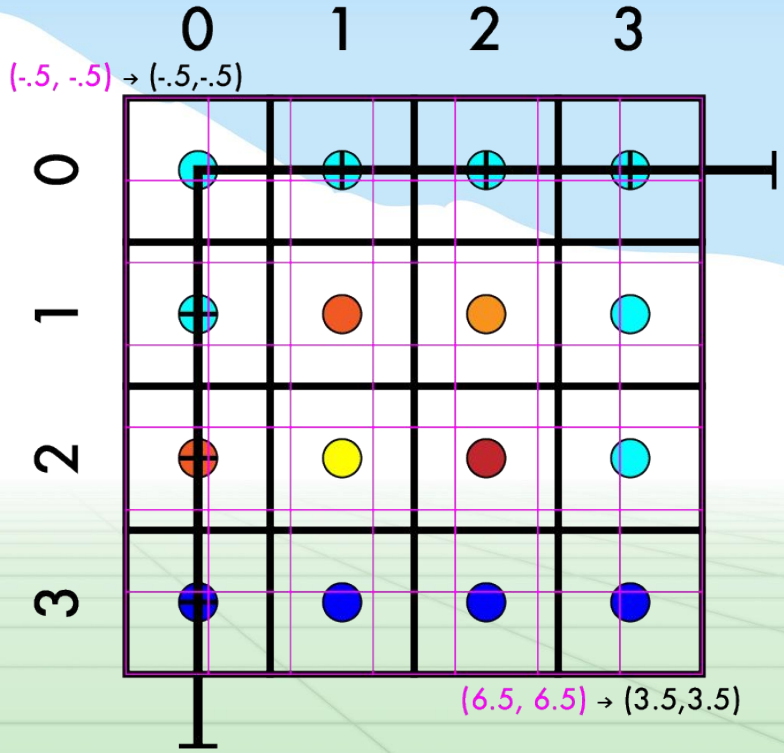
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates



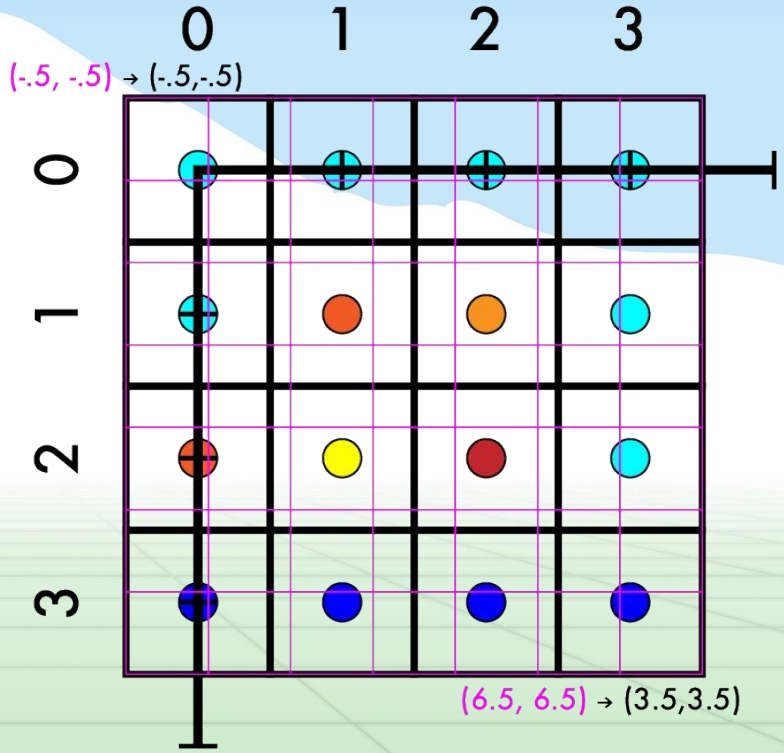
Resize 4x4 -> 7x7

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



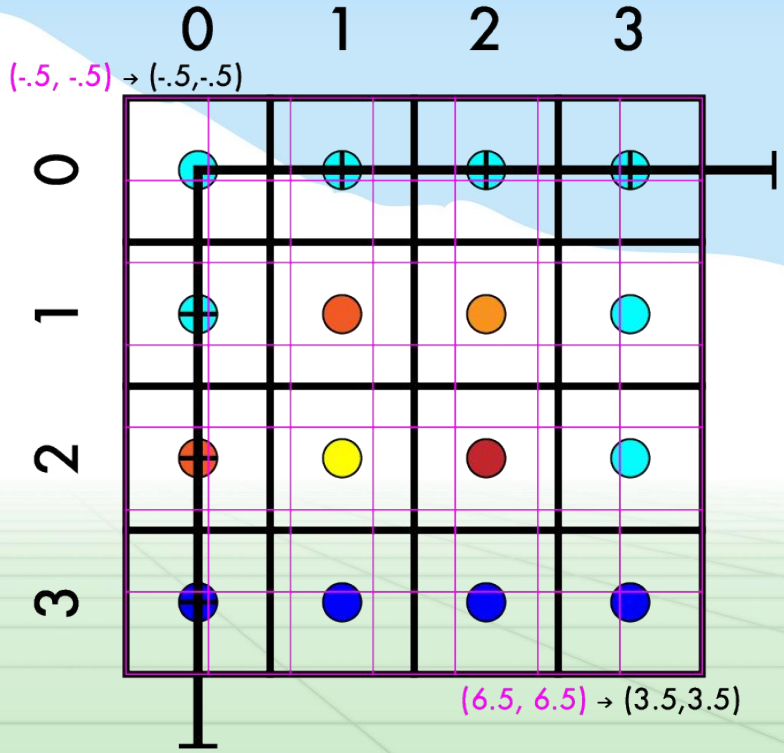
Resize 4x4 -> 7x7

- 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



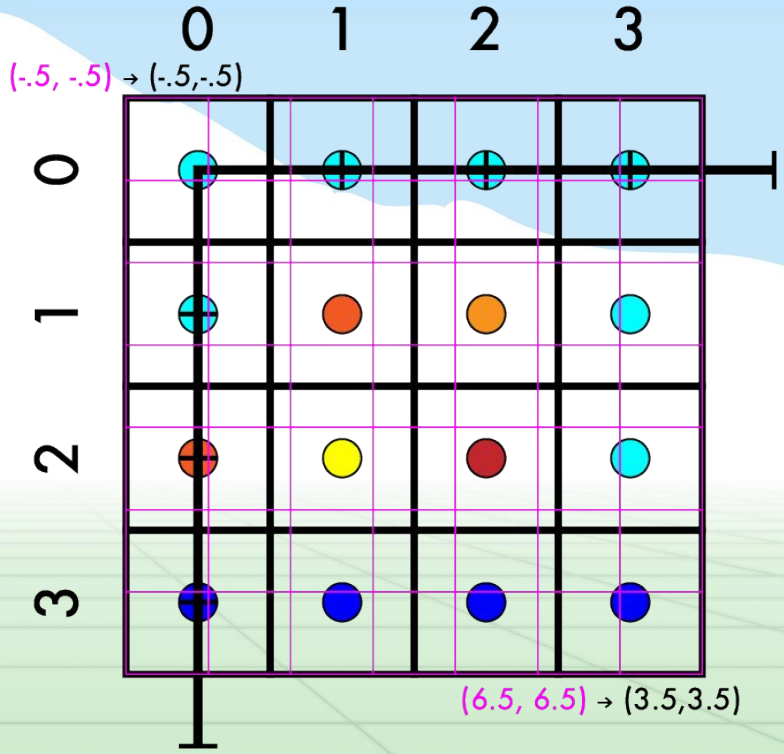
Resize 4x4 -> 7x7

- 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



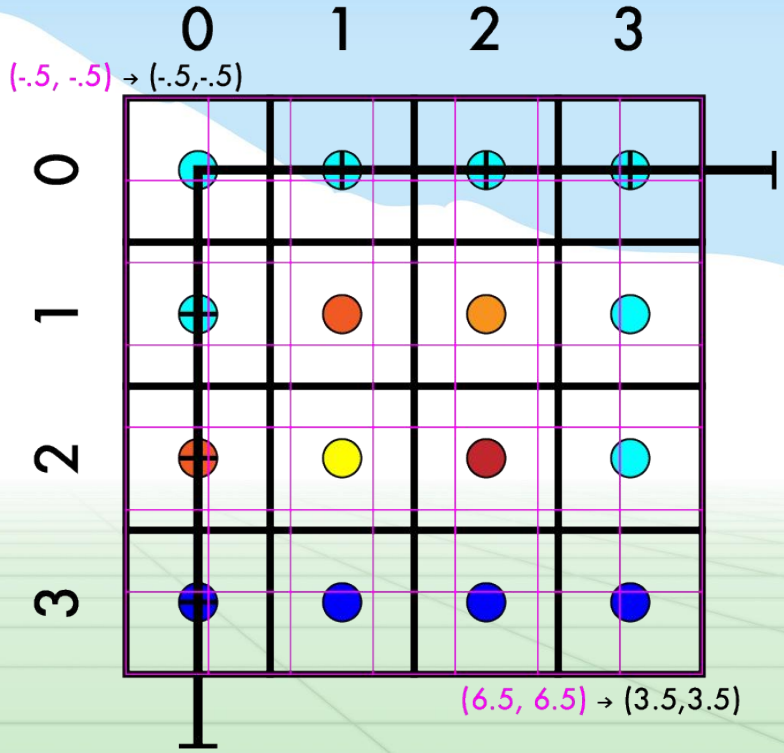
Resize 4x4 -> 7x7

- 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



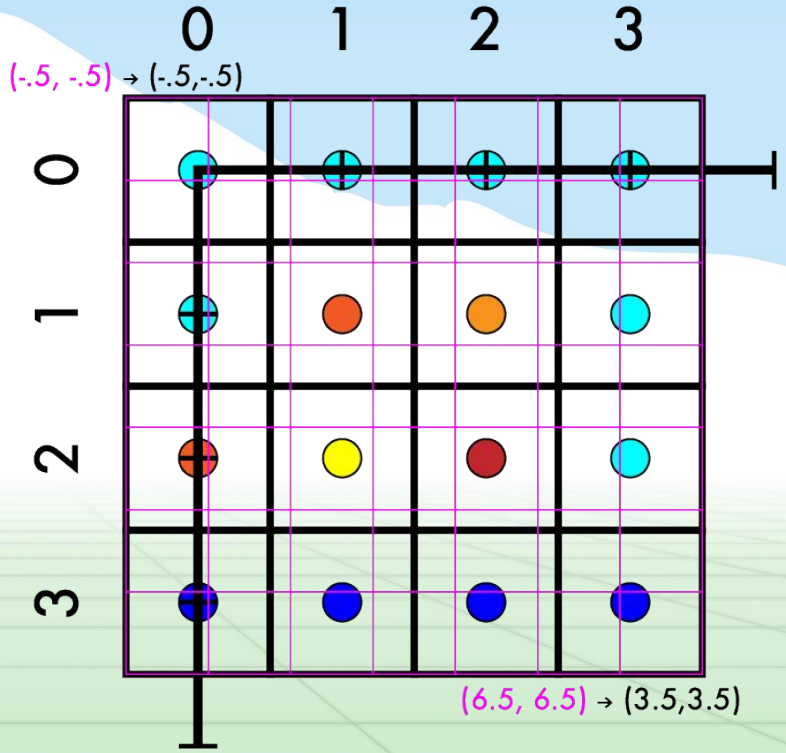
Resize 4x4 -> 7x7

- 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



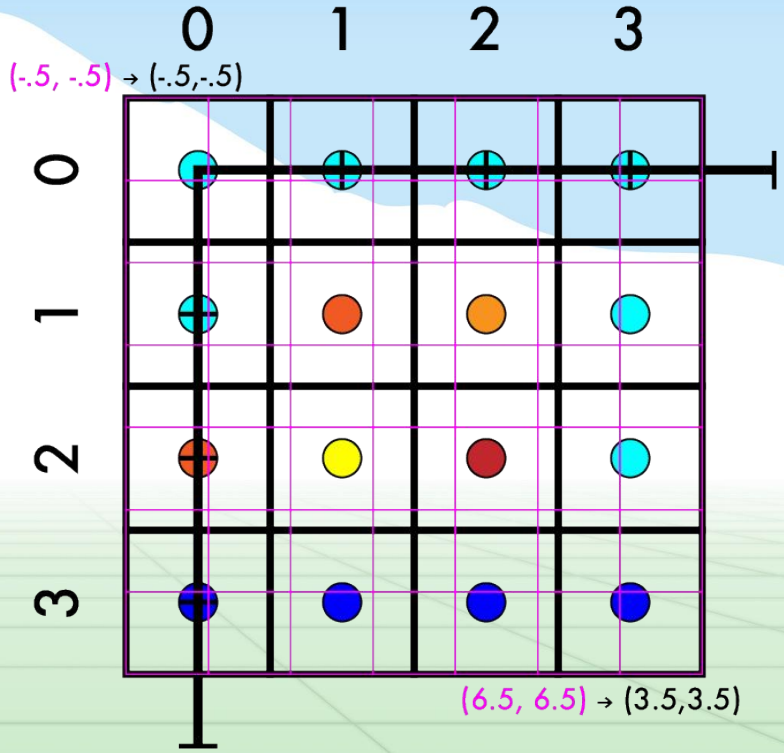
Resize 4x4 -> 7x7

- 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



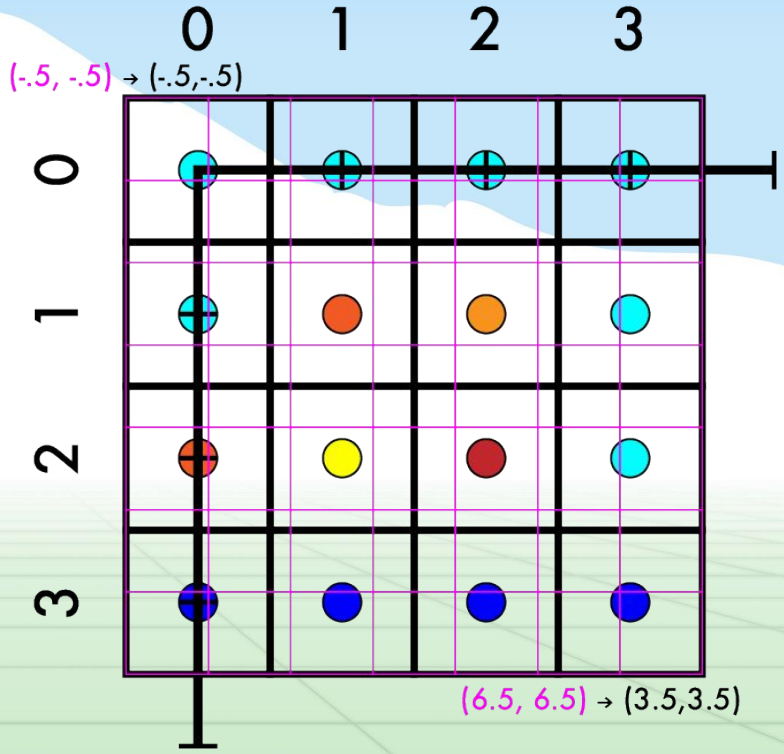
Resize 4x4 -> 7x7

- 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



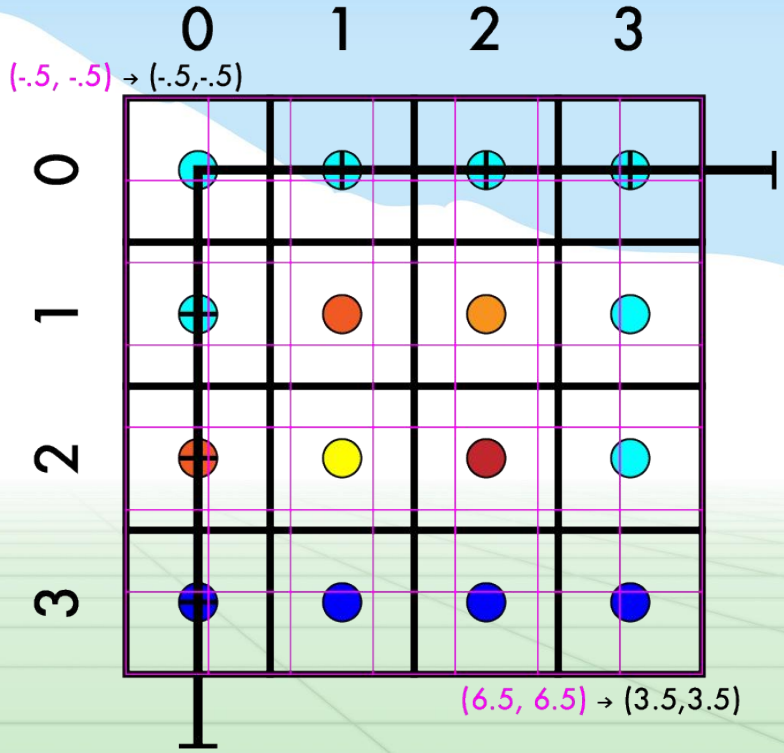
Resize 4x4 -> 7x7

- 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



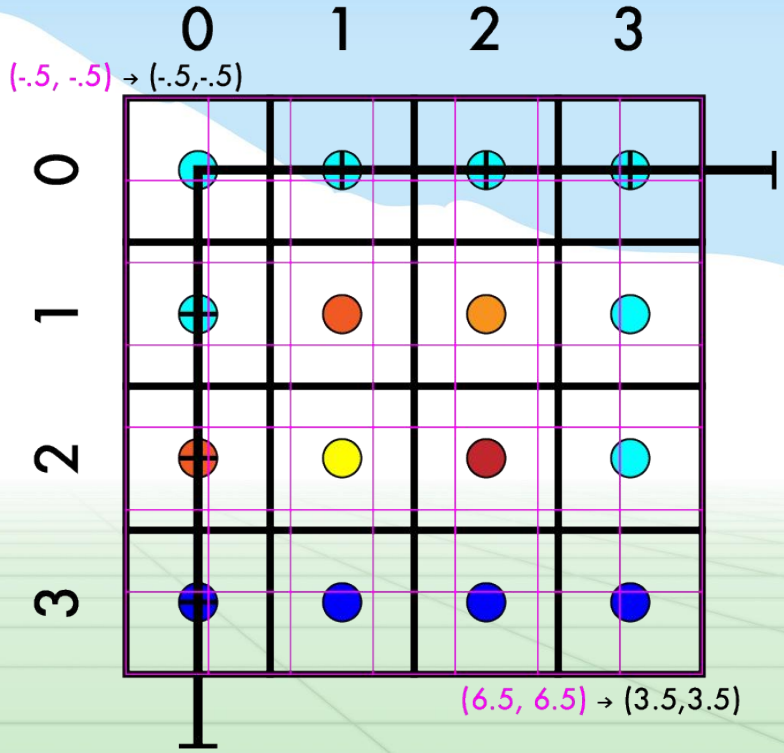
Resize 4x4 -> 7x7

- 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



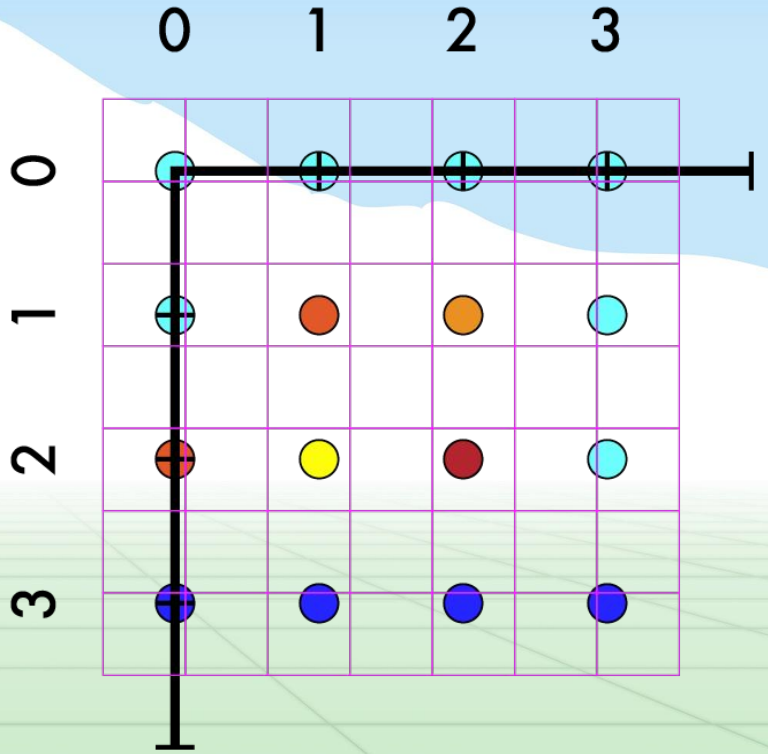
Resize 4x4 -> 7x7

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



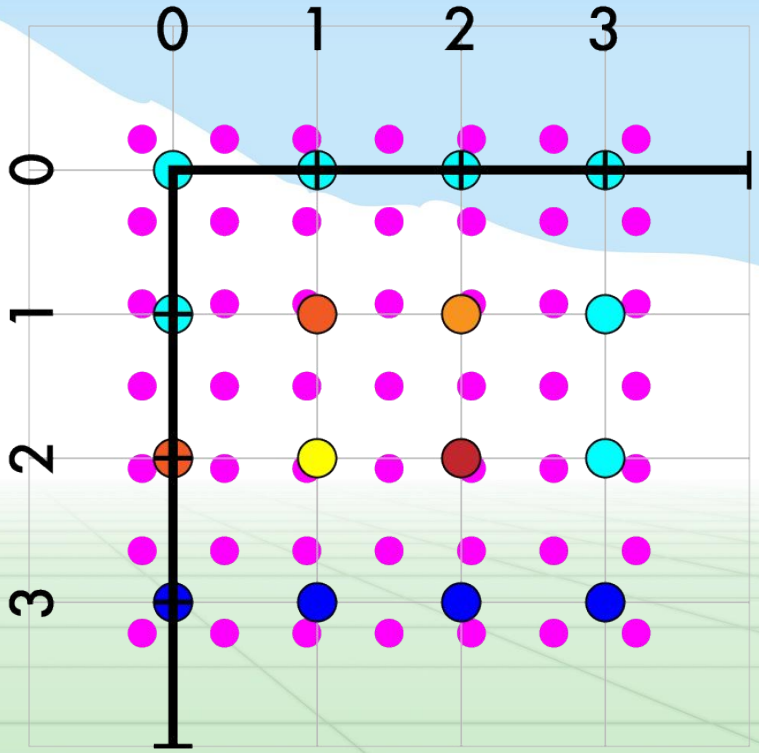
Resize 4x4 -> 7x7

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



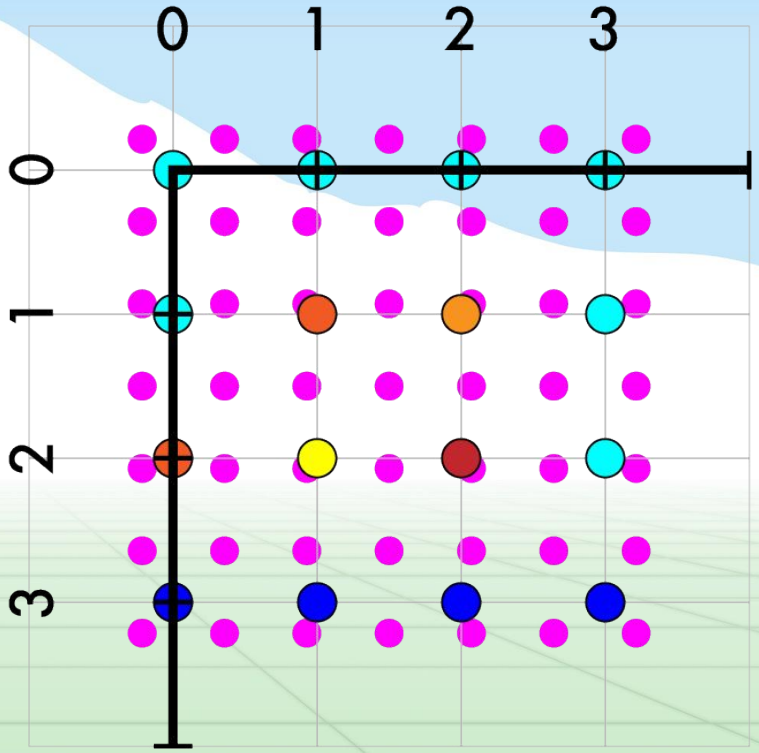
Resize 4x4 -> 7x7

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



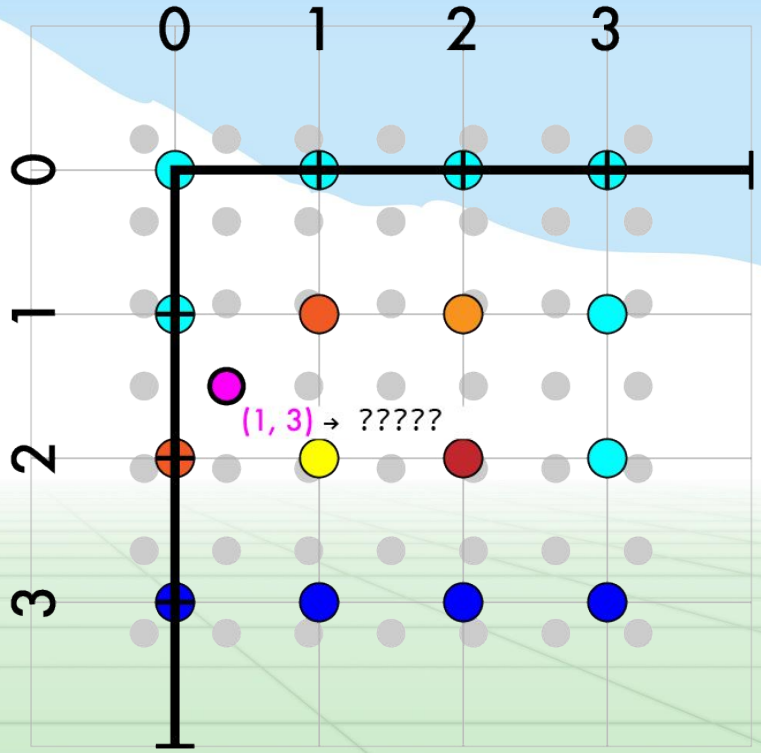
Resize 4x4 -> 7x7

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



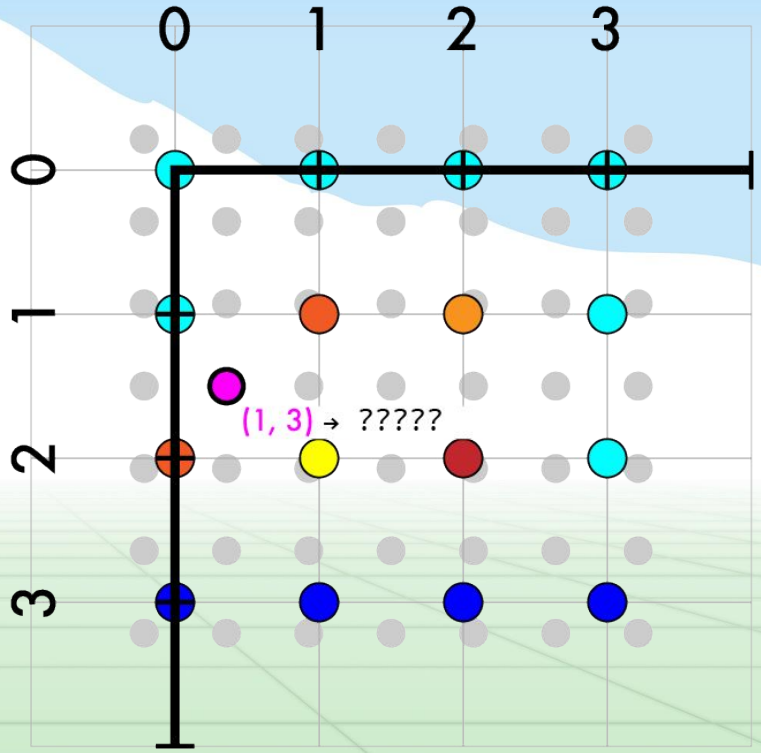
Resize 4x4 -> 7x7

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



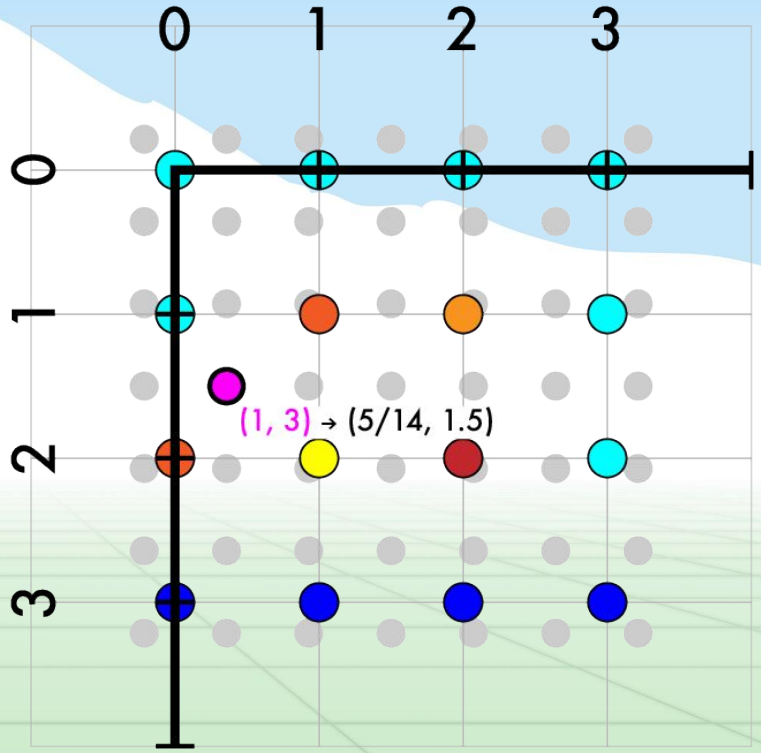
Resize 4x4 -> 7x7

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



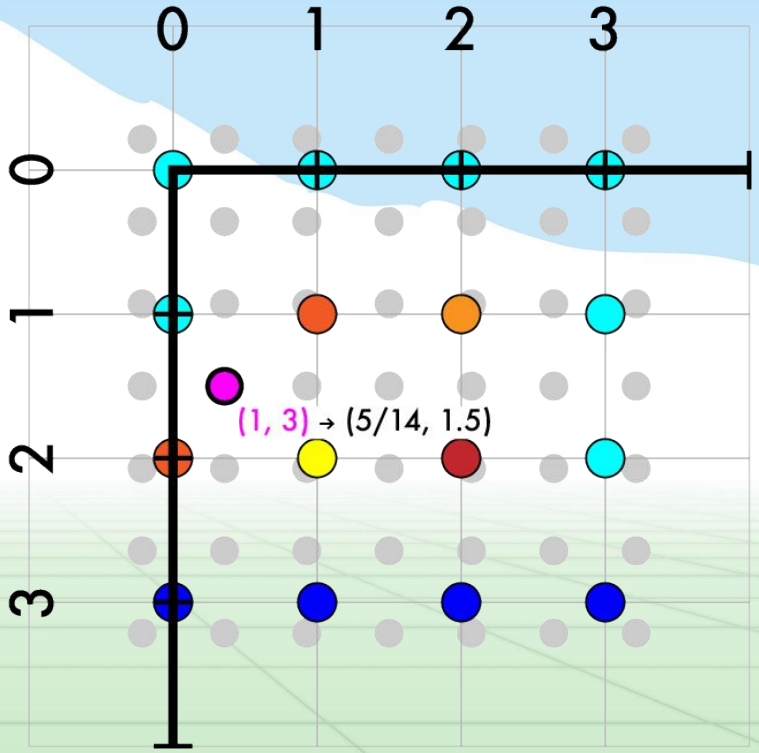
Resize 4x4 -> 7x7

- 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, 7/14)



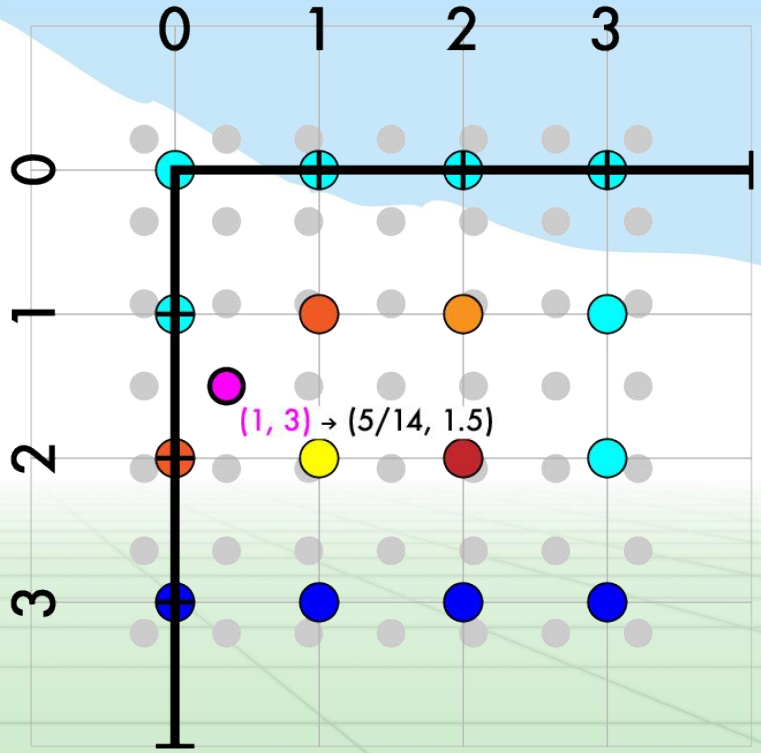
Resize 4x4 -> 7x7

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



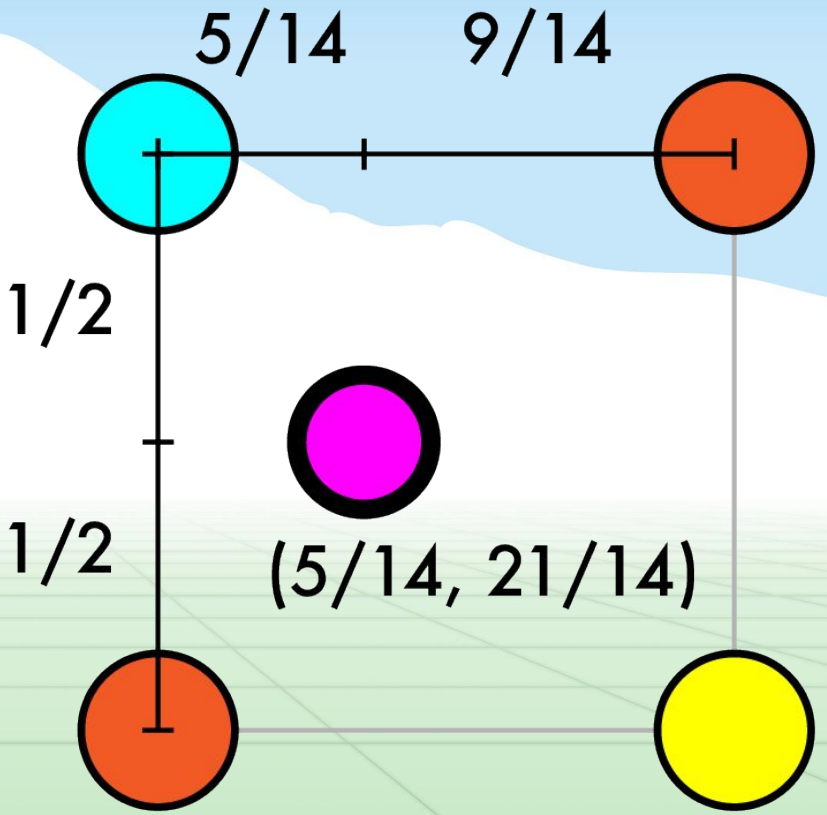
Resize 4x4 -> 7x7

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



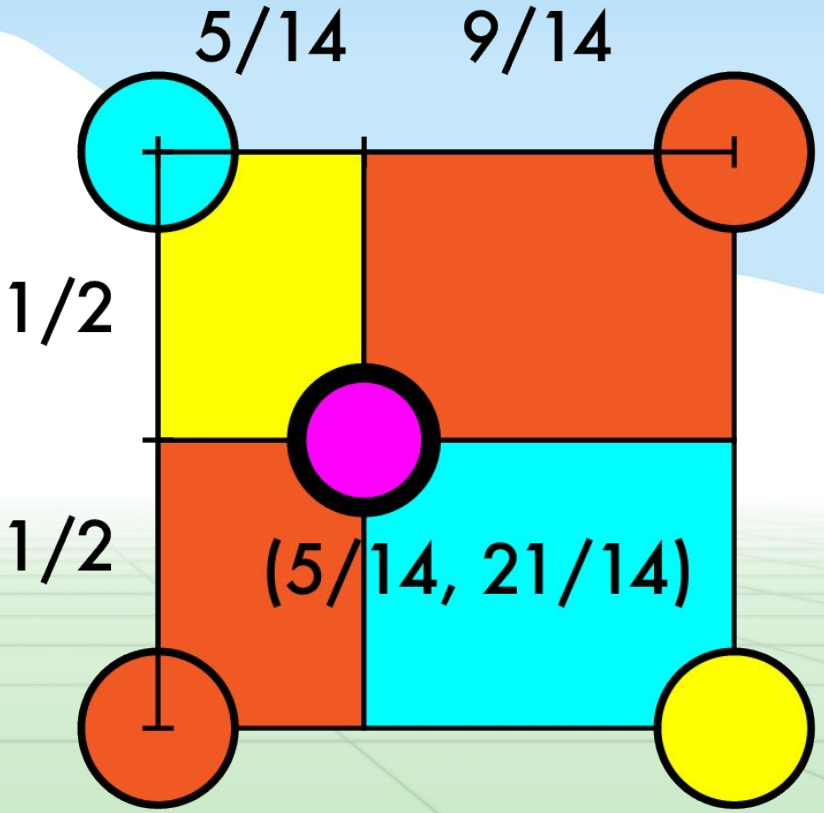
Resize 4x4 -> 7x7

- 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, 7/14)
  - Interpolate old values



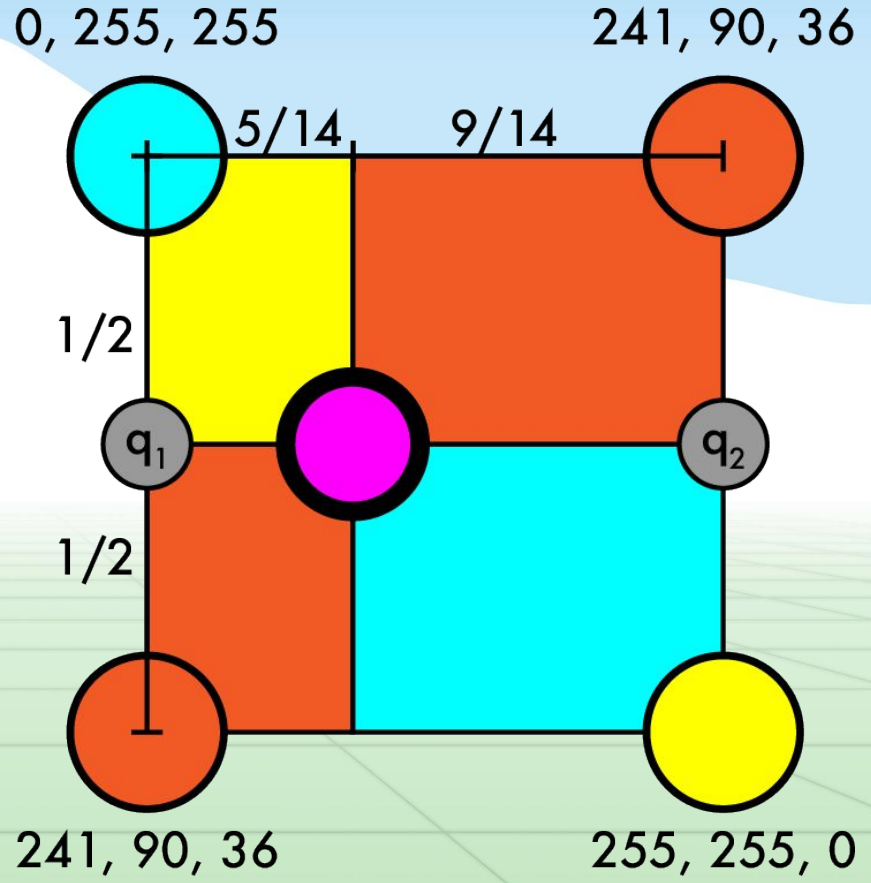
Resize 4x4 -> 7x7

- 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, 7/14)
  - Interpolate old values
    - Size of opposite rects



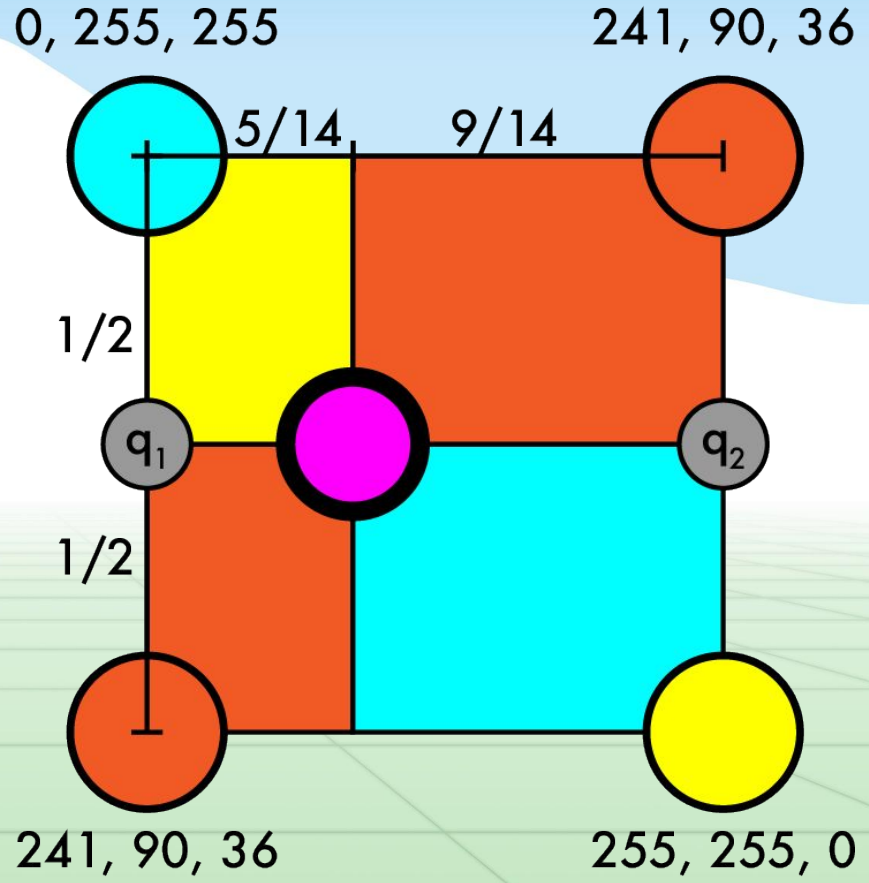
Resize 4x4 -> 7x7

- 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, 7/14)
  - Interpolate old values
    - Size of opposite rects
    - OR find q1 and q2, then interpolate between them



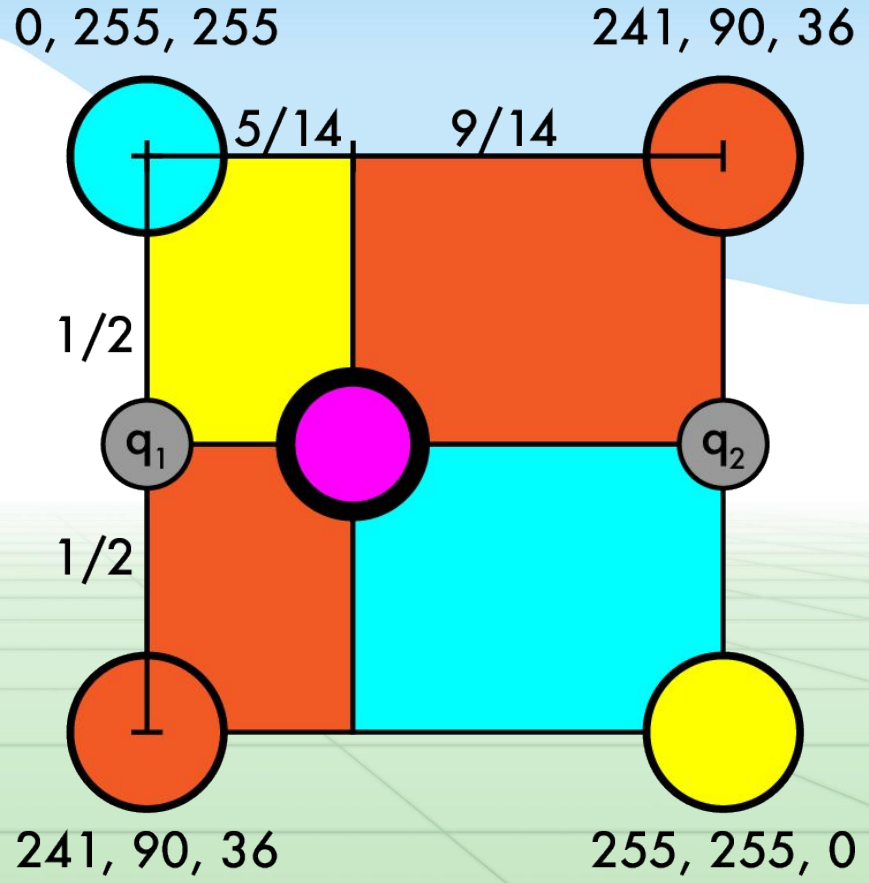
Resize 4x4 -> 7x7

- 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, 7/14)
  - Interpolate old values
    - q1 = r1, g1, b1
    - r1 = .5*0 + .5*241
    - g1 = .5*255 + .5*90
    - b1 = .5*241 + .5*36



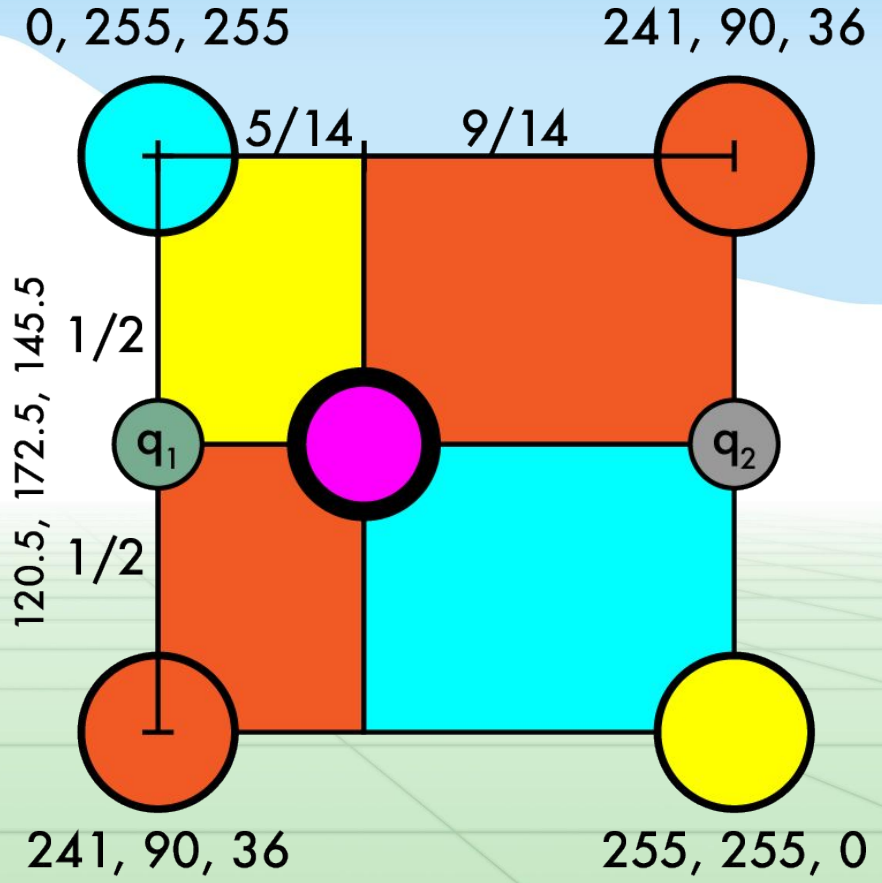
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 7/14)
  - Interpolate old values
    - q1 = r1, g1, b1
    - r1 = .5*0 + .5*241
    - g1 = .5*255 + .5*90
    - b1 = .5*241 + .5*36



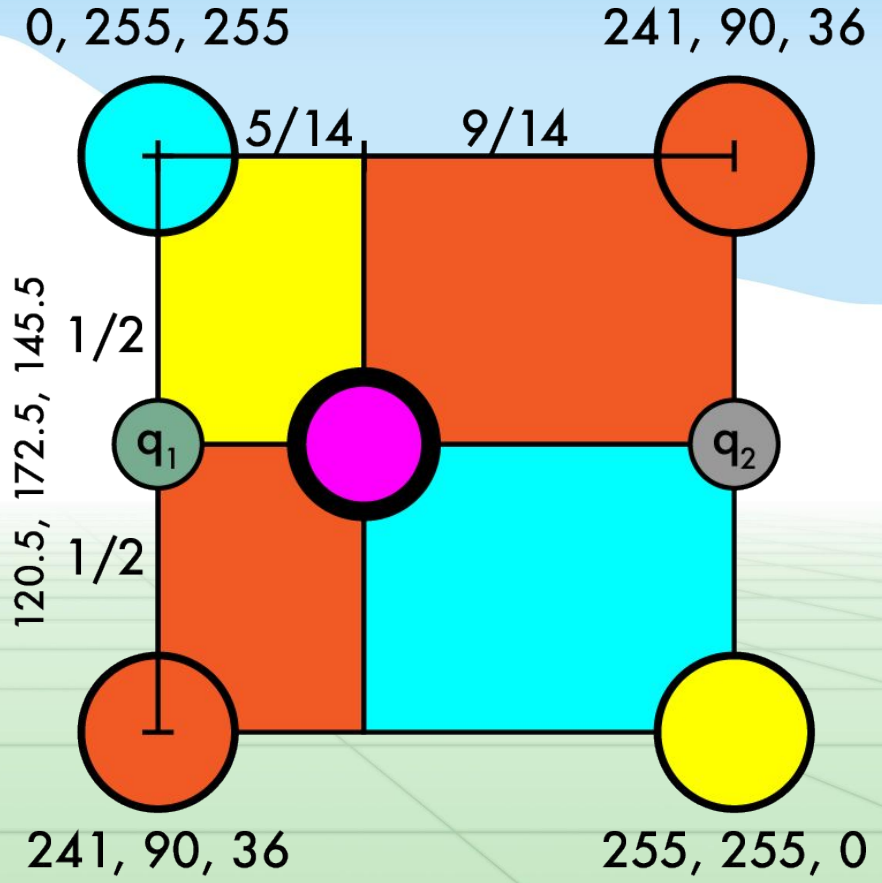
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 7/14)
  - Interpolate old values
    - q1 = (120.5, 172.5, 145.5)
    - q2 = r2, g2, b2



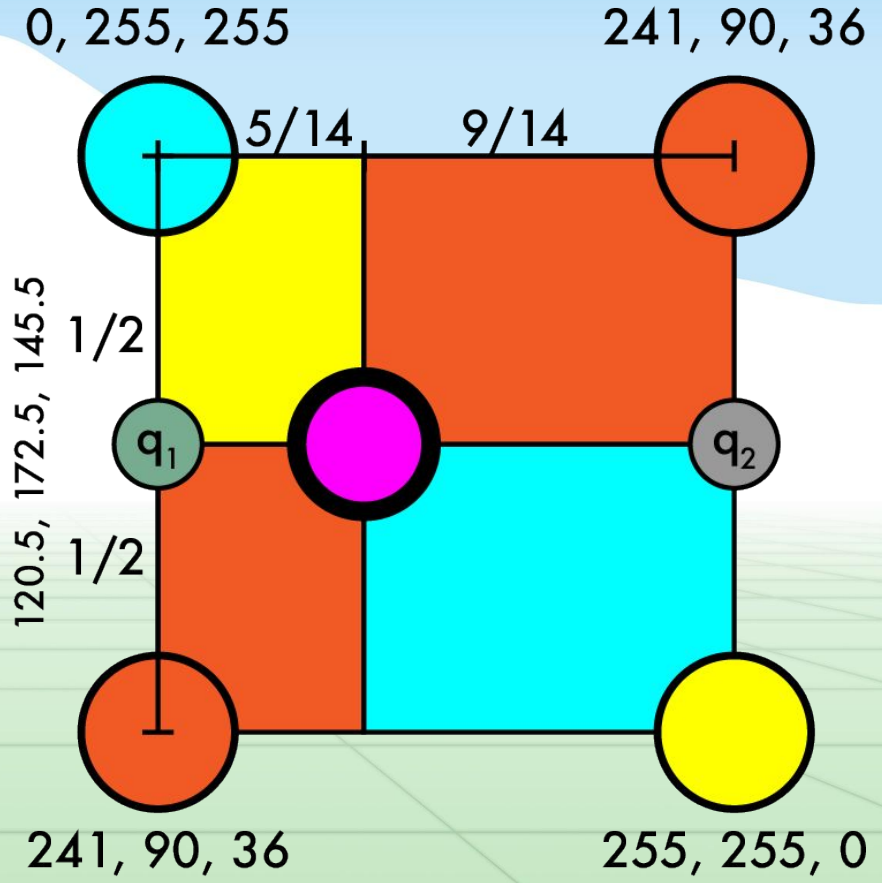
Resize 4x4 -> 7x7

- 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, 7/14)
  - Interpolate old values
    - q1 = (120.5, 172.5, 145.5)
    - q2 = r2, g2, b2



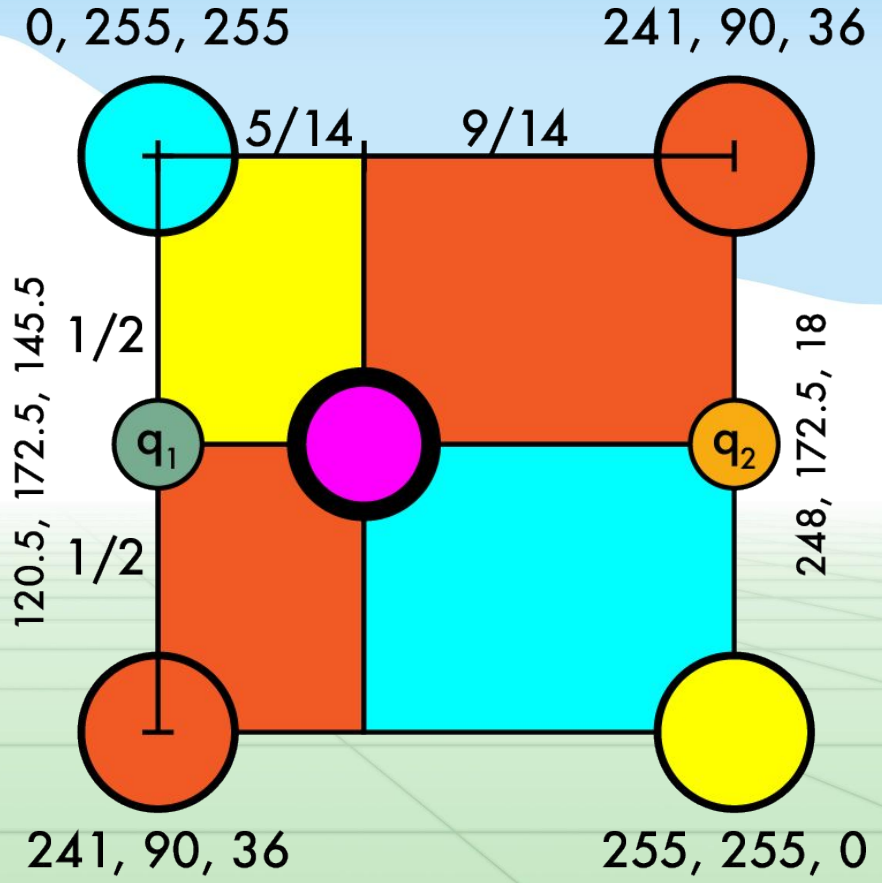
Resize 4x4 -> 7x7

- 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, 7/14)
  - Interpolate old values
    - q1 = (120.5, 172.5, 145.5)
    - q2 = r2, g2, b2
    - r2 = .5*241 + .5*255
    - g2 = .5*90 + .5*255
    - b2 = .5*36 + .5*0



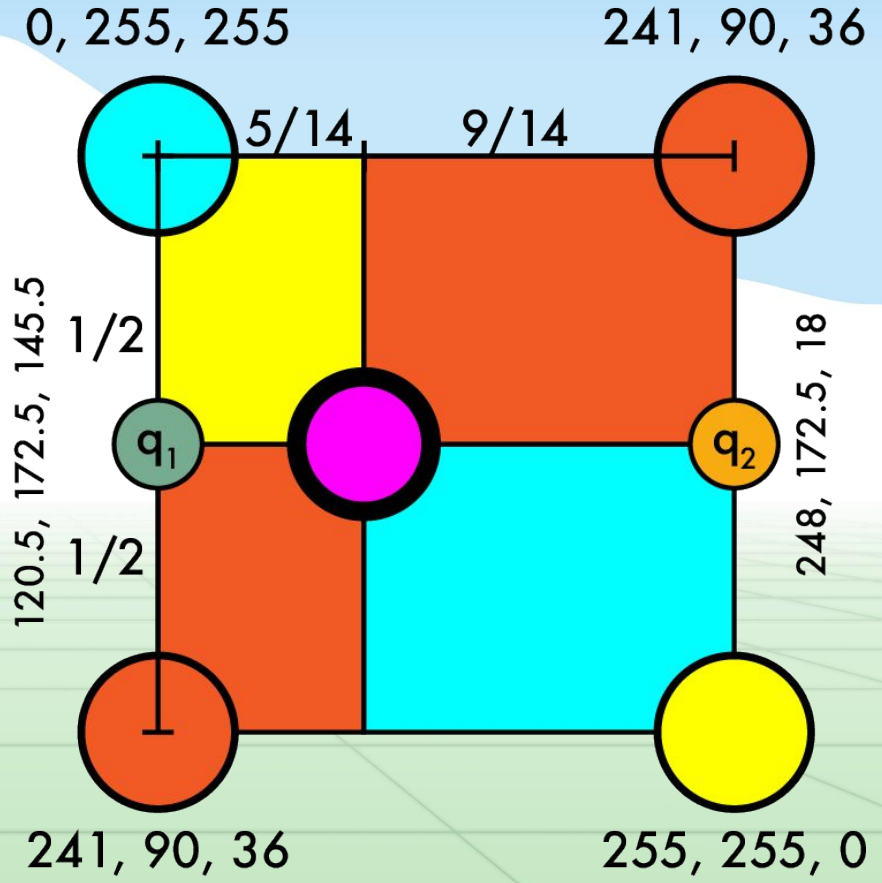
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 7/14)
  - Interpolate old values
    - q1 = (120.5, 172.5, 145.5)
    - q2 = (248, 172.5, 18)



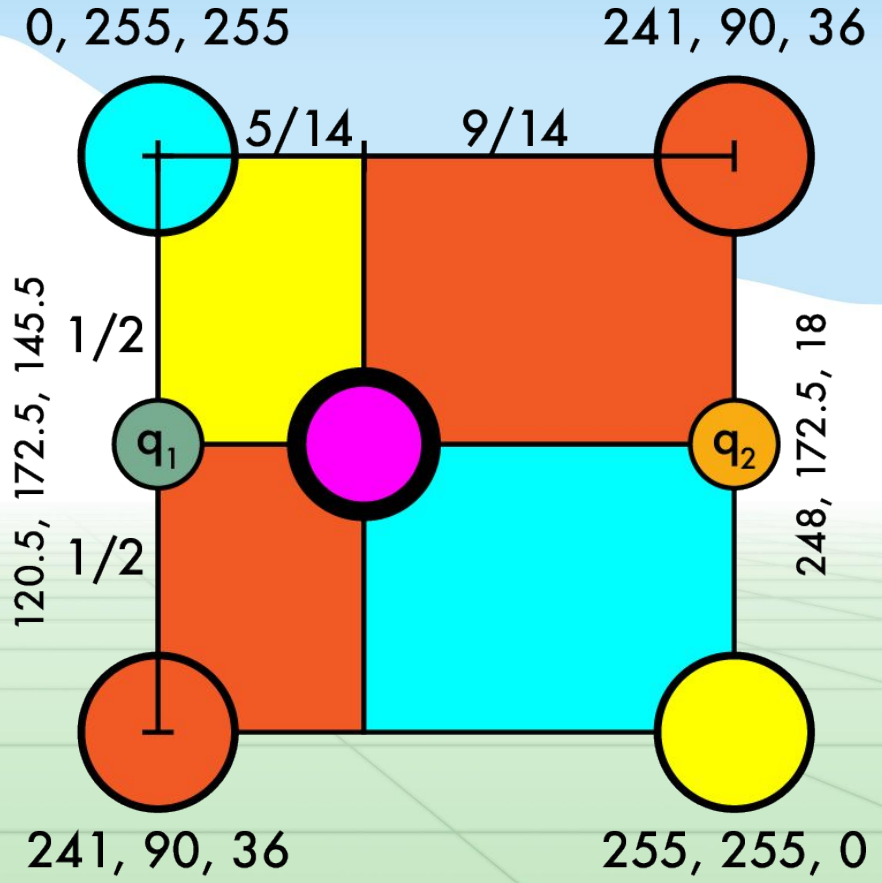
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 7/14)
  - Interpolate old values
    - q1 = (120.5, 172.5, 145.5)
    - q2 = (248, 172.5, 18)
    - q = r, g, b
    - w = 9/14*w1 + 5/14 w2



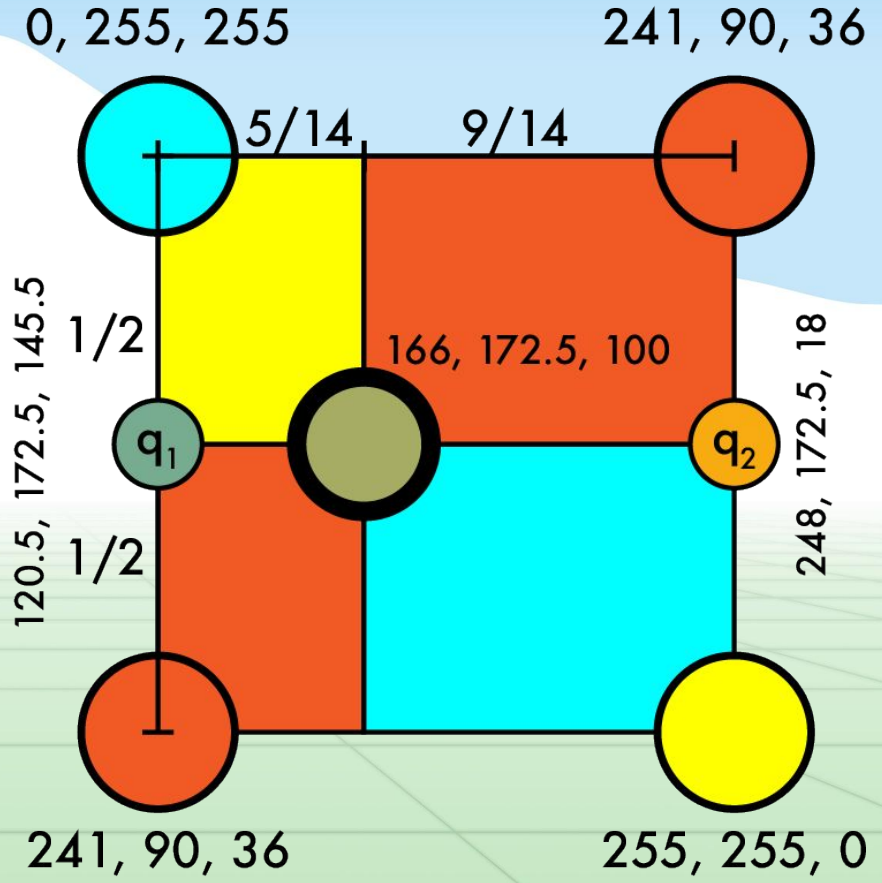
Resize 4x4 -> 7x7

- 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, 7/14)
  - Interpolate old values
    - q1 = (120.5, 172.5, 145.5)
    - q2 = (248, 172.5, 18)
    - q = r, g, b
    - w = 9/14*w1 + 5/14 w2
    - 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



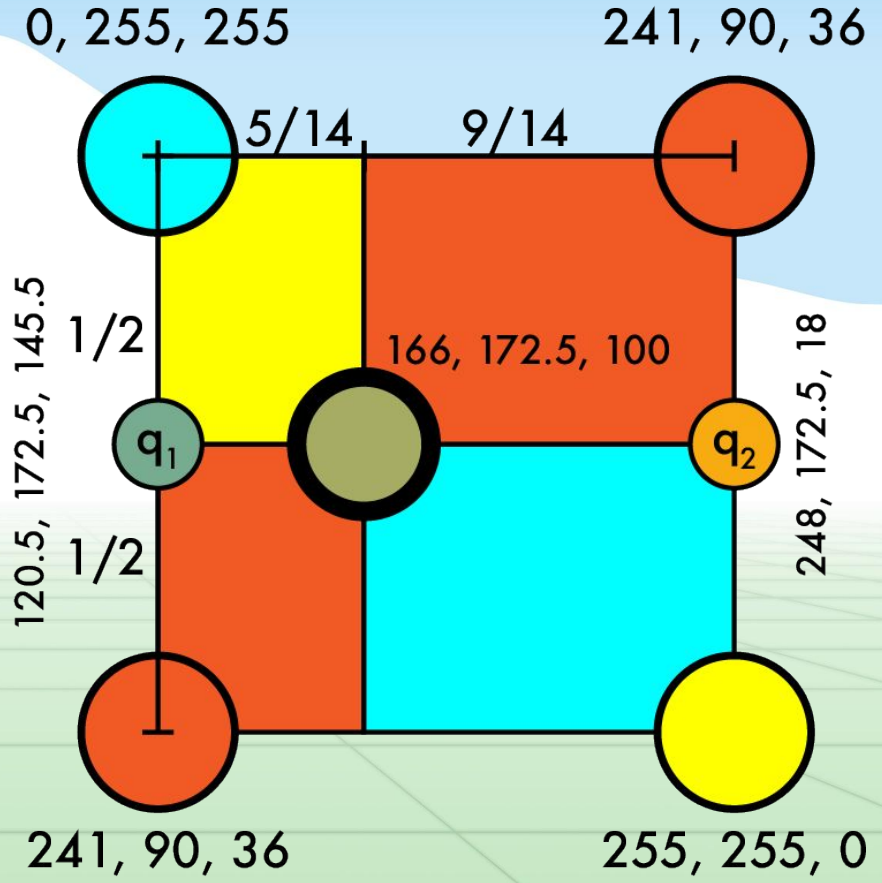
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 7/14)
  - Interpolate old values
    - q1 = (120.5, 172.5, 145.5)
    - q2 = (248, 172.5, 18)
    - q = (166, 172.5, 100)



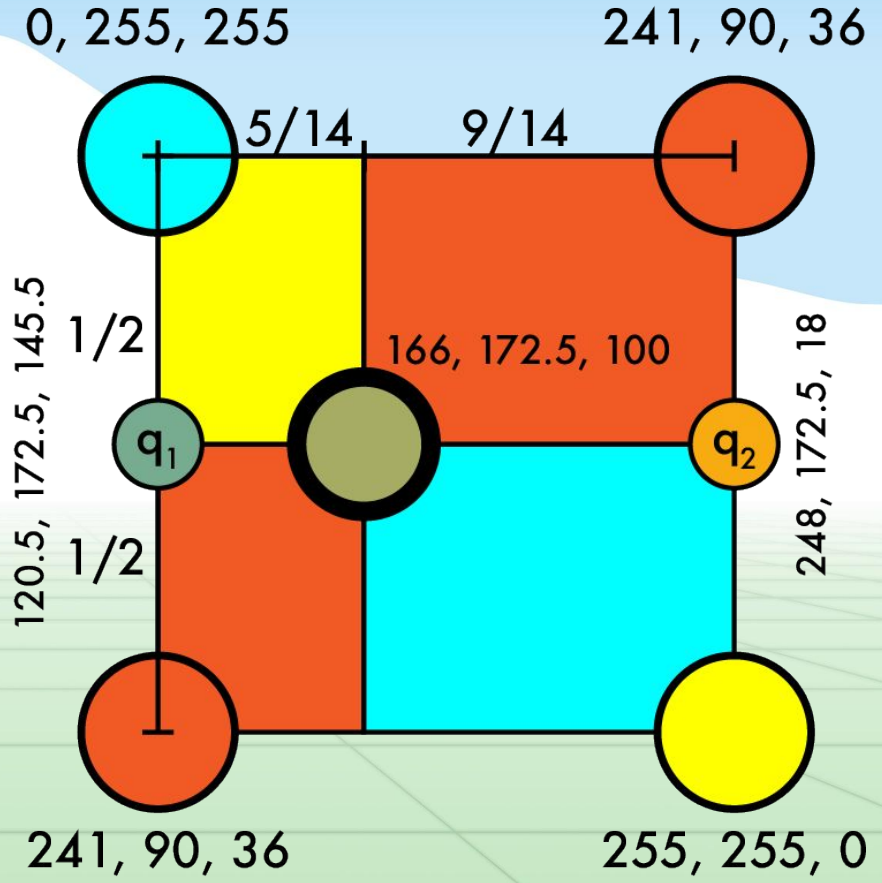
Resize 4x4 -> 7x7

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



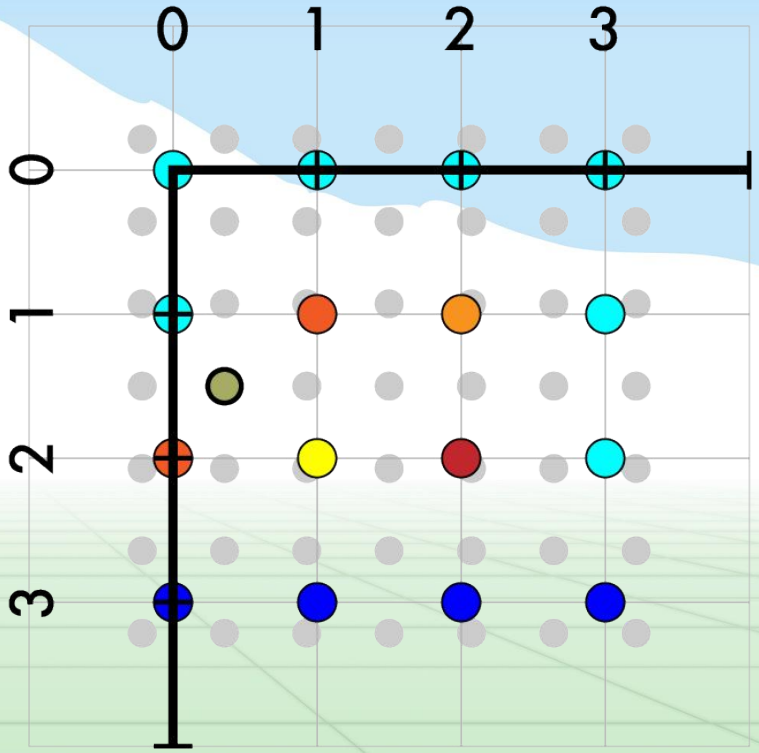
Resize 4x4 -> 7x7

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



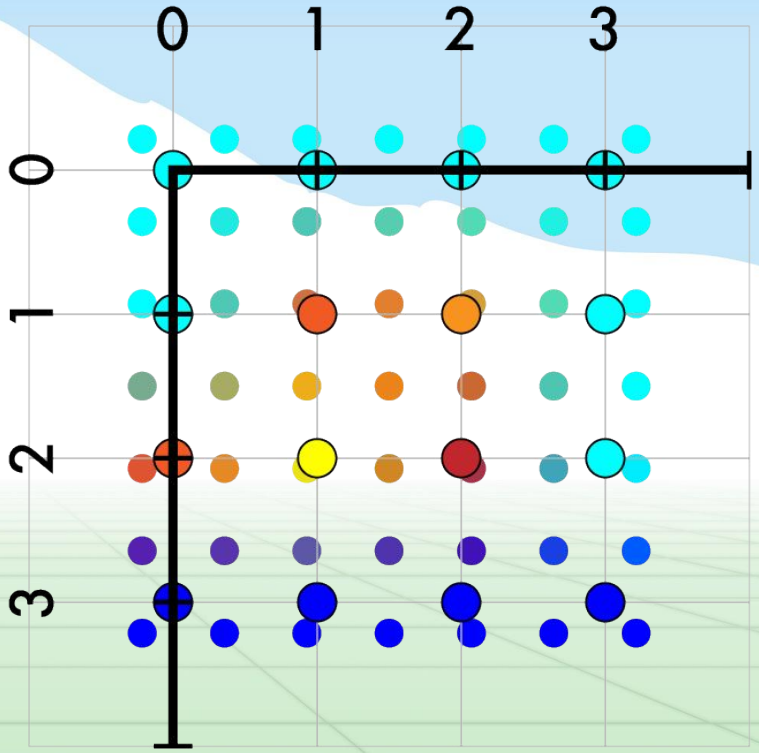
Resize 4x4 -> 7x7

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



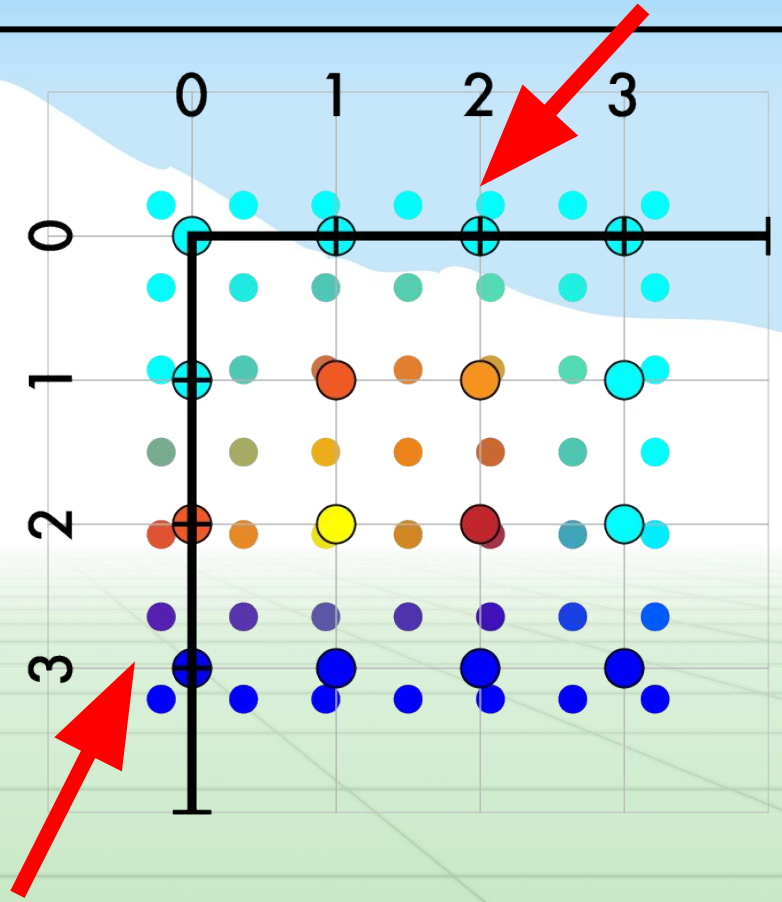
Resize 4x4 -> 7x7

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



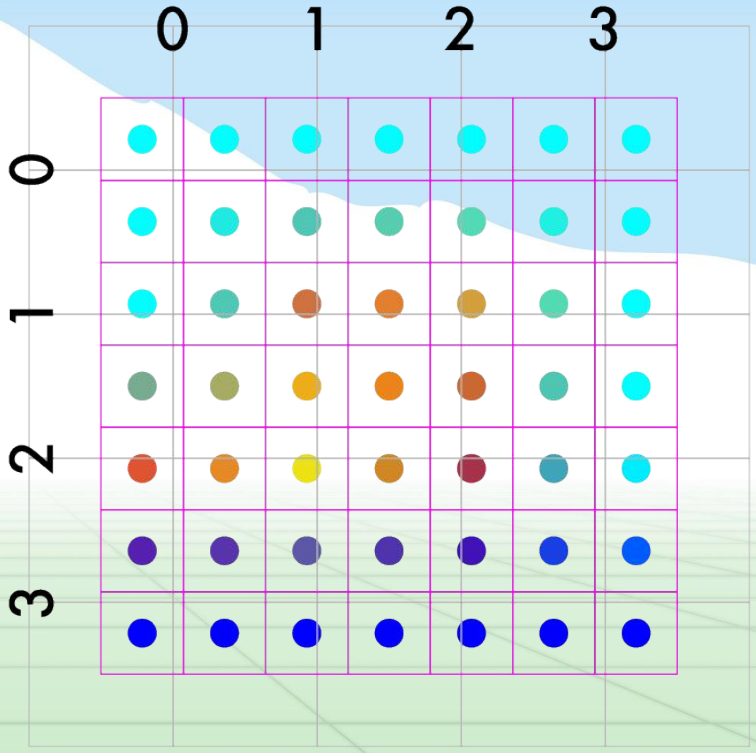
Resize 4x4 -> 7x7

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



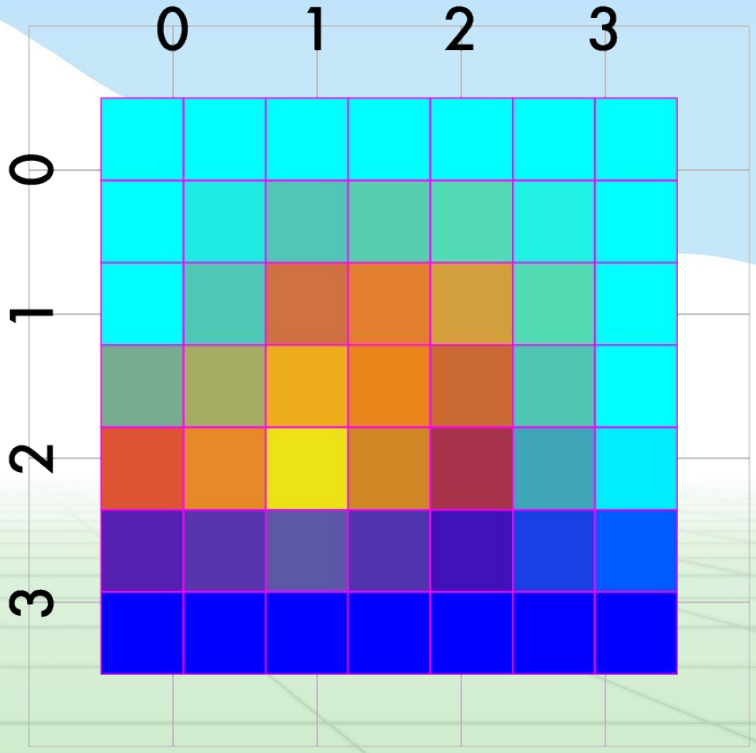
Resize 4x4 -> 7x7

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

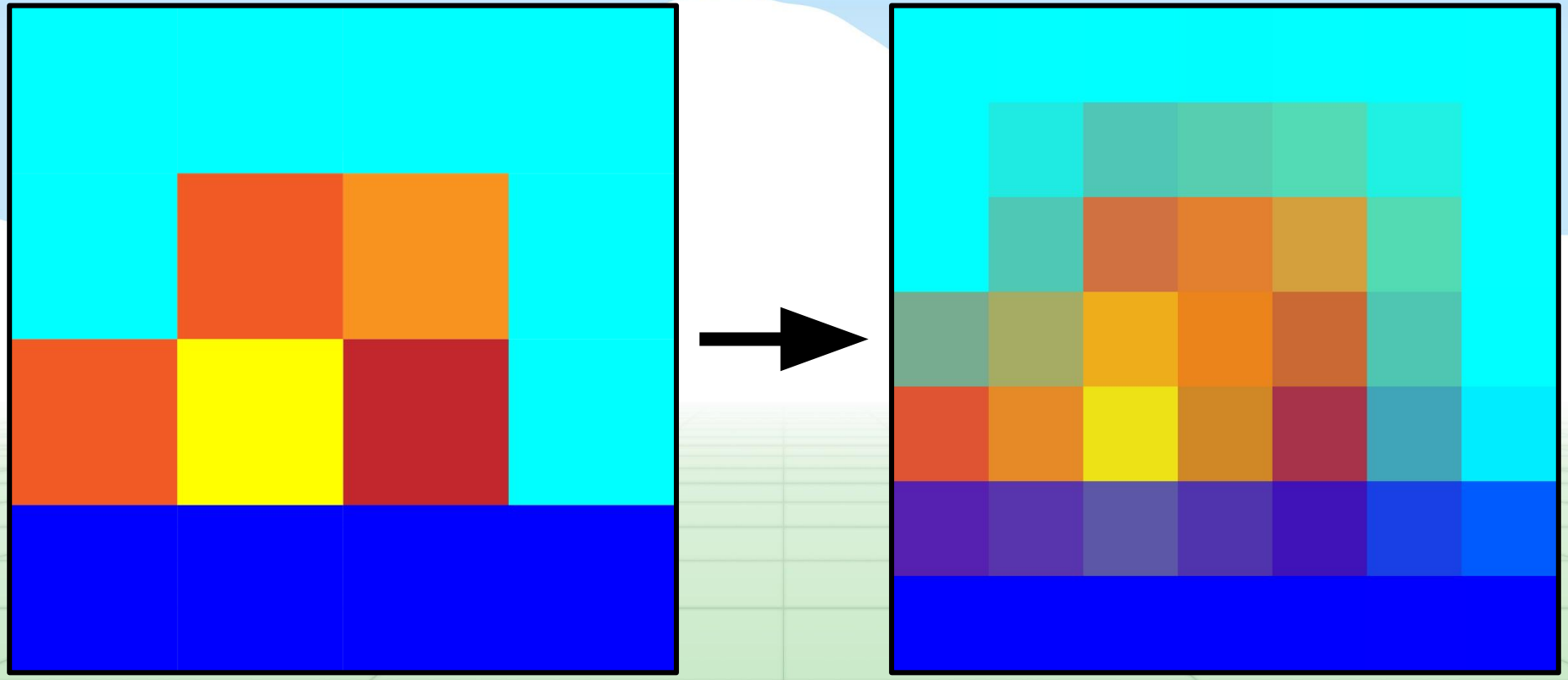


Resize 4x4 -> 7x7

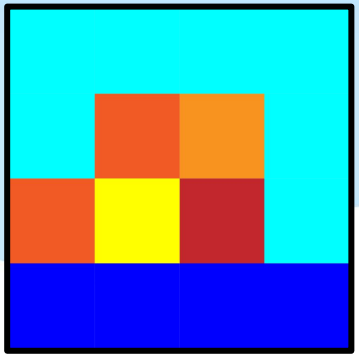
- Create our new image
- Match up coordinates
  - 4/7 X - 3/14 = Y
- Iterate over new pts
  - Map to old coords
  - (1, 3) -> (5/14, 7/14)
  - Interpolate old values
    - q = (166, 172.5, 100)
- Fill in the rest

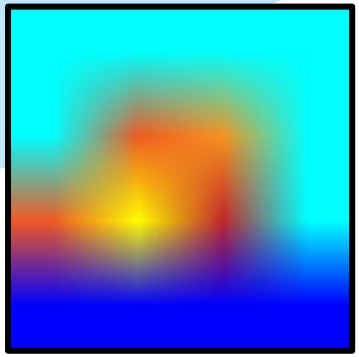


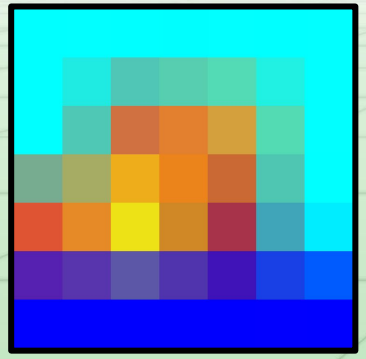
We did it!

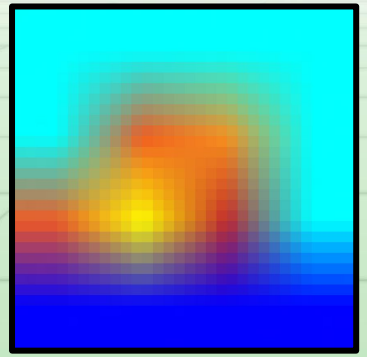


Different scales

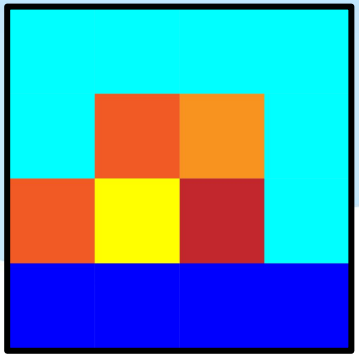


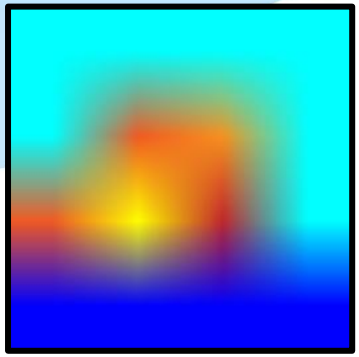


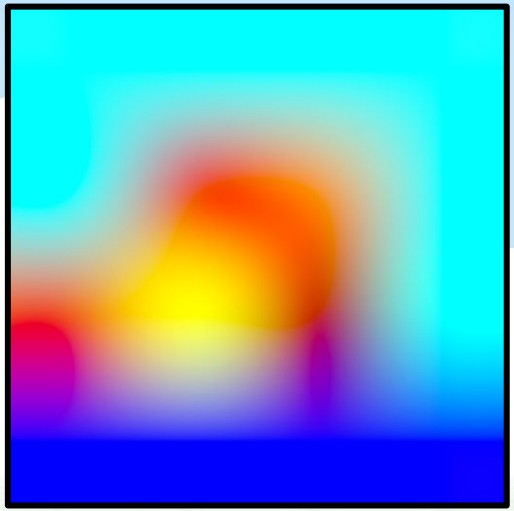


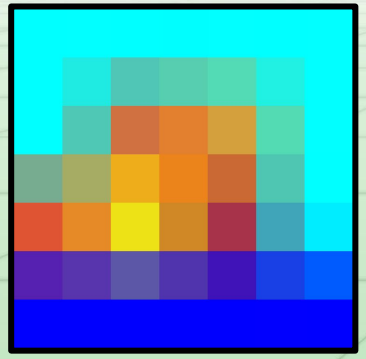


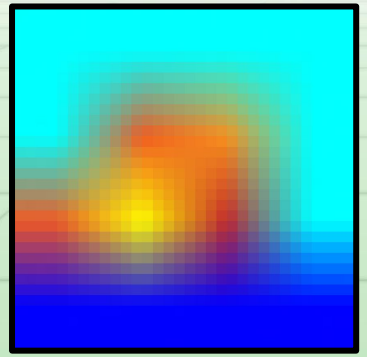
Different methods

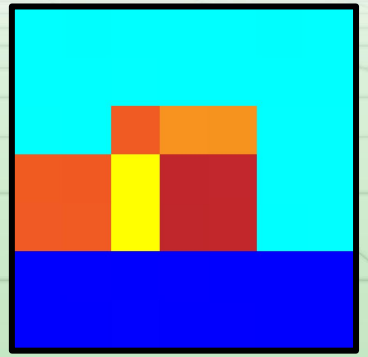












Let’s do something interesting already!!

Want to make image smaller

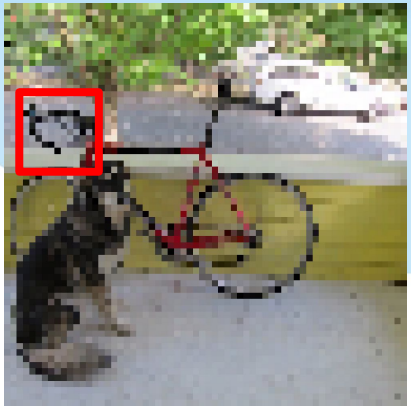
448x448 -> 64x64

448x448 -> 64x64

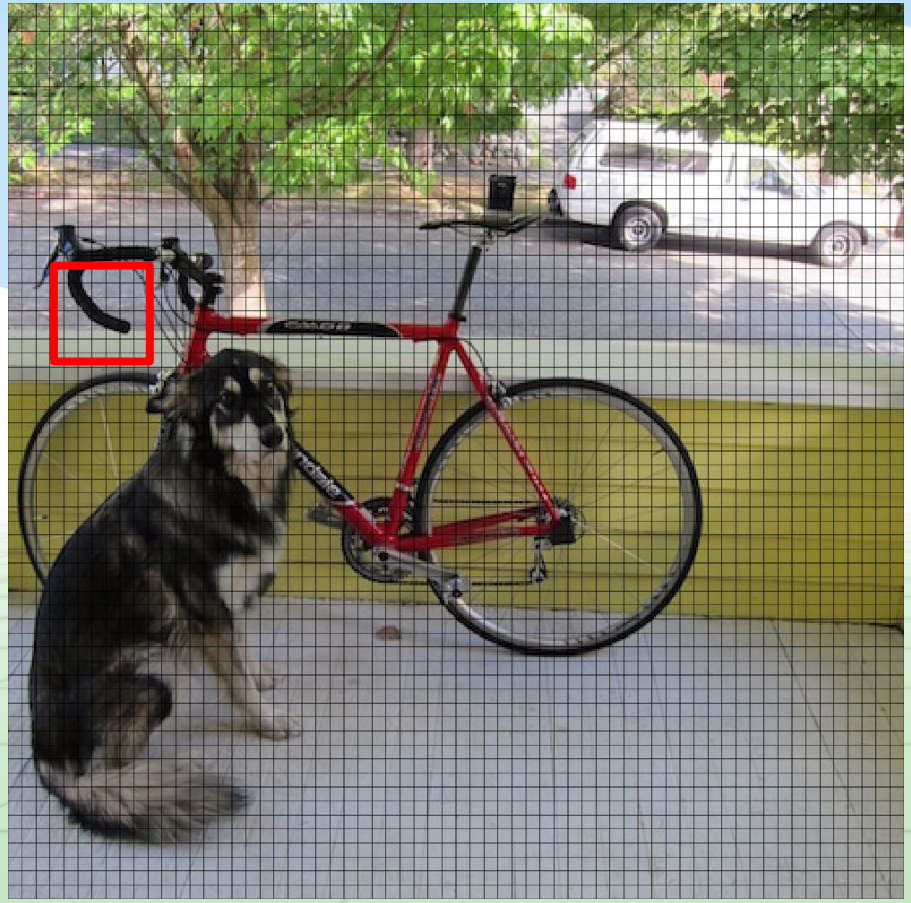
448x448 -> 64x64

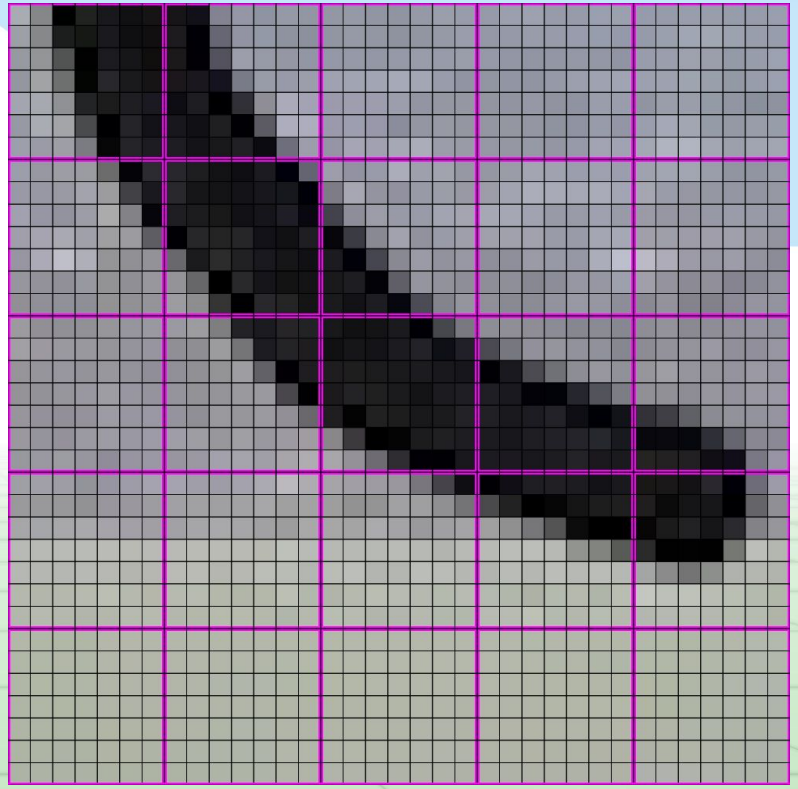
448x448 -> 64x64

448x448 -> 64x64

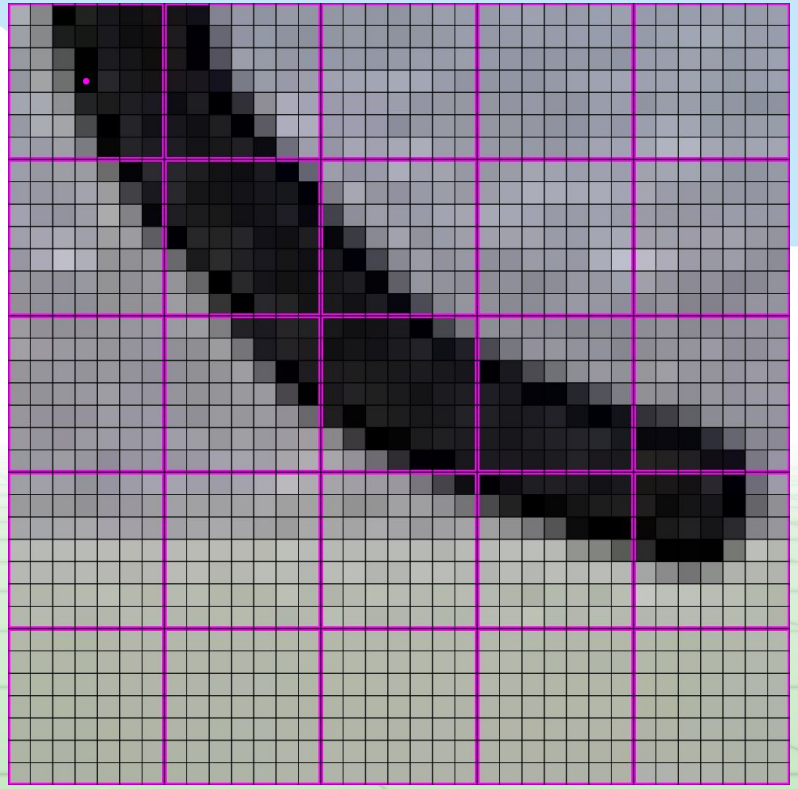


448x448 -> 64x64

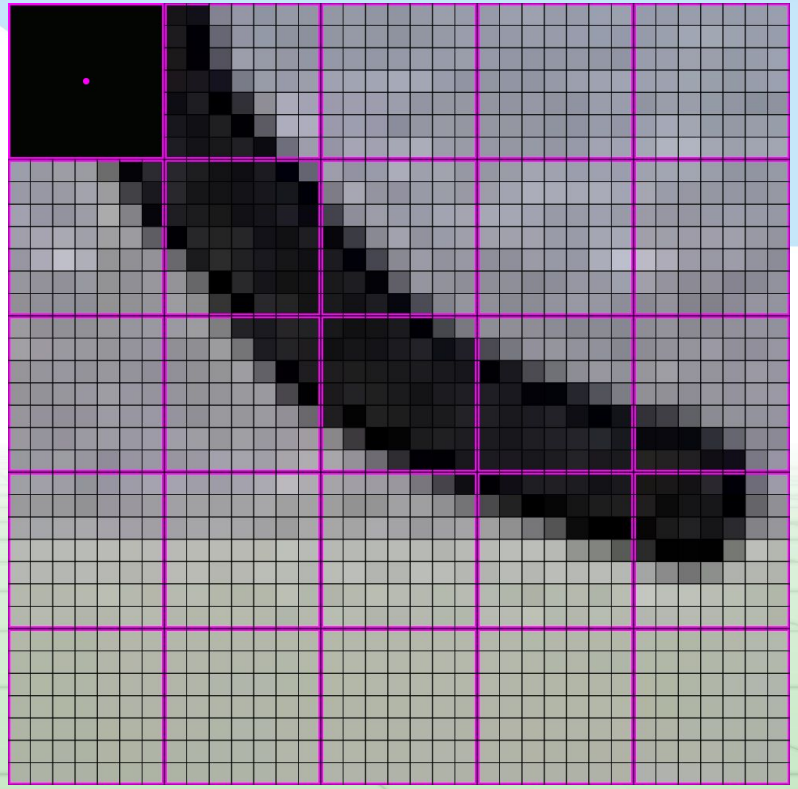




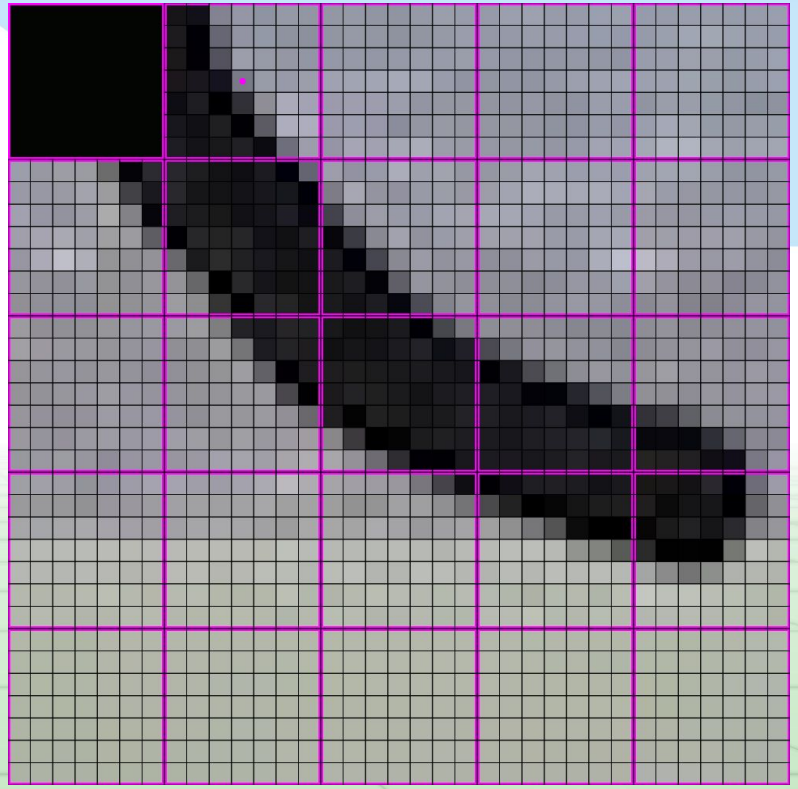
448x448 -> 64x64



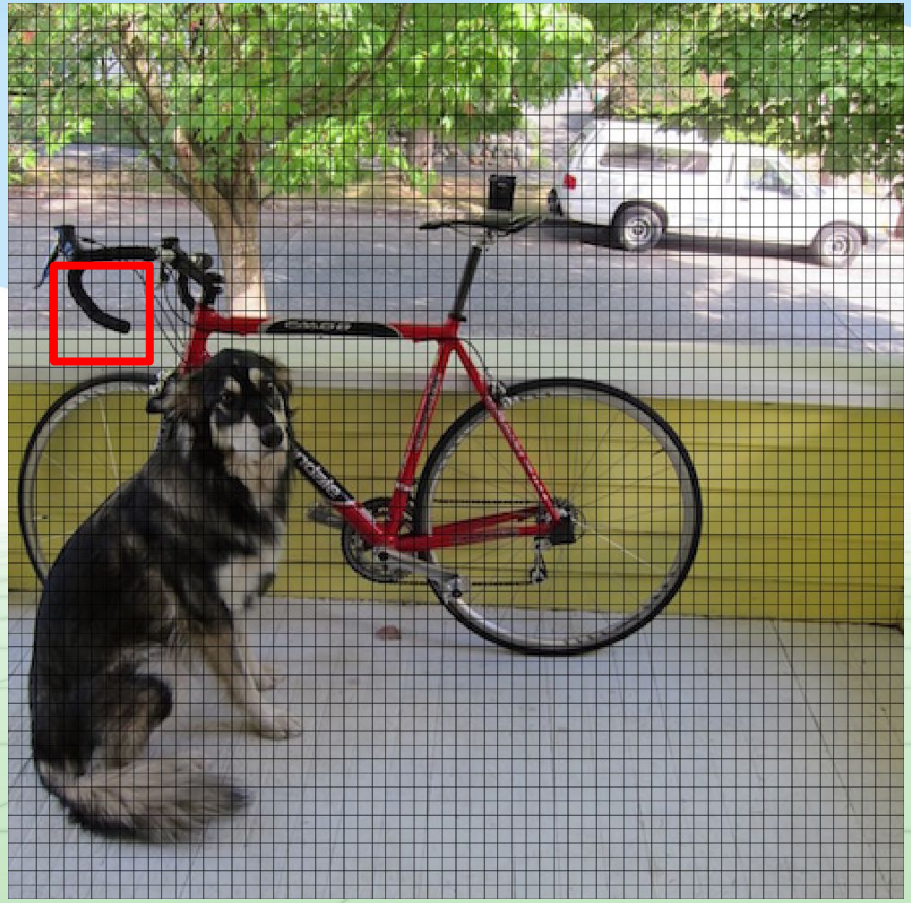
448x448 -> 64x64

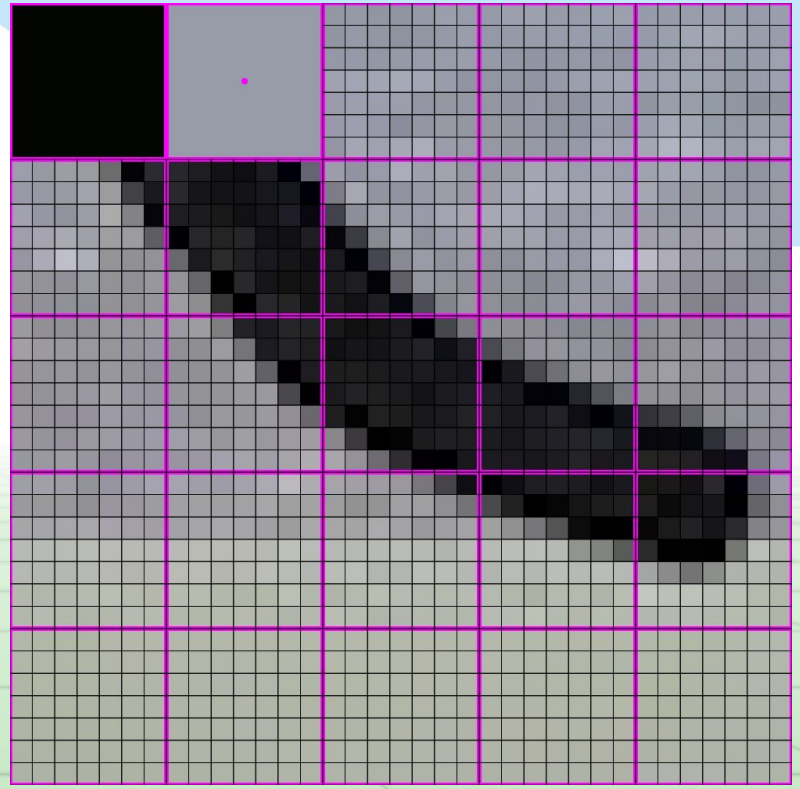


448x448 -> 64x64

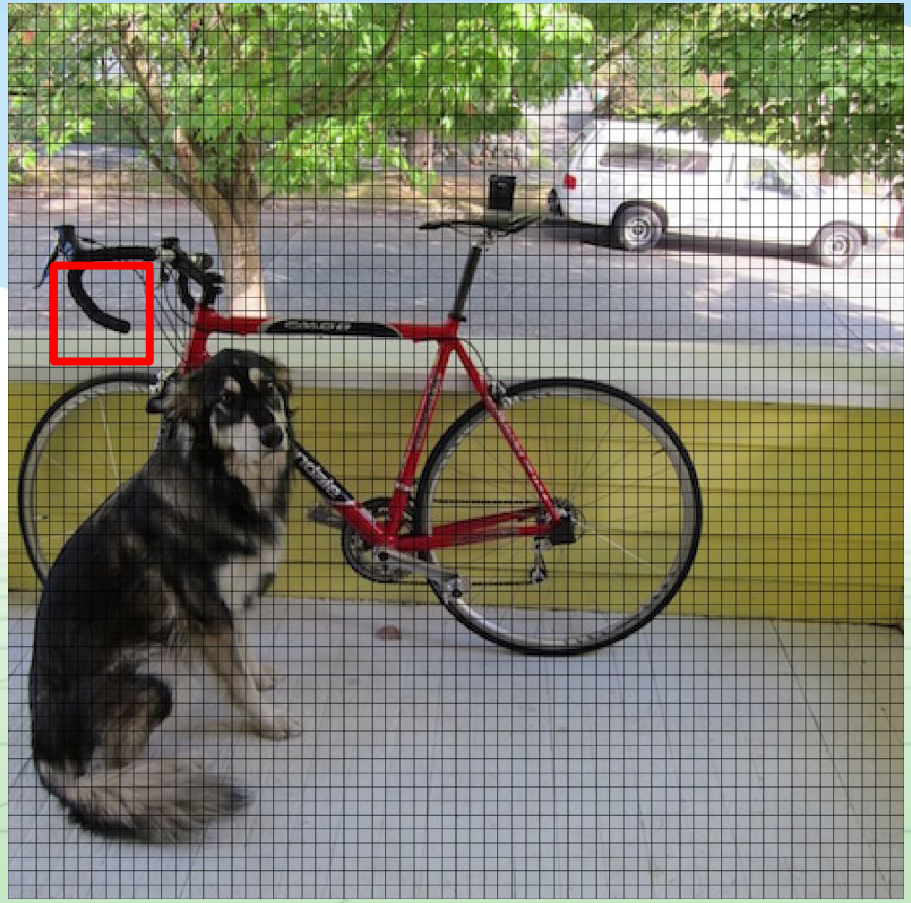


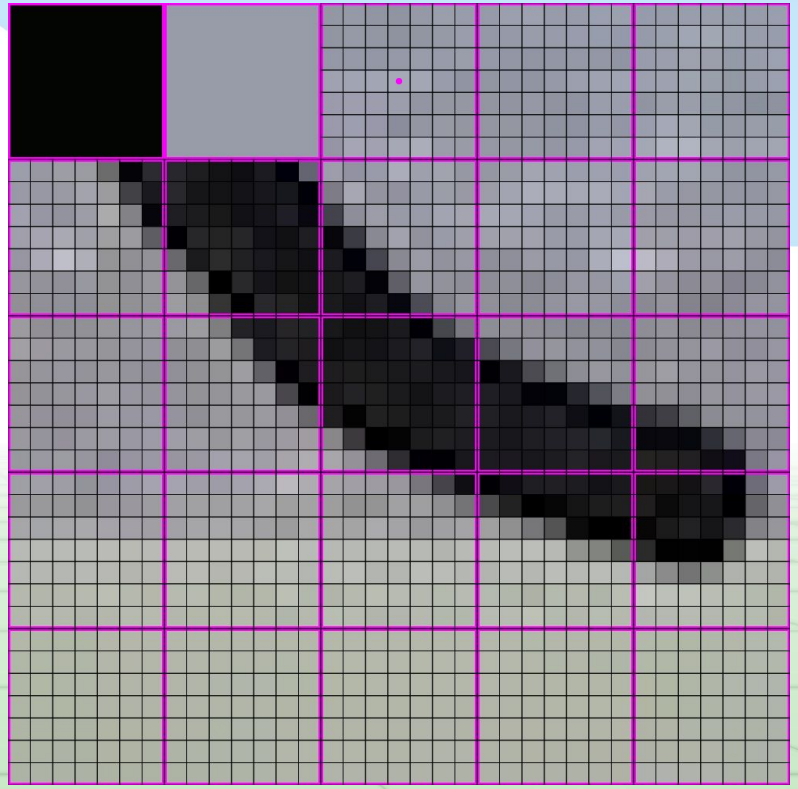
448x448 -> 64x64



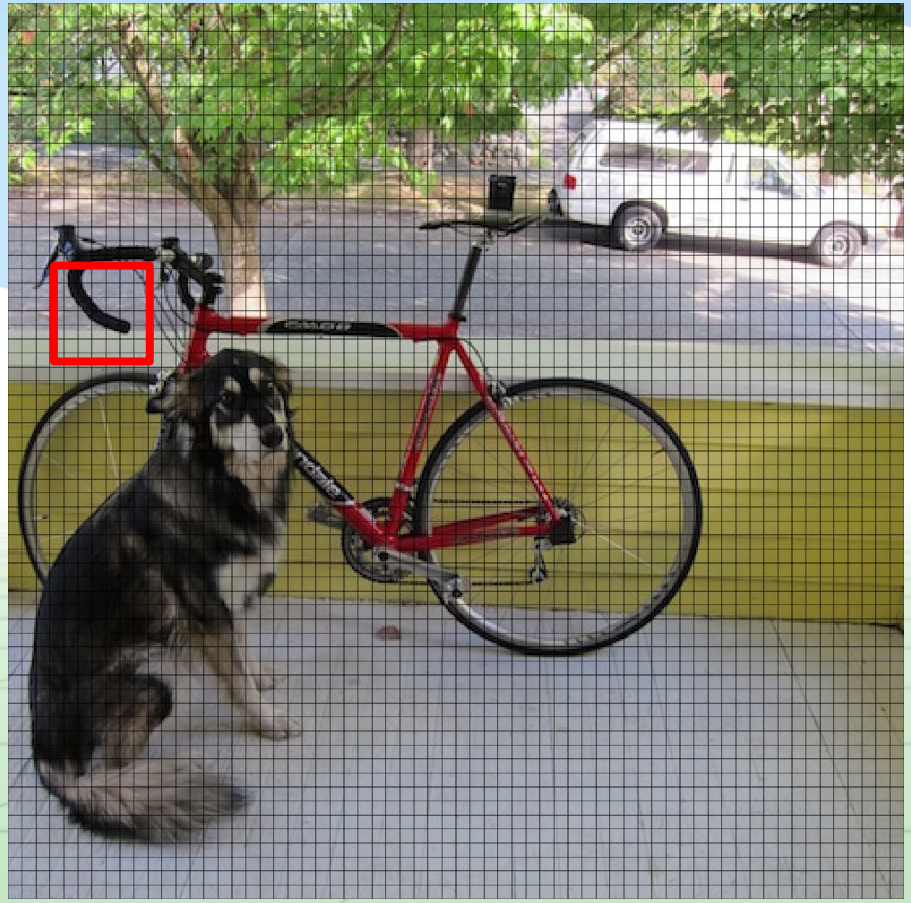


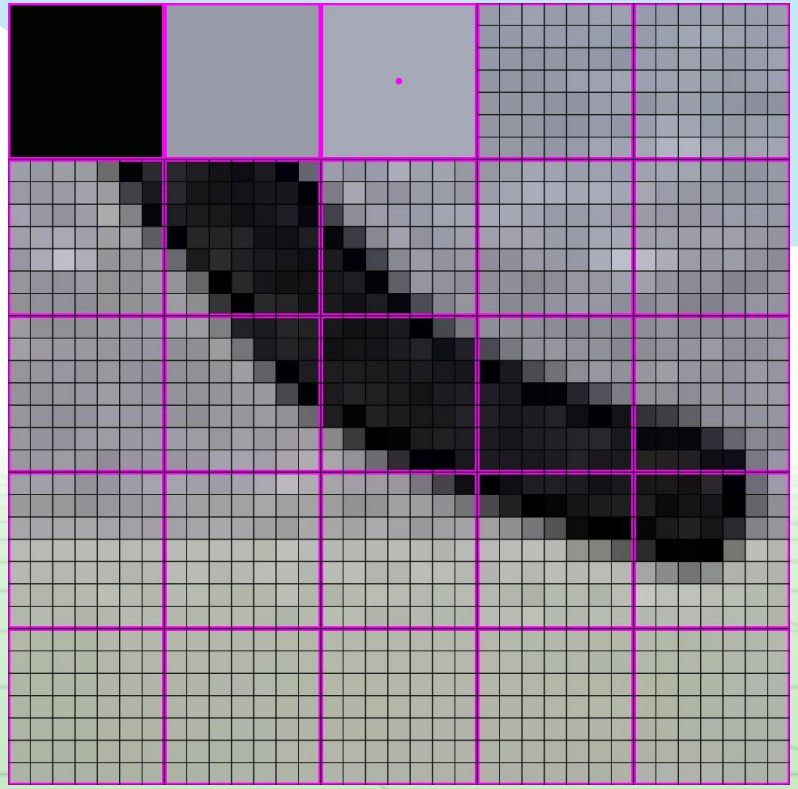
448x448 -> 64x64



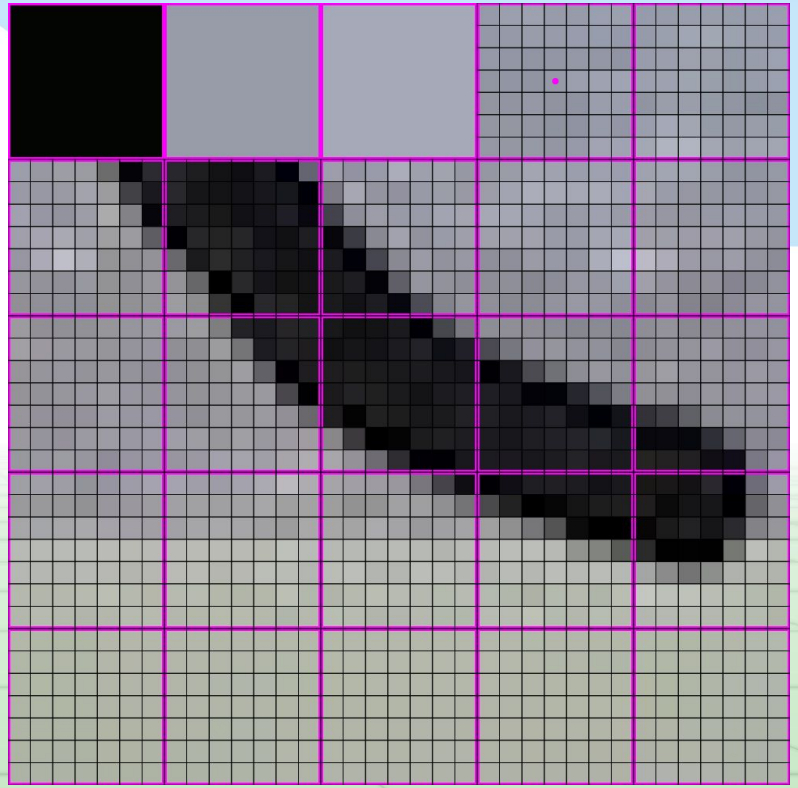


448x448 -> 64x64

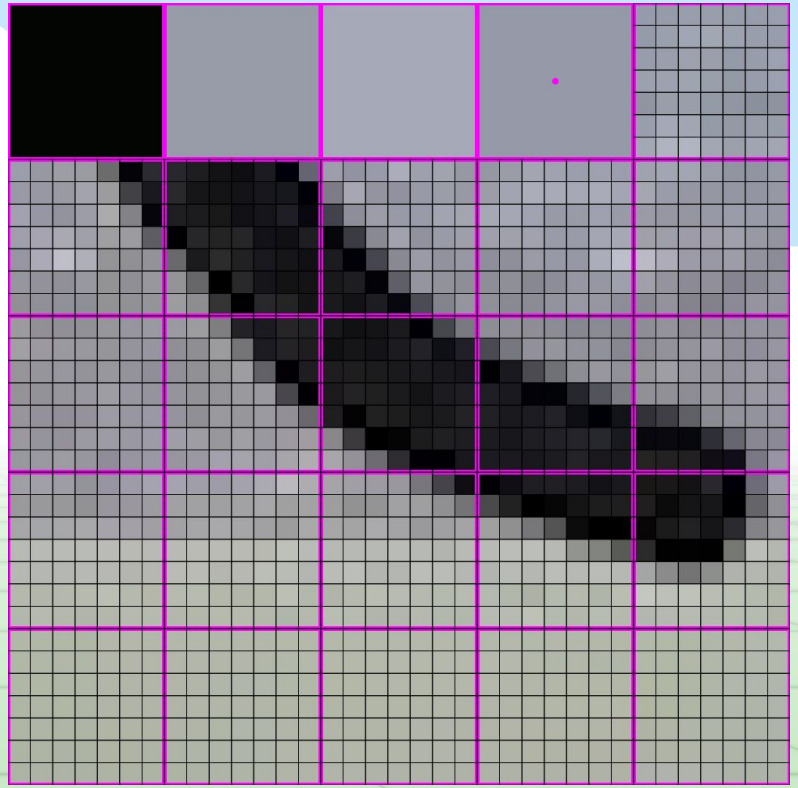




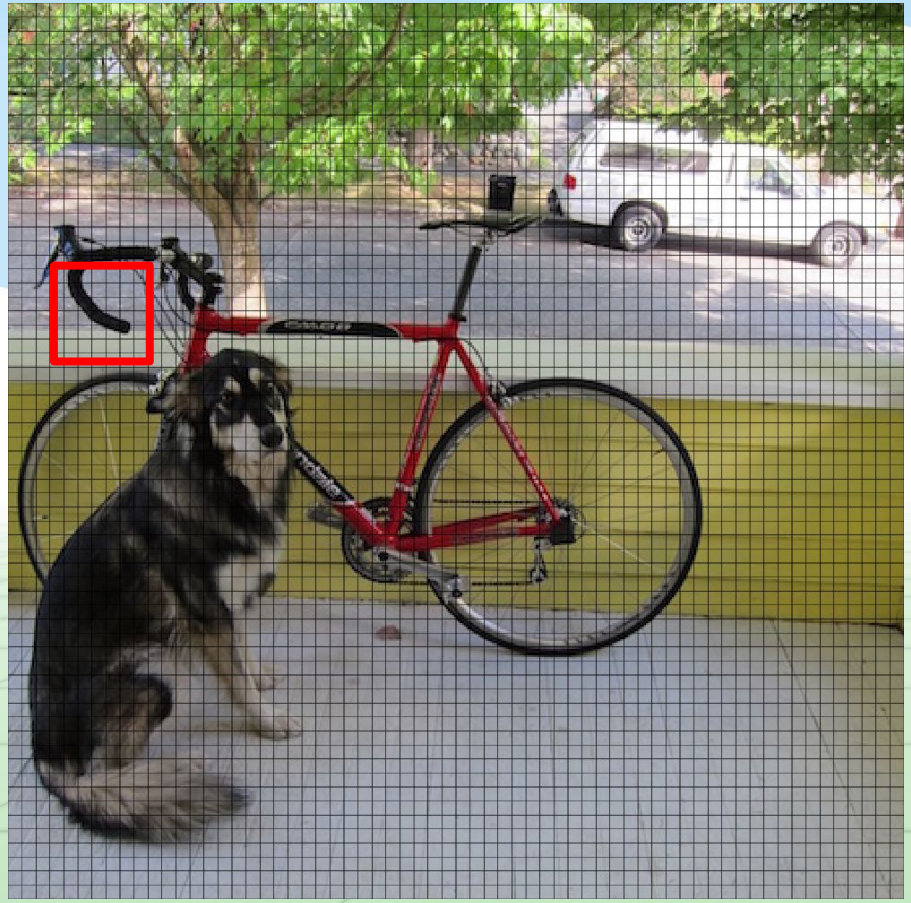
448x448 -> 64x64

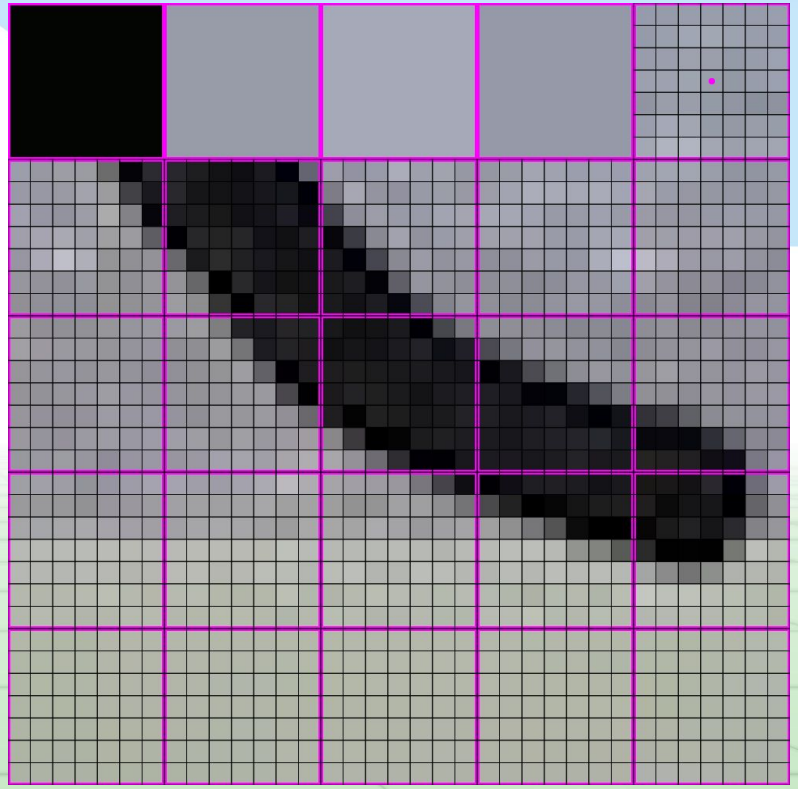


448x448 -> 64x64

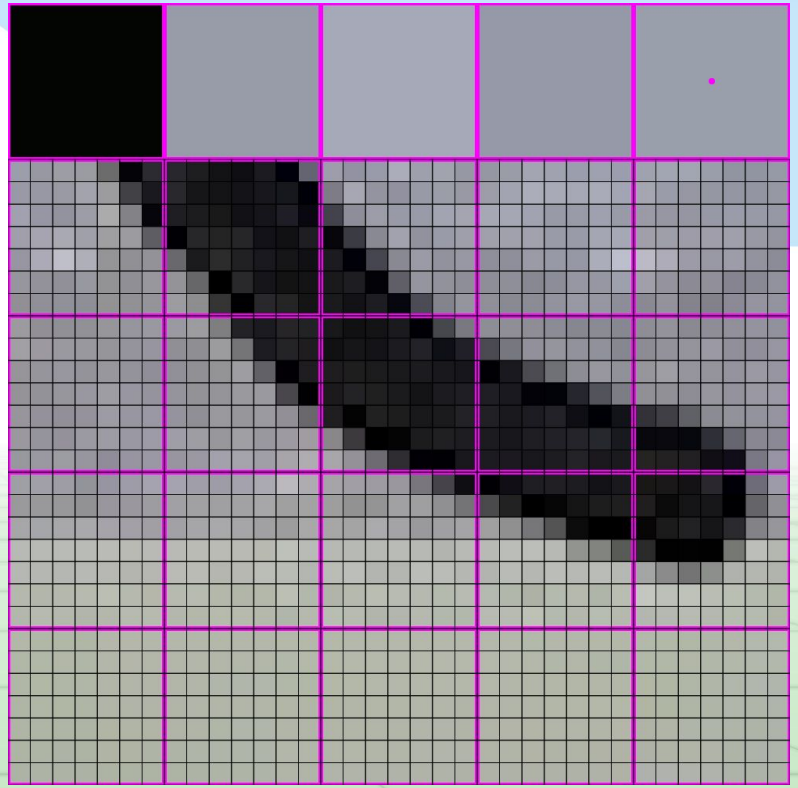


448x448 -> 64x64

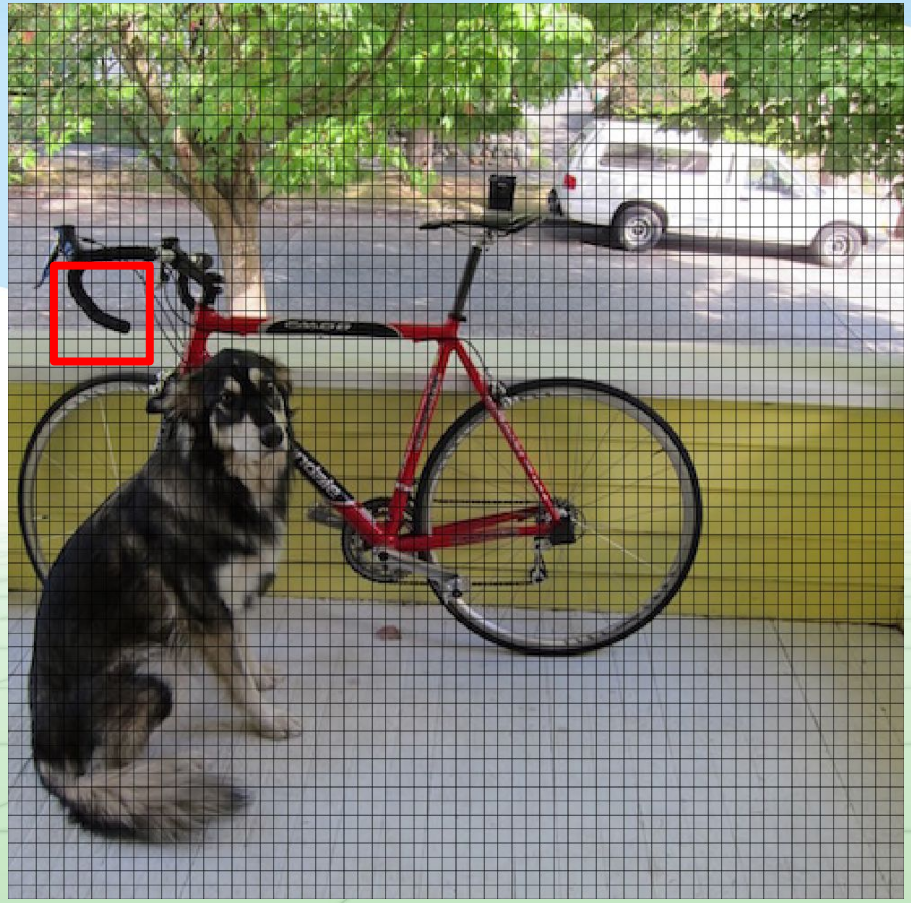


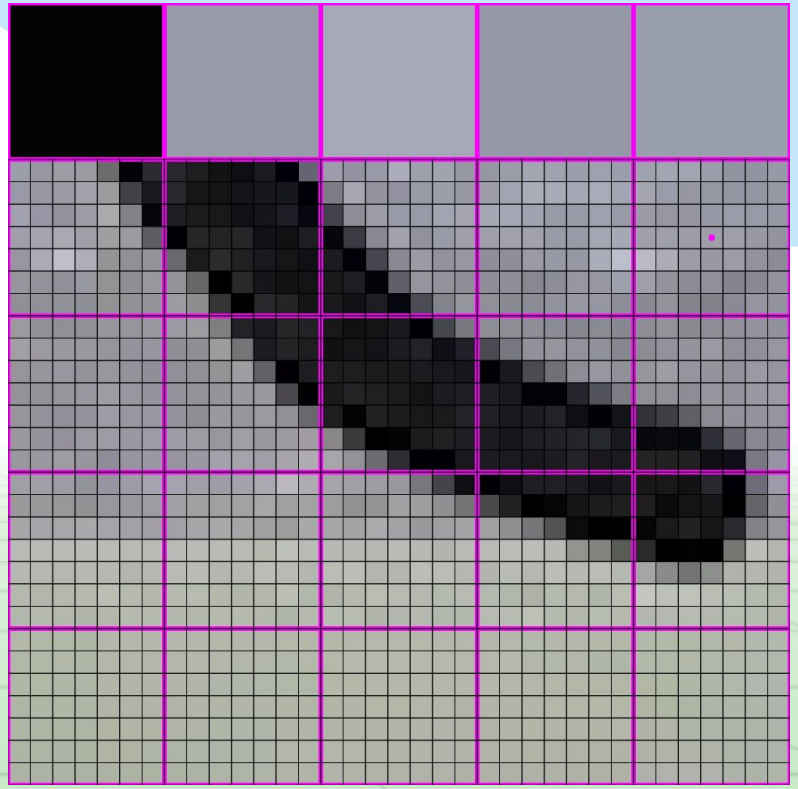


448x448 -> 64x64

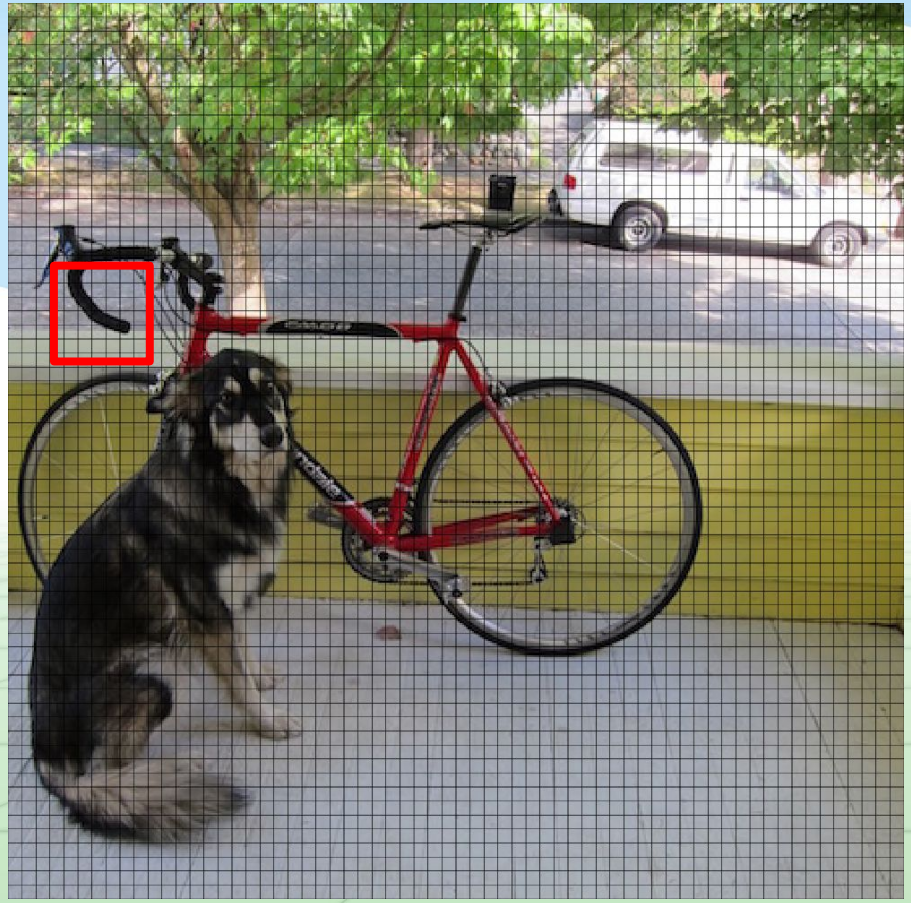


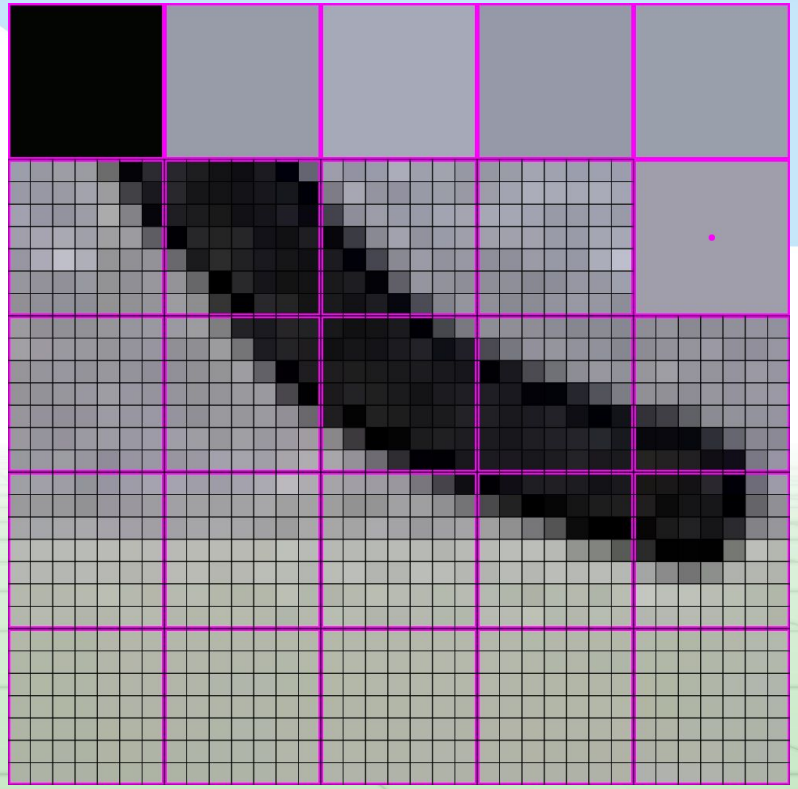
448x448 -> 64x64



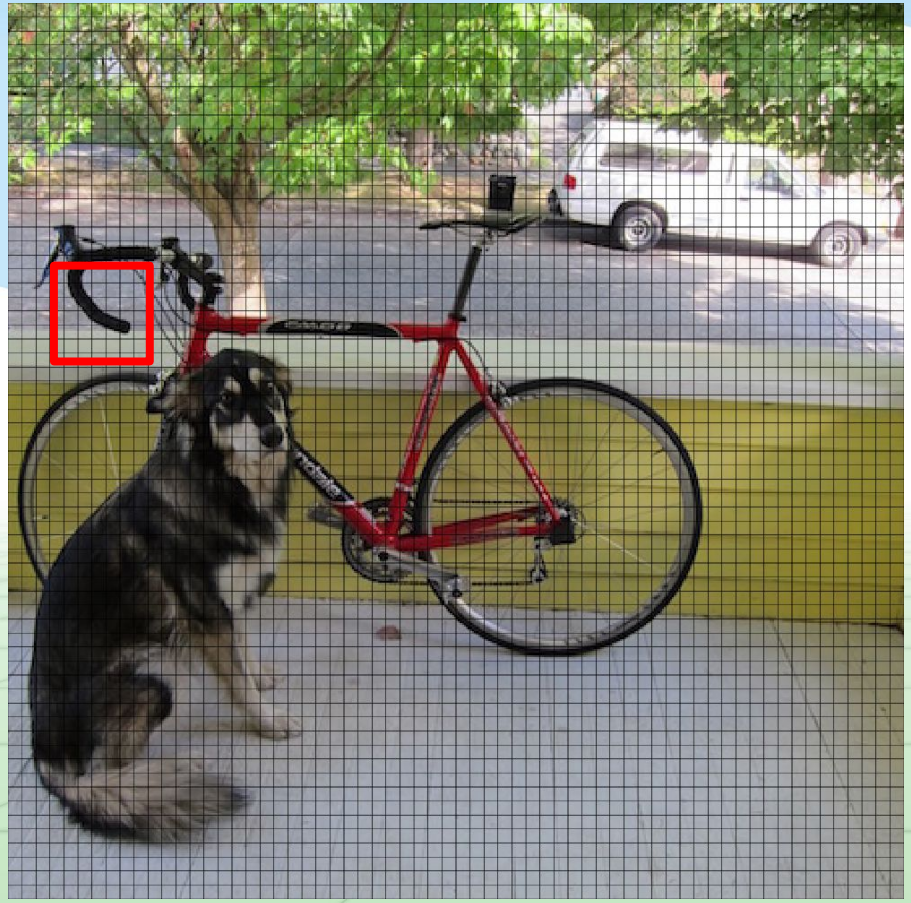


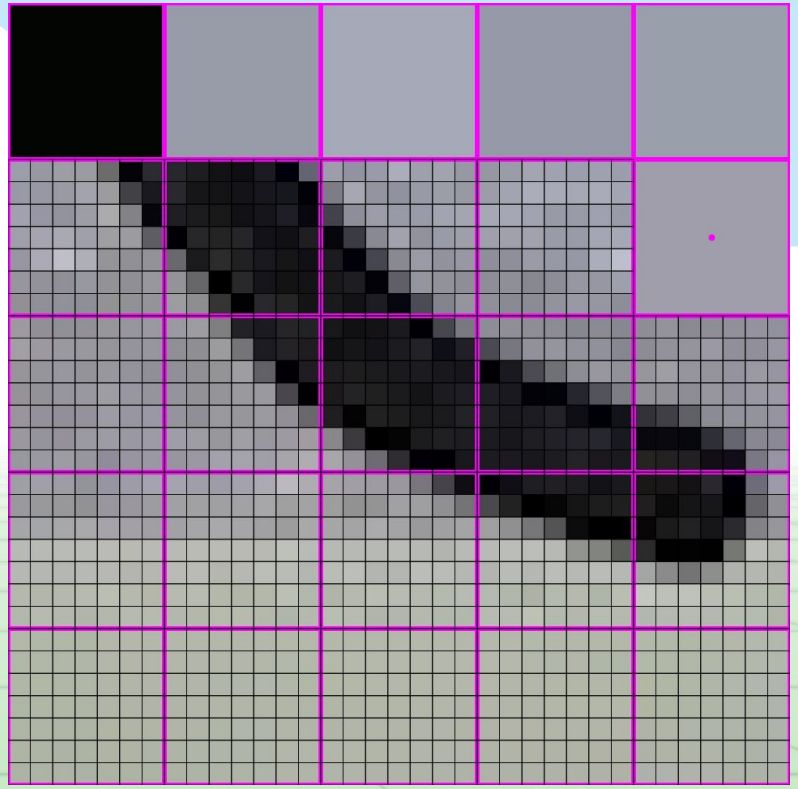
448x448 -> 64x64



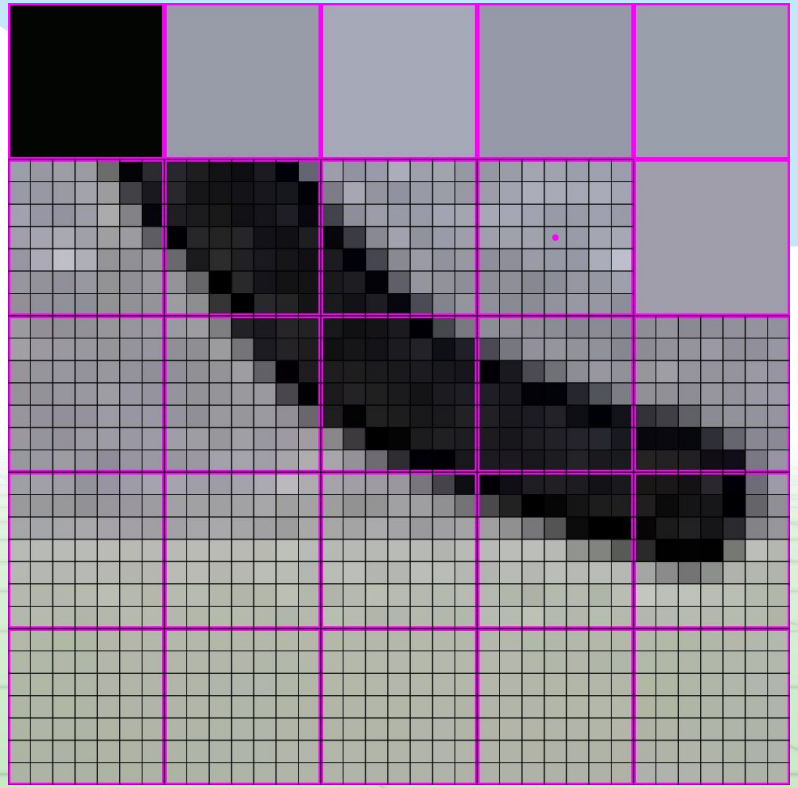


448x448 -> 64x64

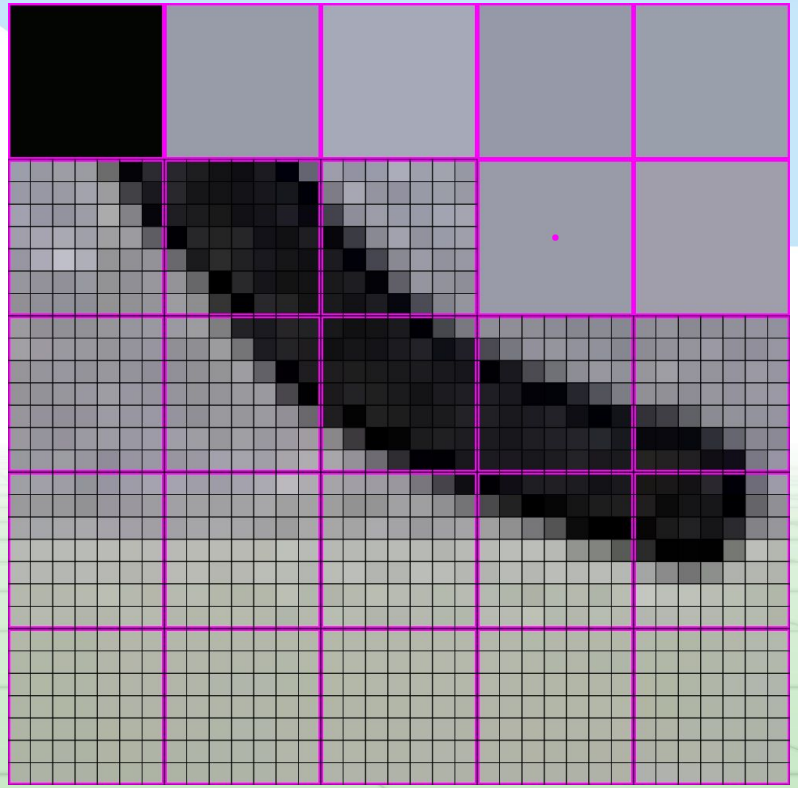




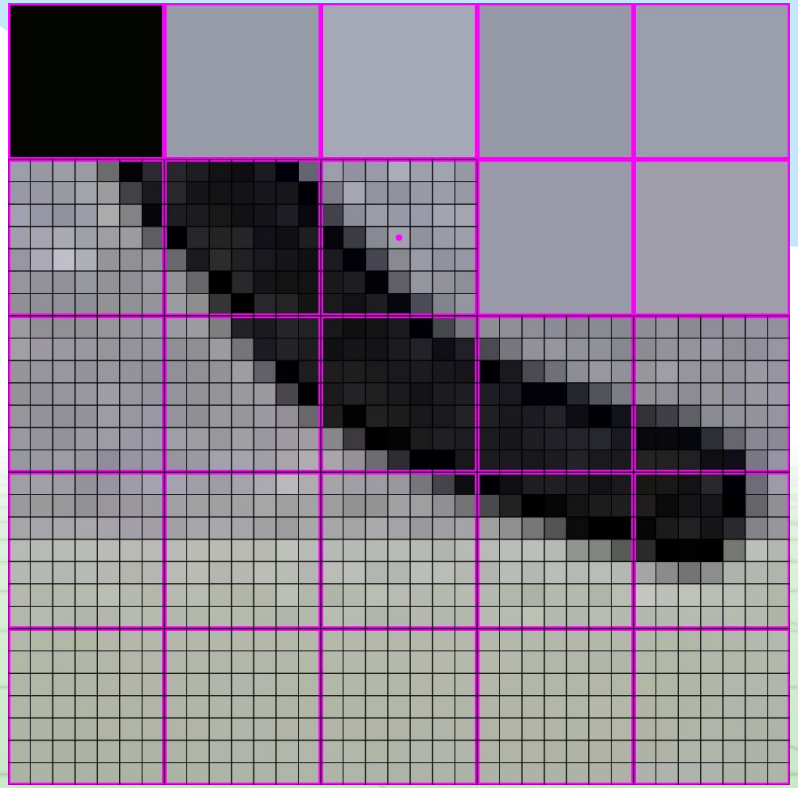
448x448 -> 64x64



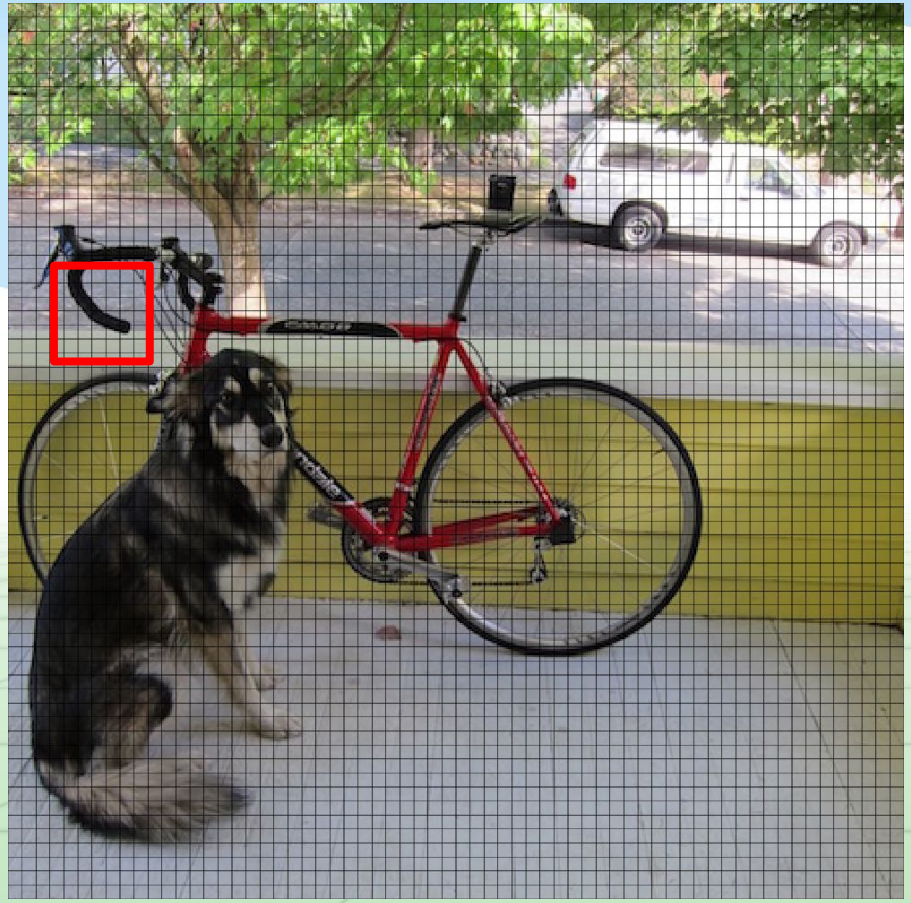
448x448 -> 64x64

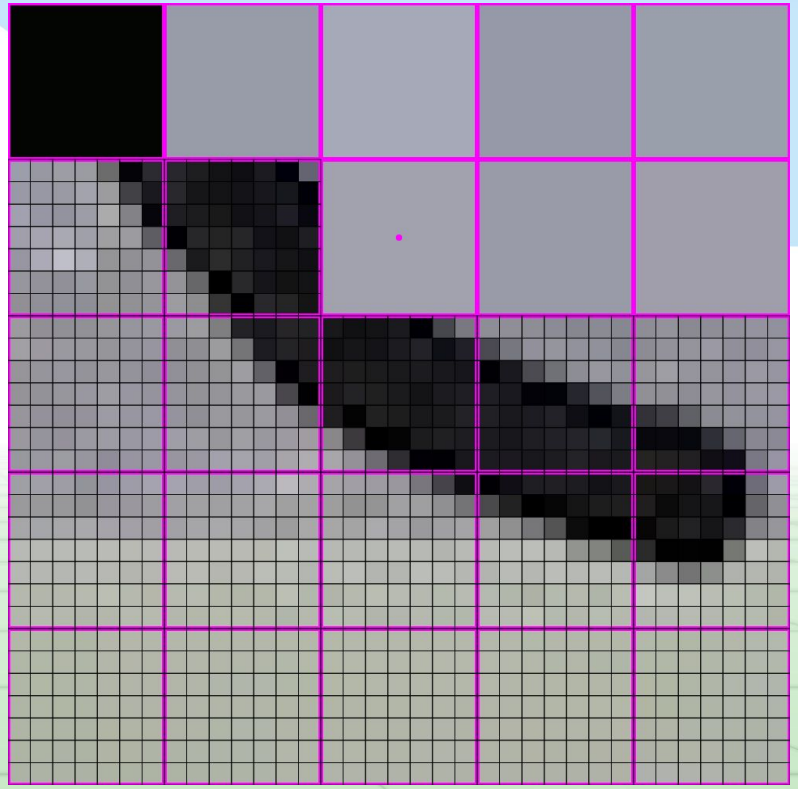


448x448 -> 64x64

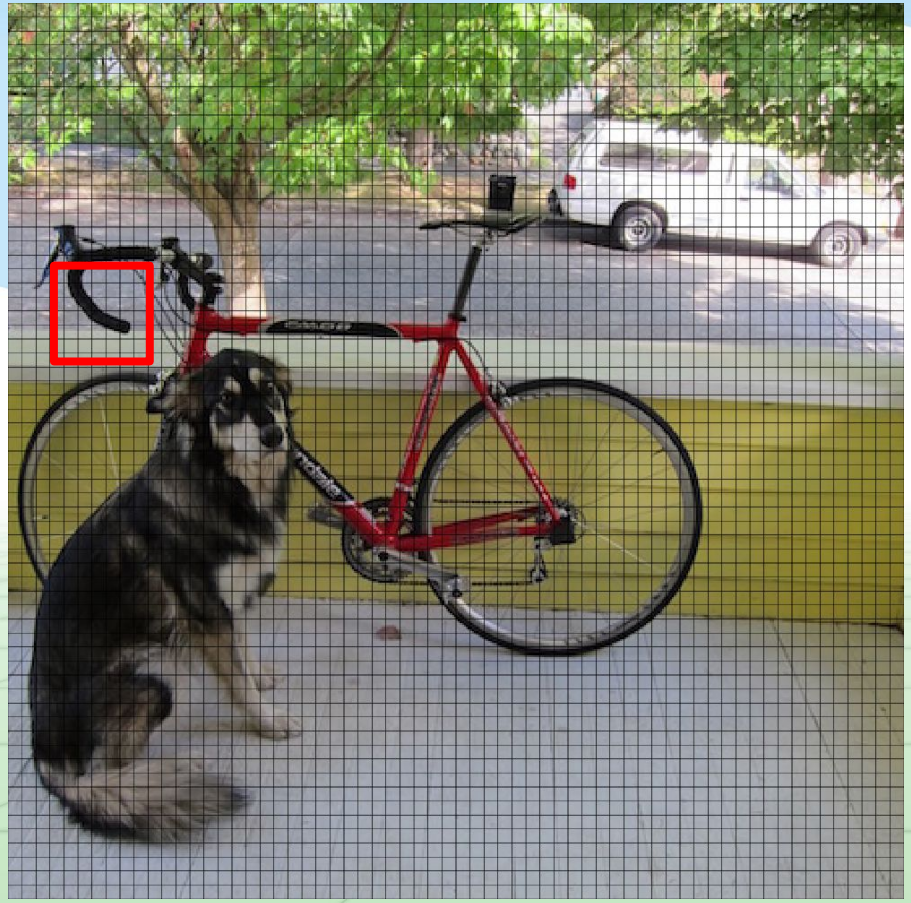


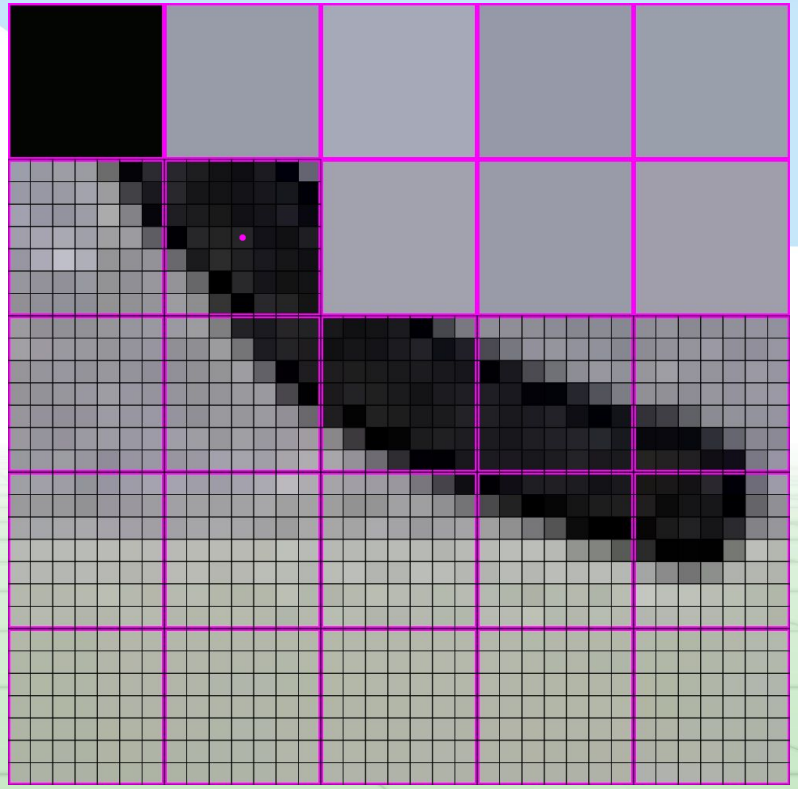
448x448 -> 64x64



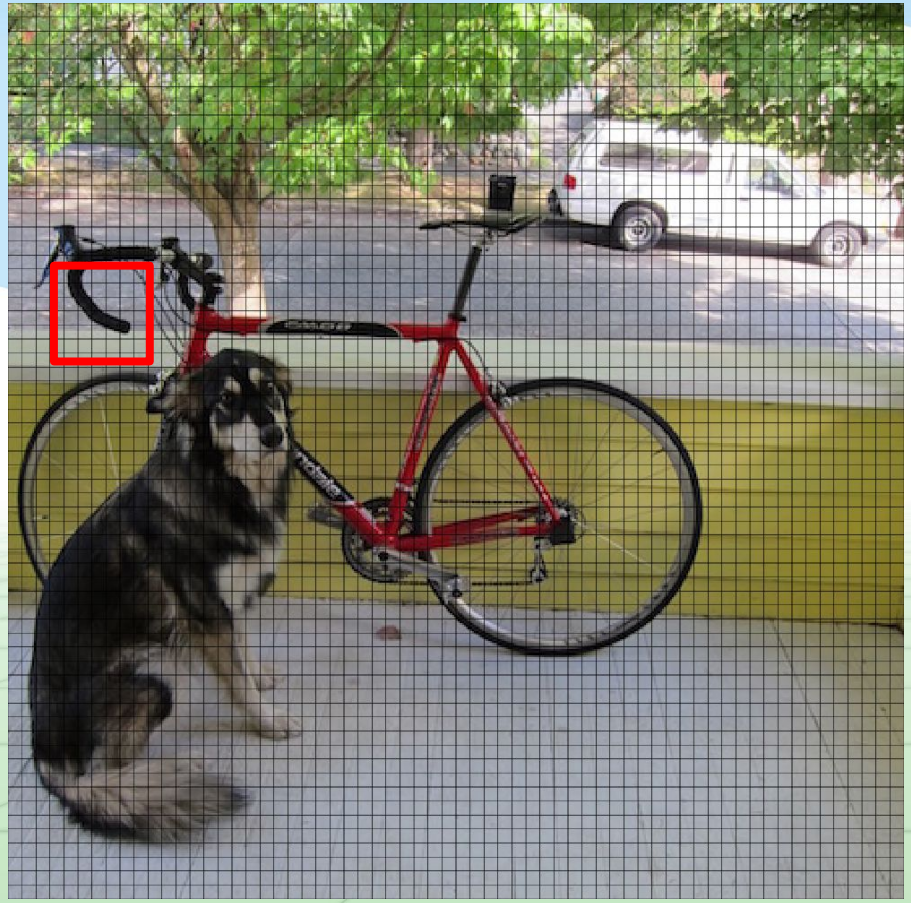


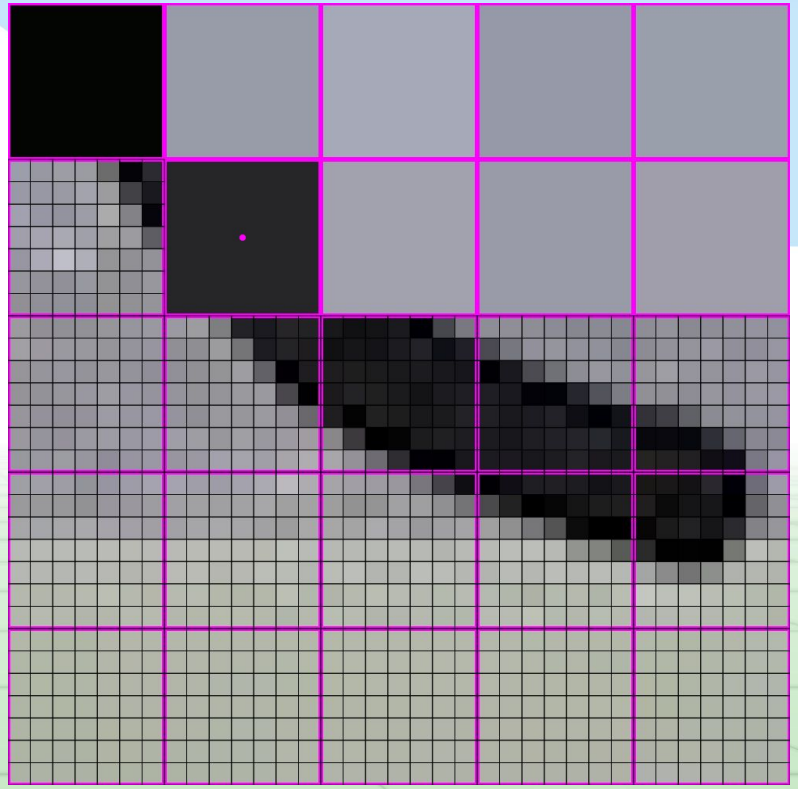
448x448 -> 64x64



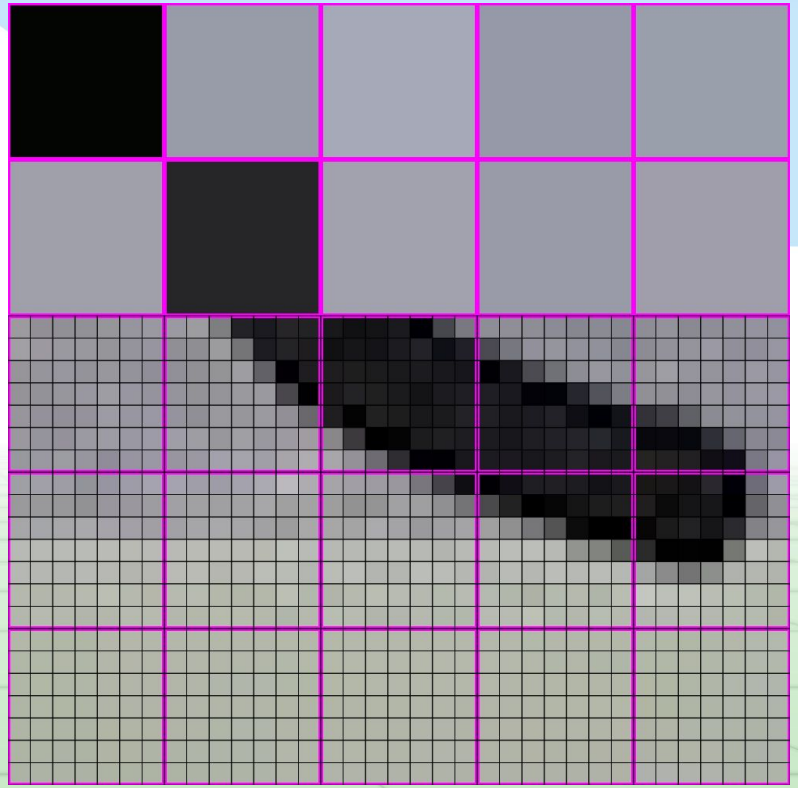


448x448 -> 64x64

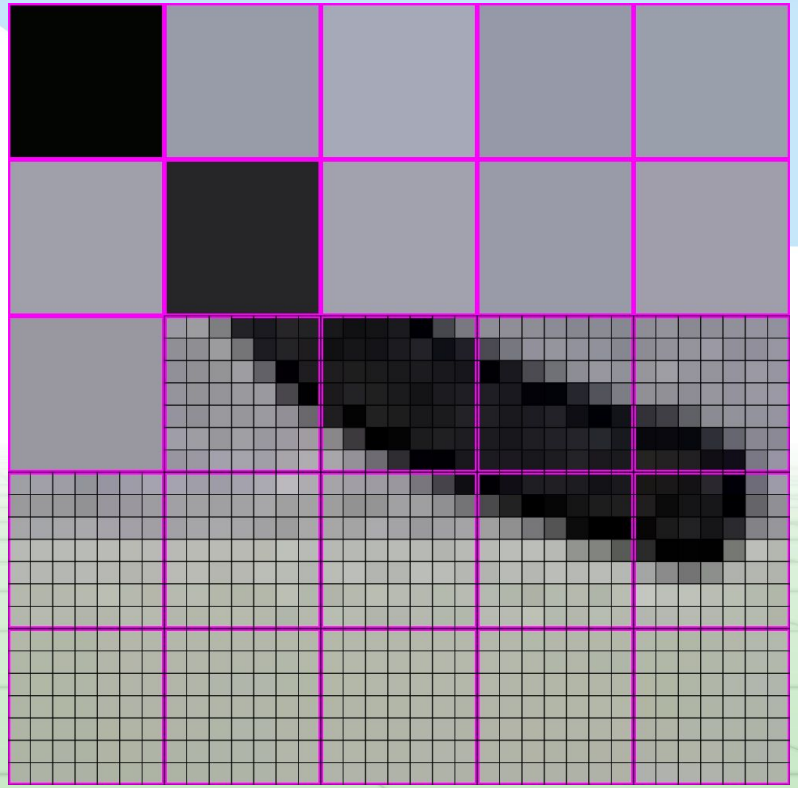




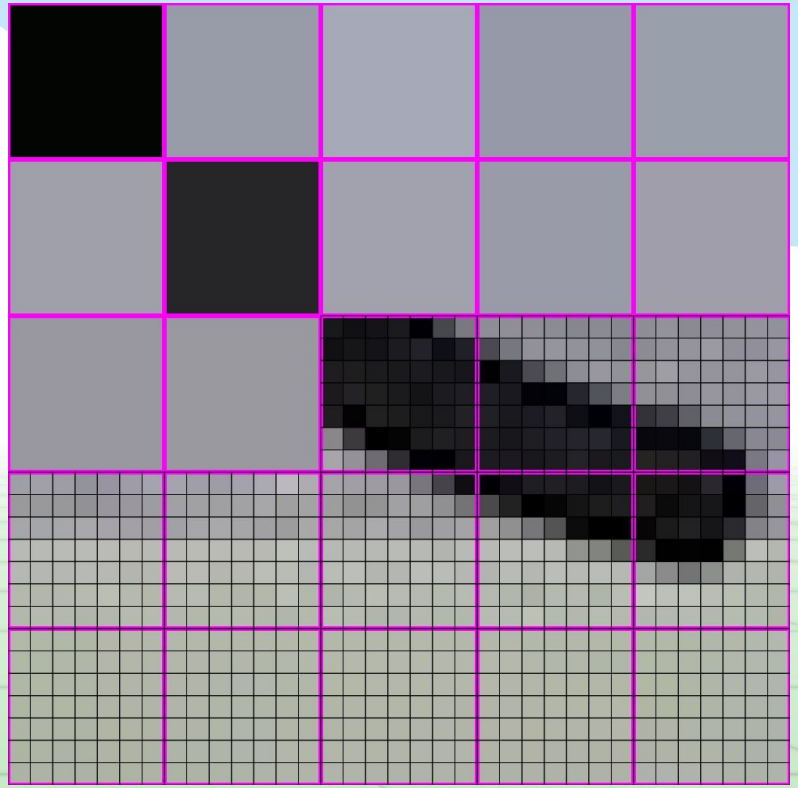
448x448 -> 64x64



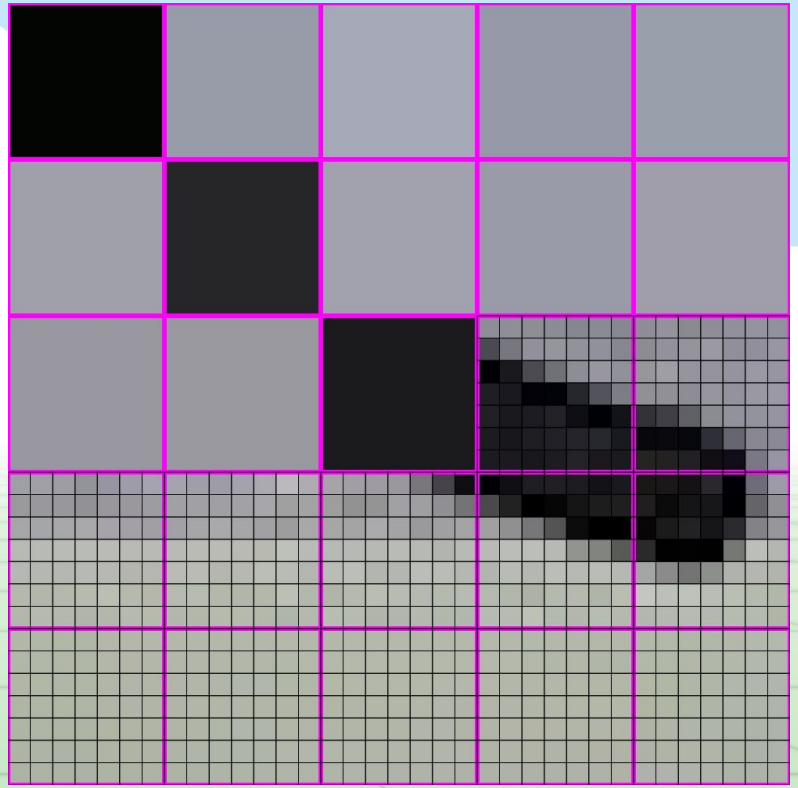
448x448 -> 64x64



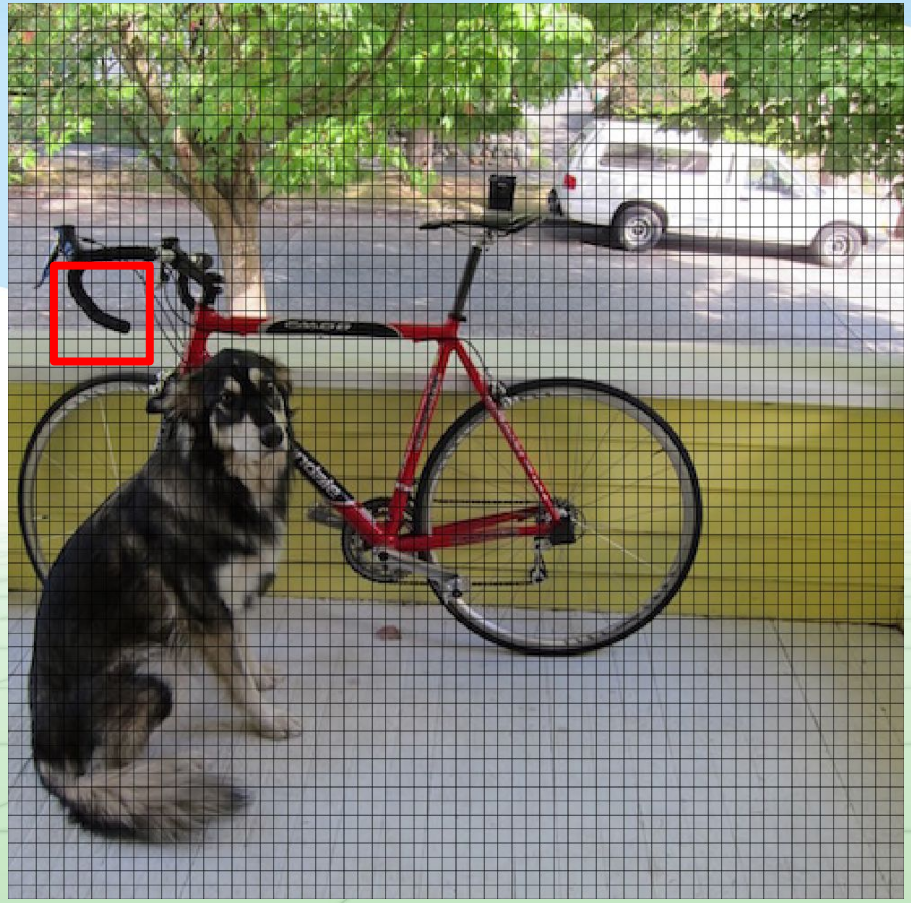
448x448 -> 64x64

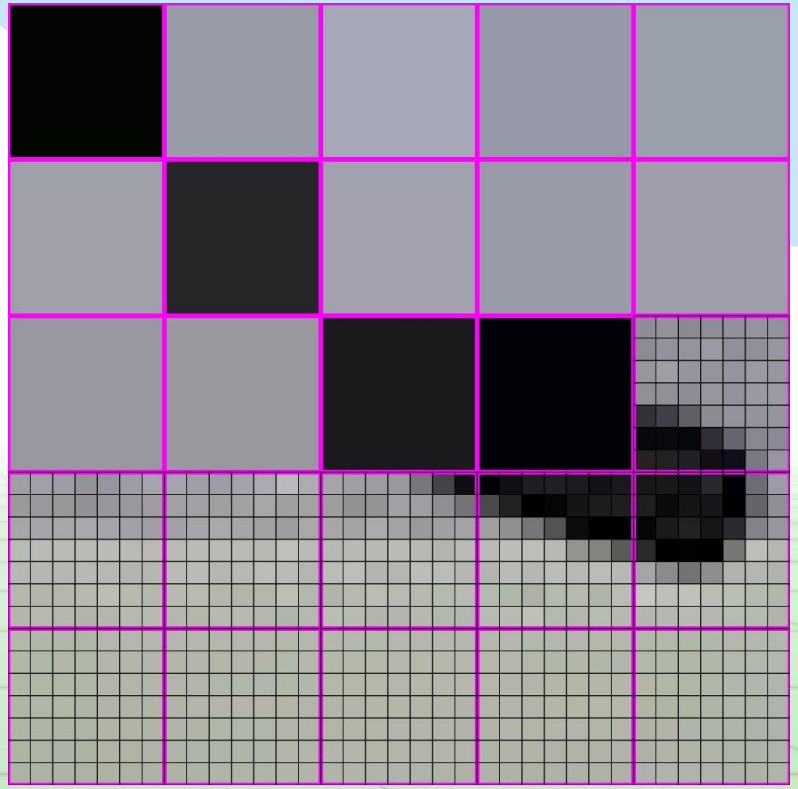


448x448 -> 64x64

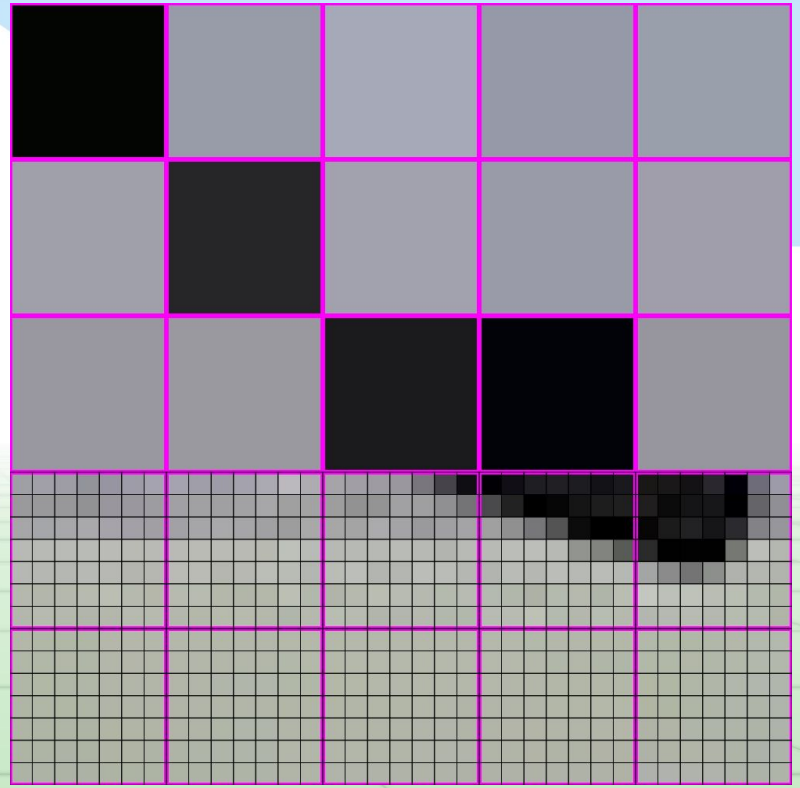


448x448 -> 64x64

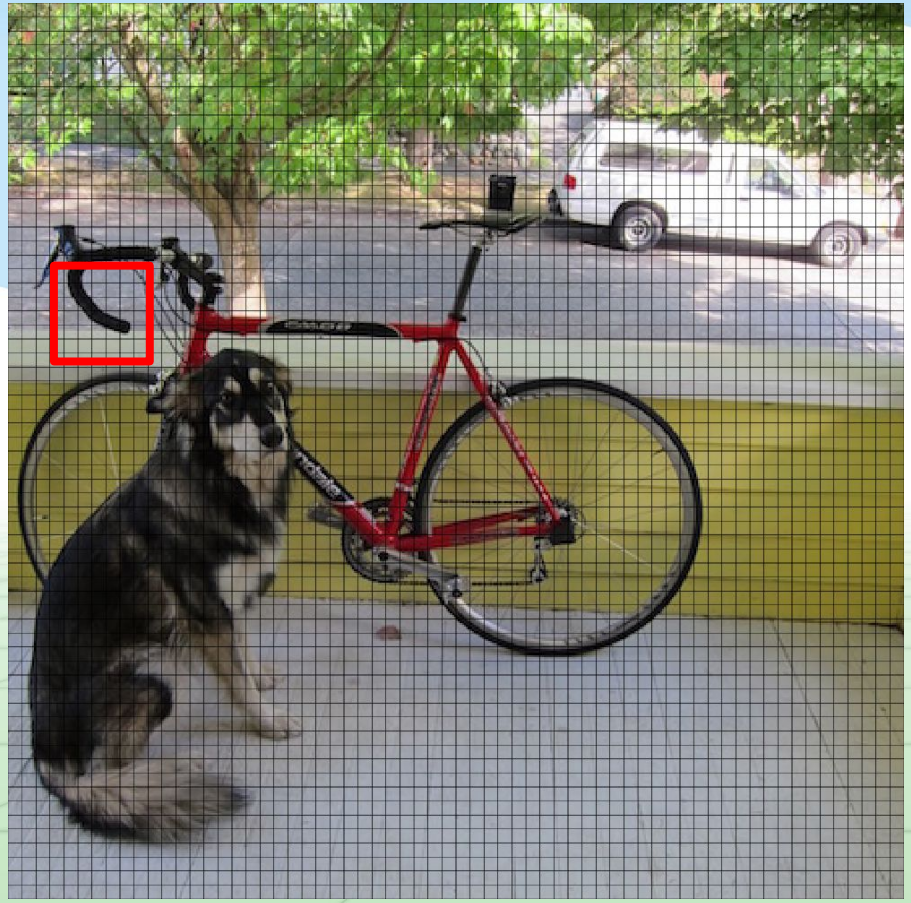


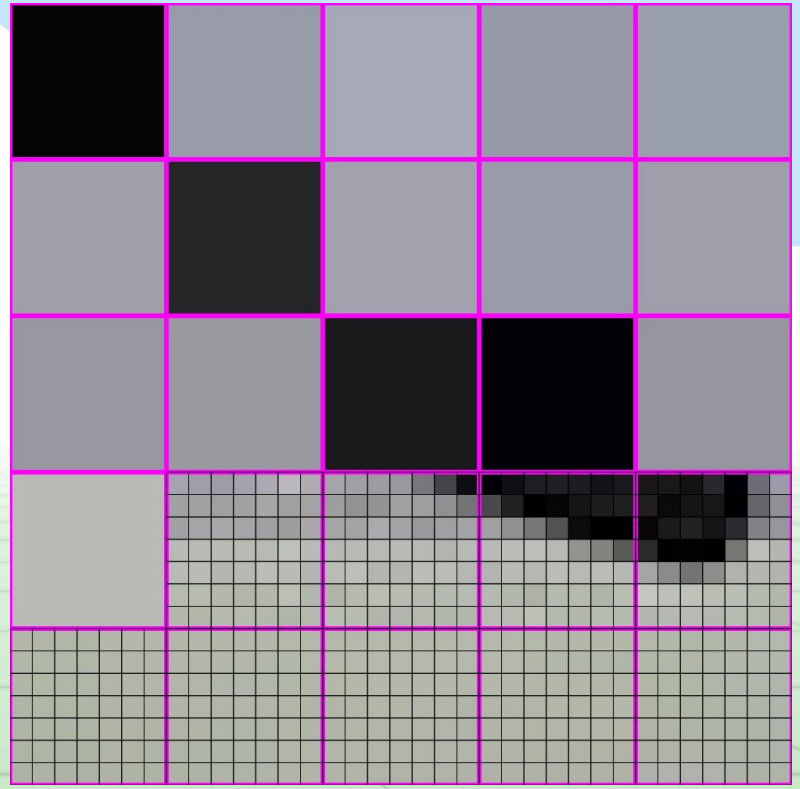


448x448 -> 64x64

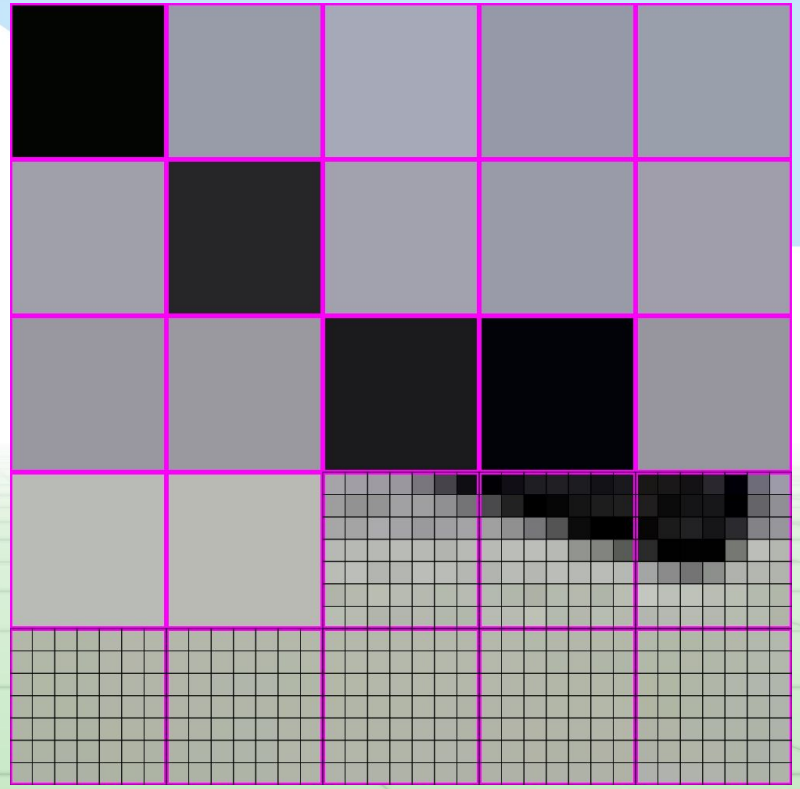


448x448 -> 64x64

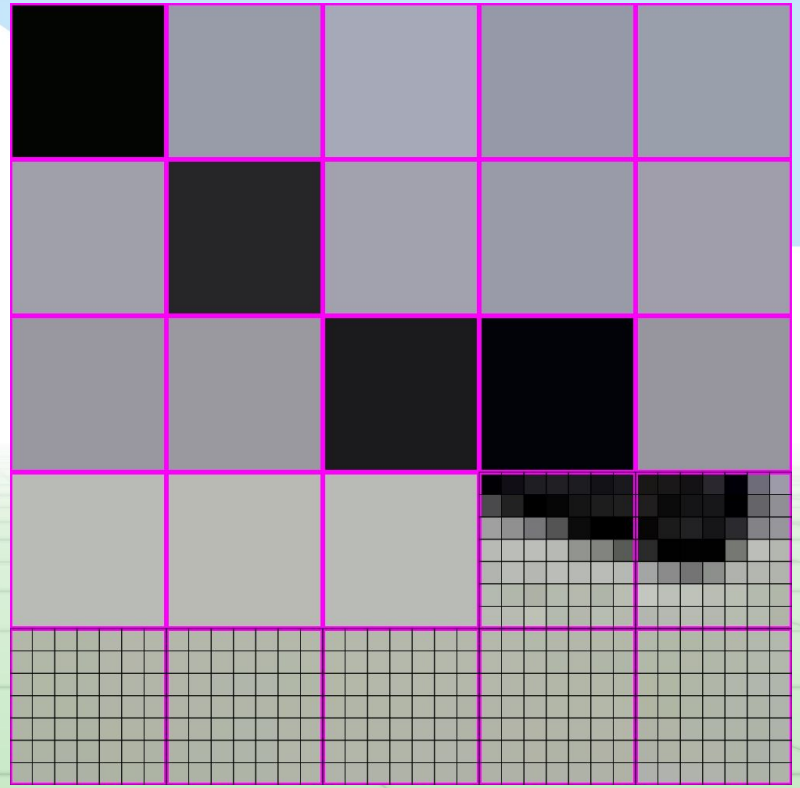




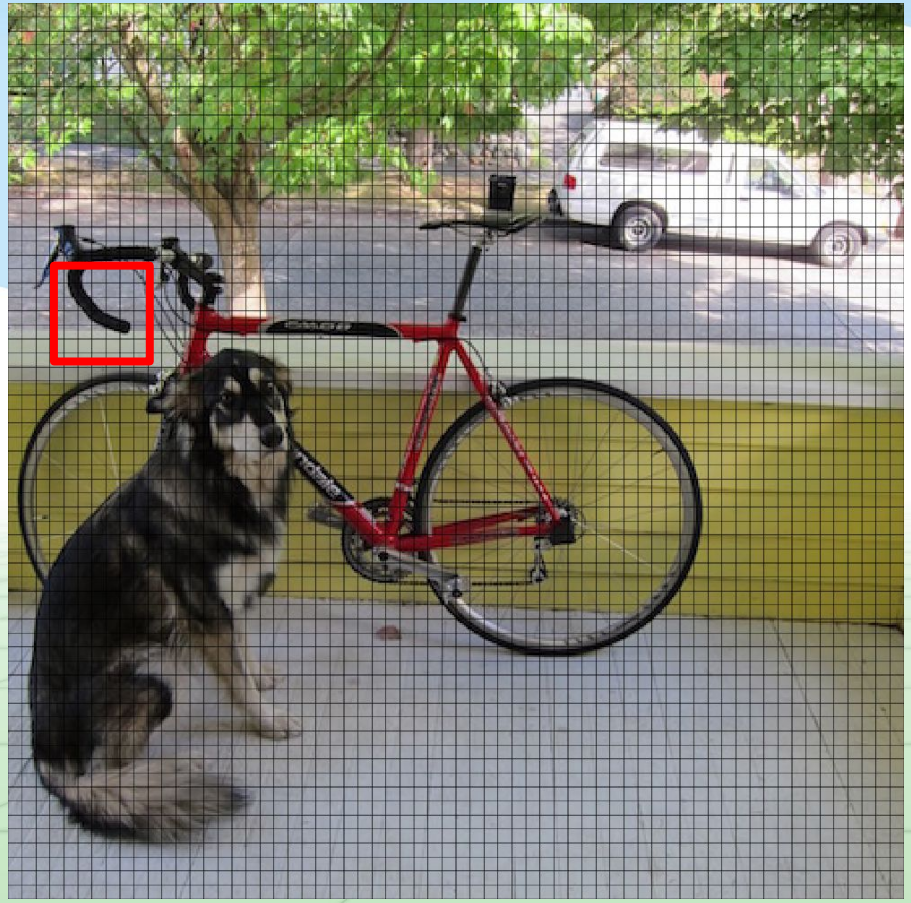
448x448 -> 64x64

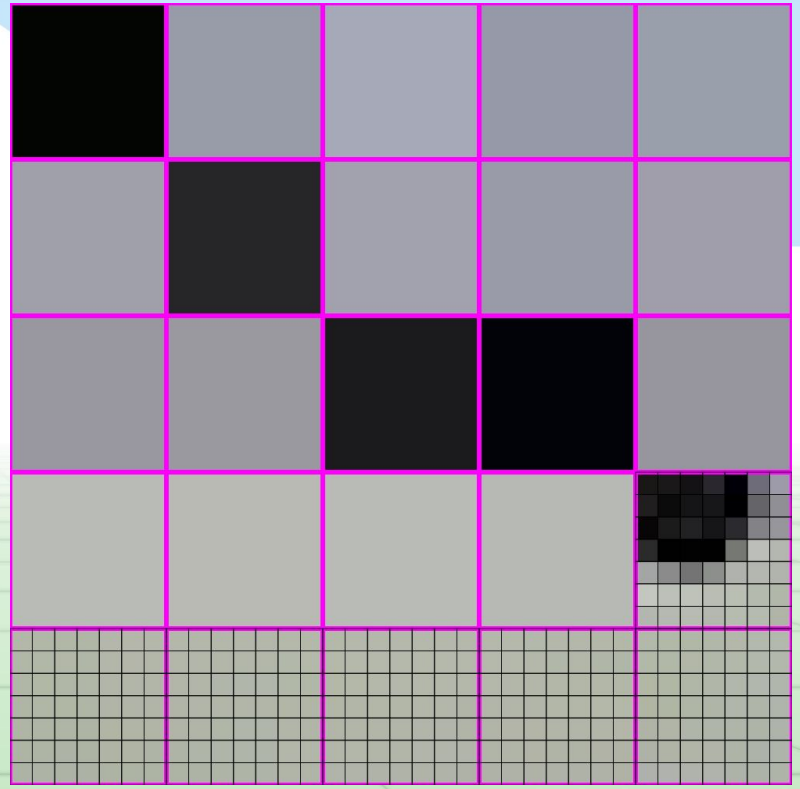


448x448 -> 64x64

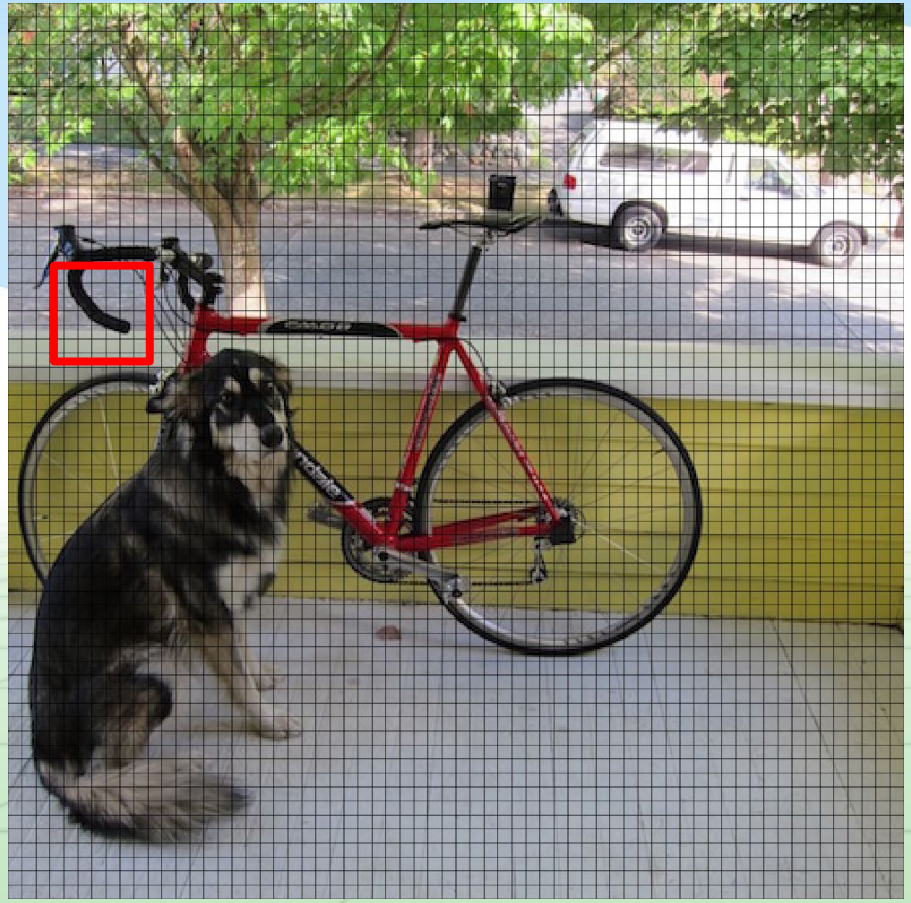


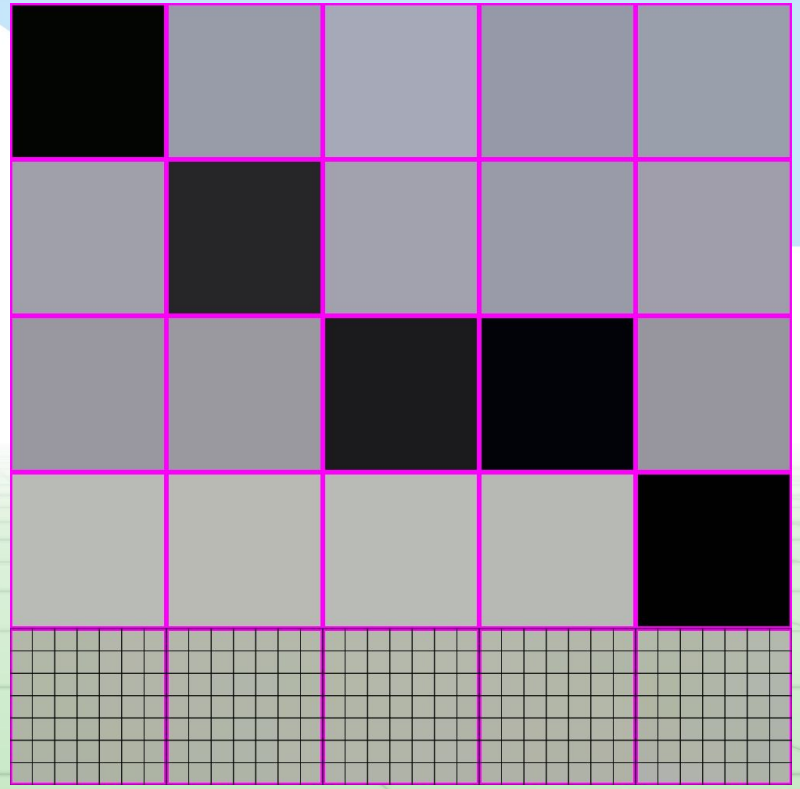
448x448 -> 64x64



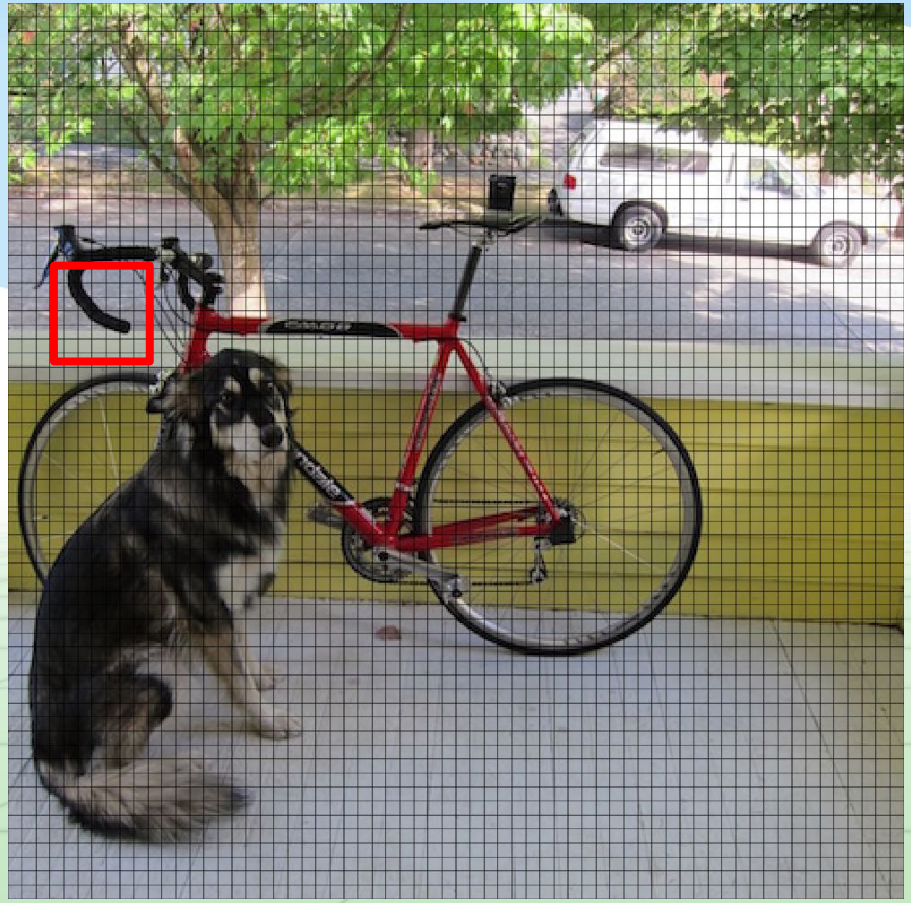


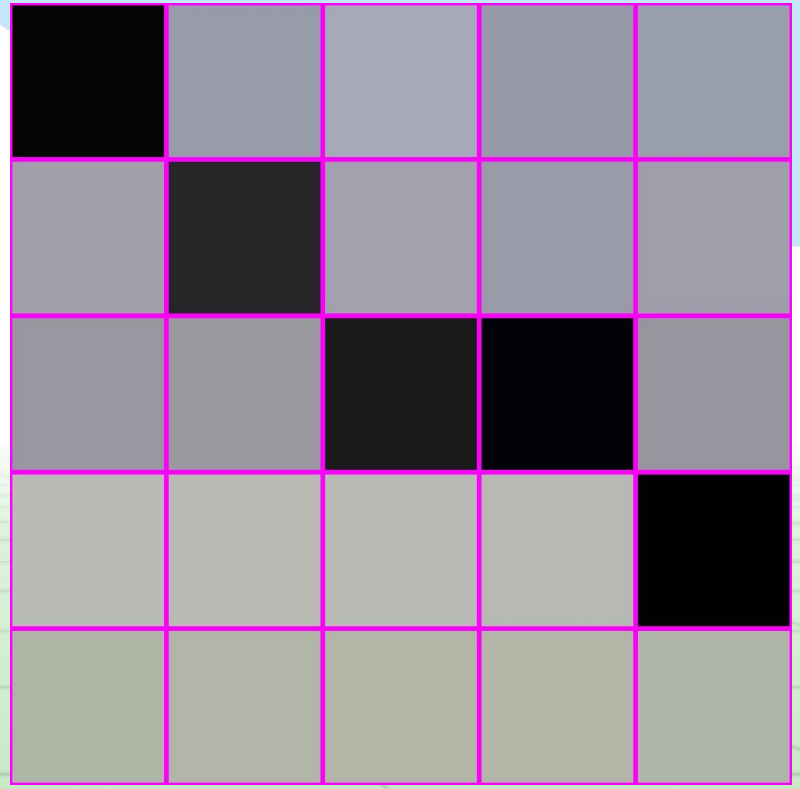
448x448 -> 64x64



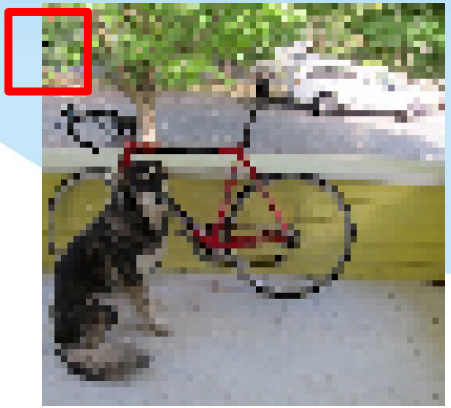


448x448 -> 64x64

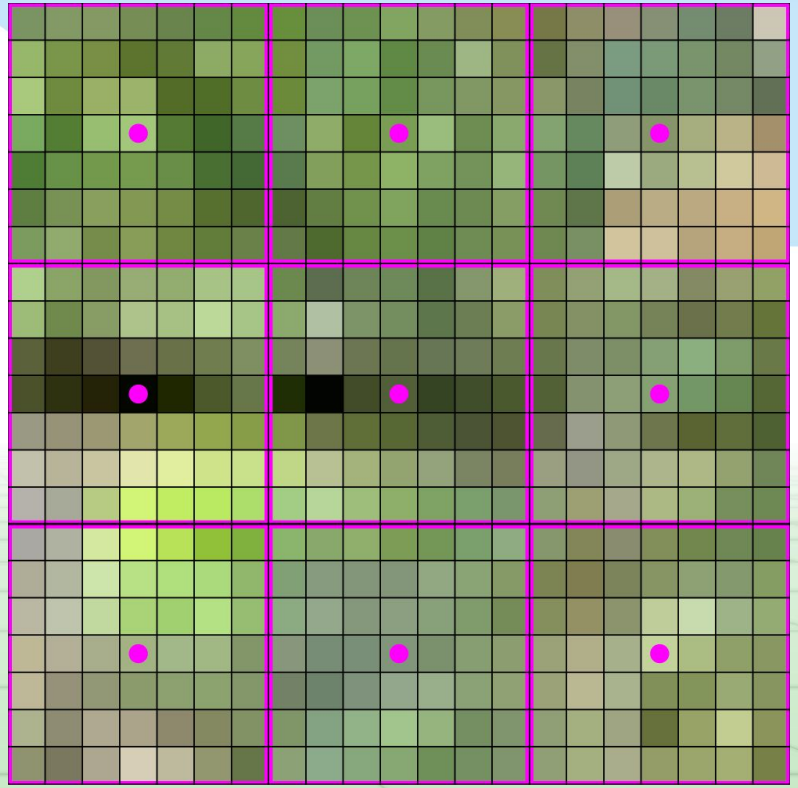




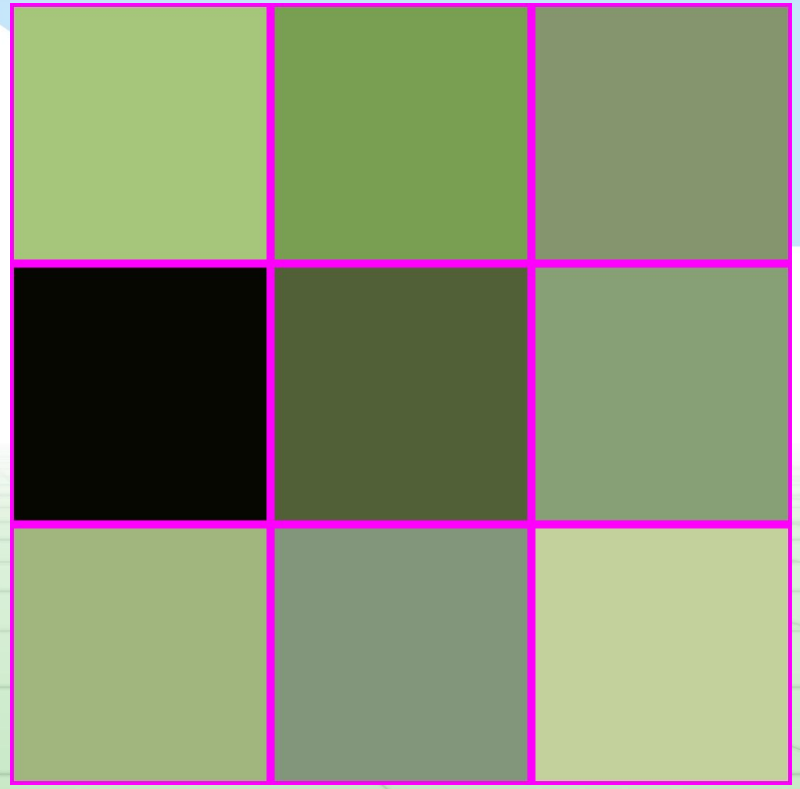
448x448 -> 64x64



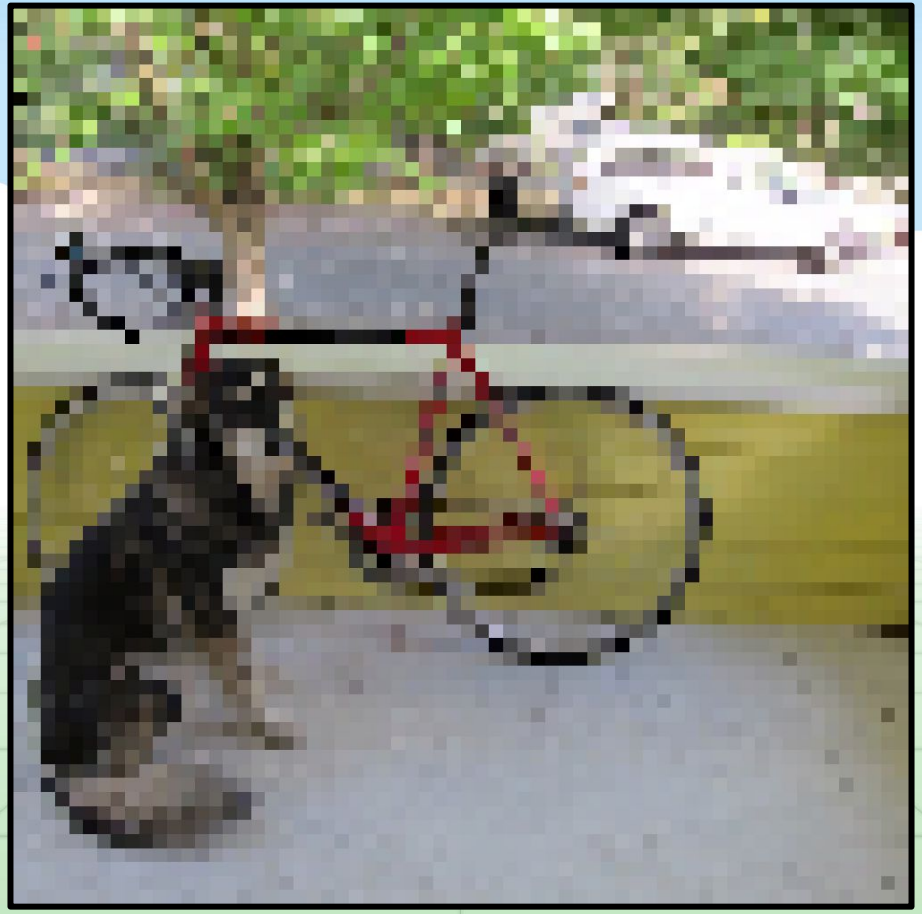
448x448 -> 64x64



448x448 -> 64x64



IS THIS ALL THERE IS??

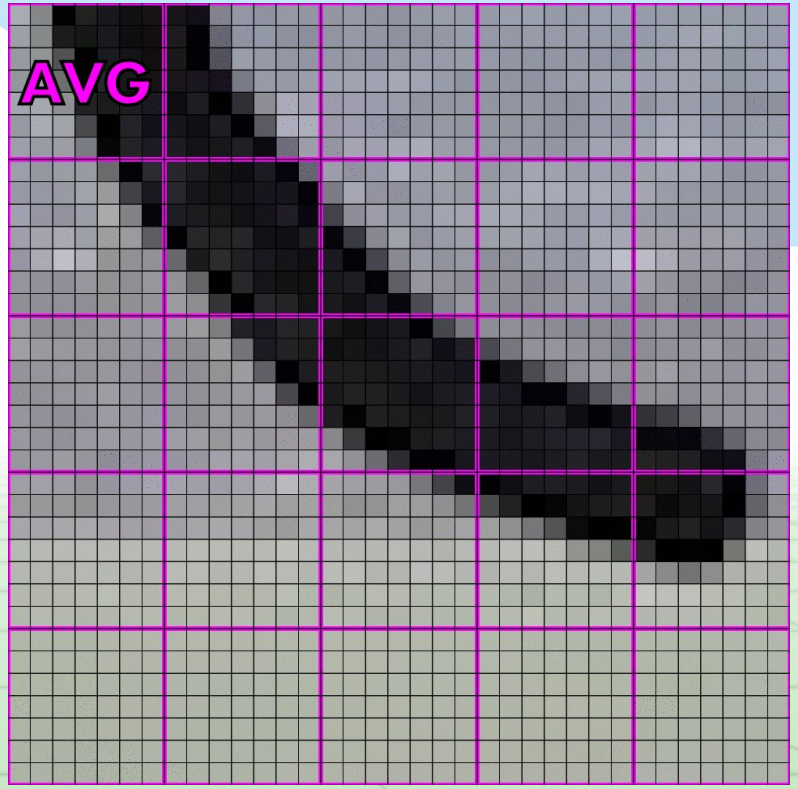


THERE IS A BETTER WAY!

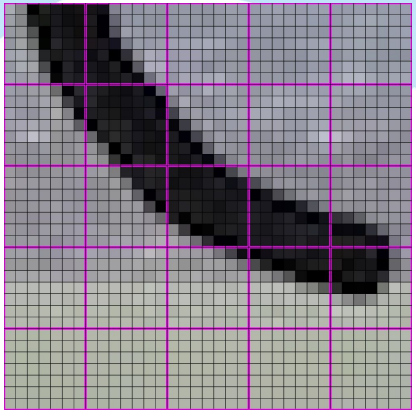
LOOK AT HOW MUCH BETTER

How do?

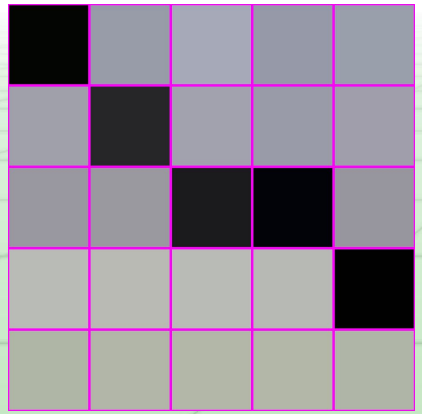
How do? Averaging!

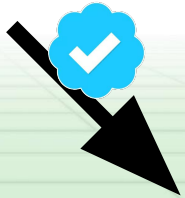


How do? Averaging!

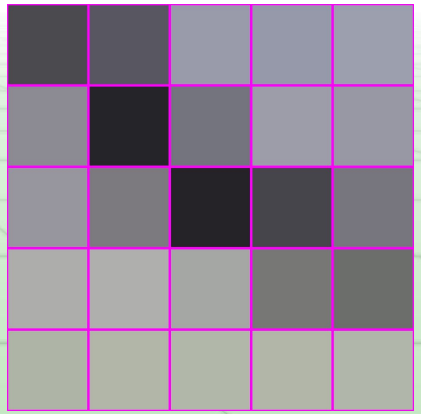


“interpolation”

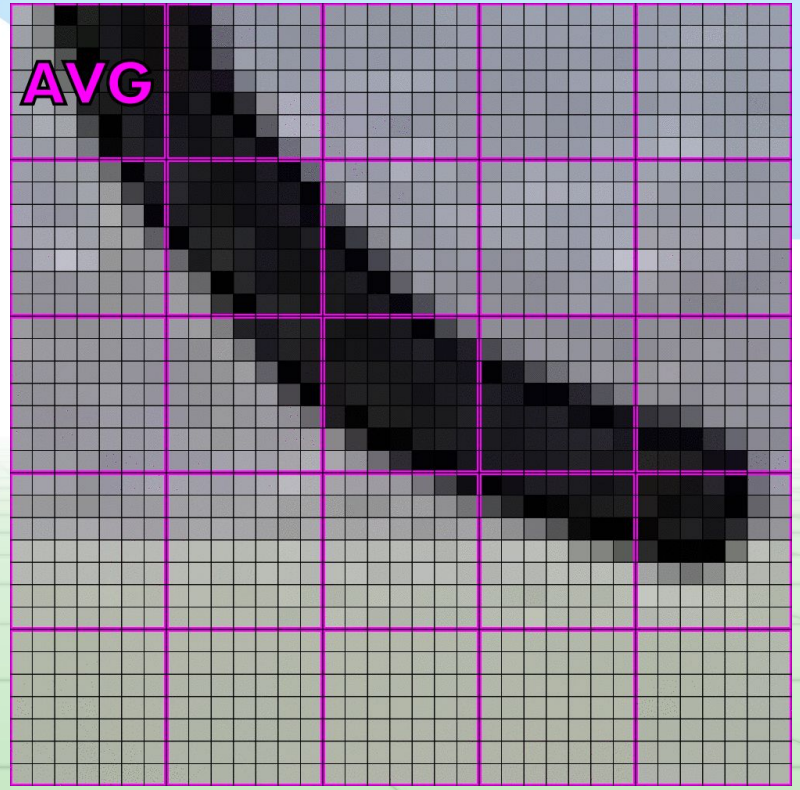




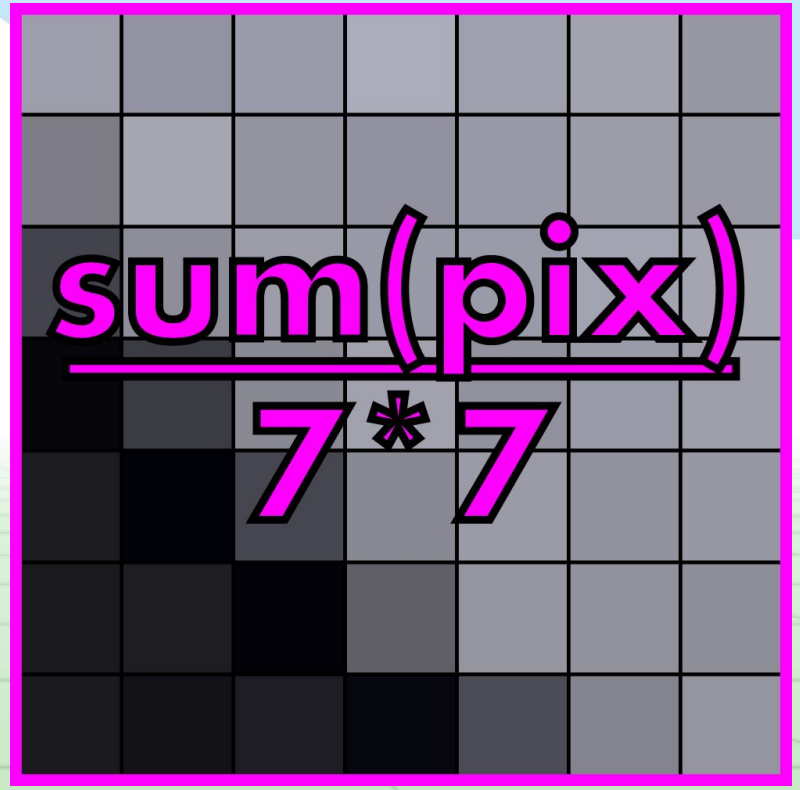
averaging



What is averaging?



What is averaging? A weighted sum



What is averaging? A weighted sum

$$\text{sum}\left[\frac{1}{49}\begin{bmatrix} 1\times & 1\times & 1\times & 1\times & 1\times & 1\times & 1\times \\ 1\times & 1\times & 1\times & 1\times & 1\times & 1\times & 1\times \\ 1\times & 1\times & 1\times & 1\times & 1\times & 1\times & 1\times \\ 1\times & 1\times & 1\times & 1\times & 1\times & 1\times & 1\times \\ 1\times & 1\times & 1\times & 1\times & 1\times & 1\times & 1\times \\ 1\times & 1\times & 1\times & 1\times & 1\times & 1\times & 1\times \\ 1\times & 1\times & 1\times & 1\times & 1\times & 1\times & 1\times \end{bmatrix}\right]$$

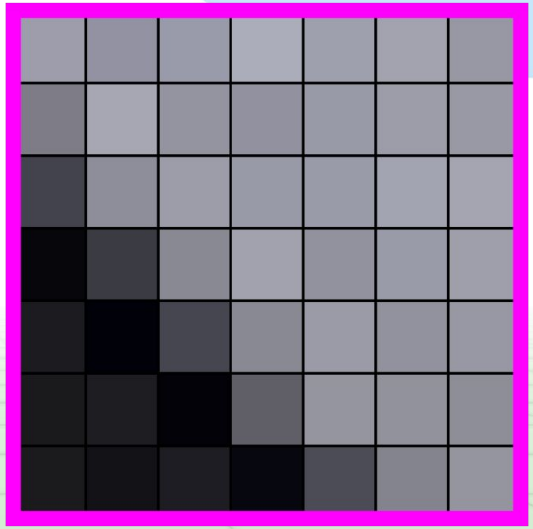
Call this operation “*convolution*”

Filter or *kernel*

$$\frac{1}{49}$$

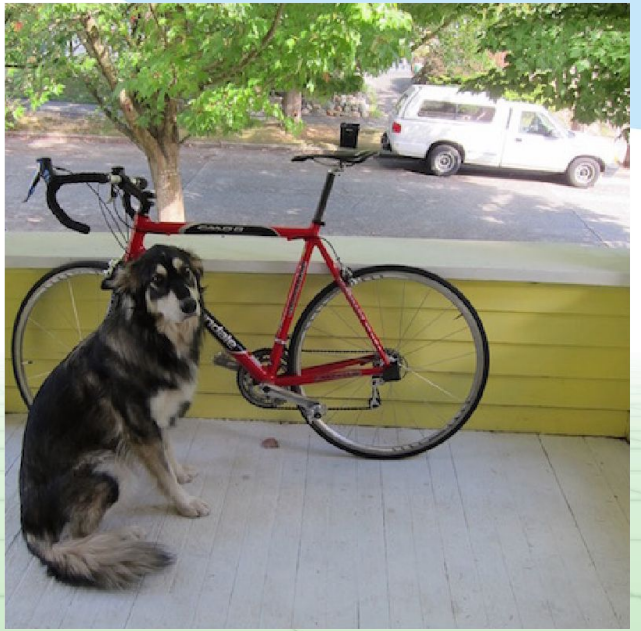
1×	1×	1×	1×	1×	1×	1×
1×	1×	1×	1×	1×	1×	1×
1×	1×	1×	1×	1×	1×	1×
1×	1×	1×	1×	1×	1×	1×
1×	1×	1×	1×	1×	1×	1×
1×	1×	1×	1×	1×	1×	1×
1×	1×	1×	1×	1×	1×	1×

$*$

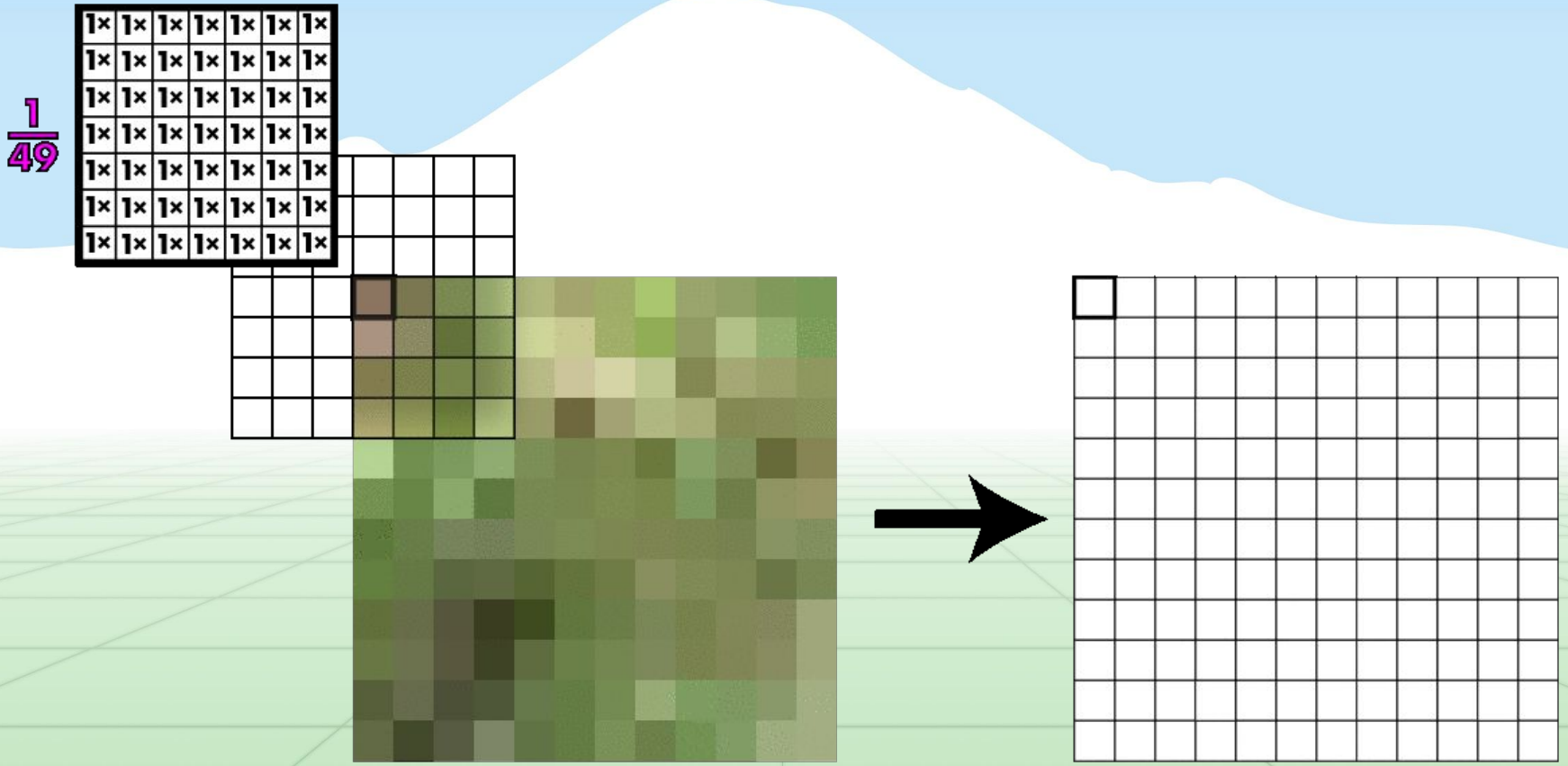


Convolutions on larger images

$$\frac{1}{49}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \quad$$

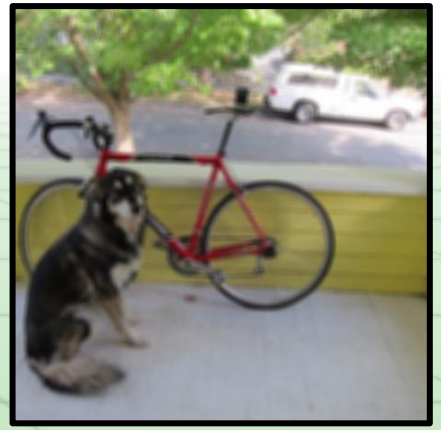


Kernel slides across image



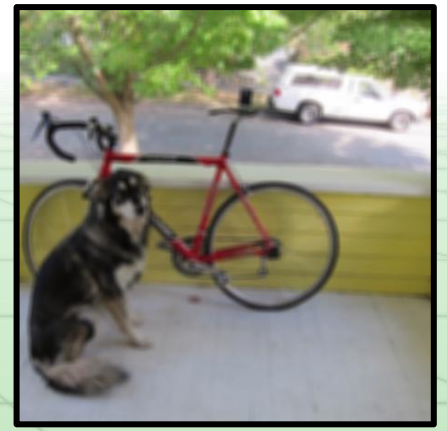
Convolutions on larger images

$$\frac{1}{49}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \quad \rightarrow$$



This is called box filter

$\frac{1}{49}$ * 1× 1× 1× 1× 1× 1× 1×
1× 1× 1× 1× 1× 1× 1×
1× 1× 1× 1× 1× 1× 1×
1× 1× 1× 1× 1× 1× 1×
1× 1× 1× 1× 1× 1× 1×
1× 1× 1× 1× 1× 1× 1×

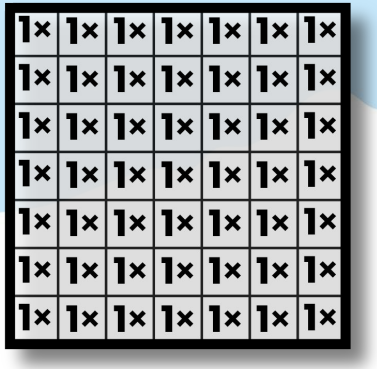


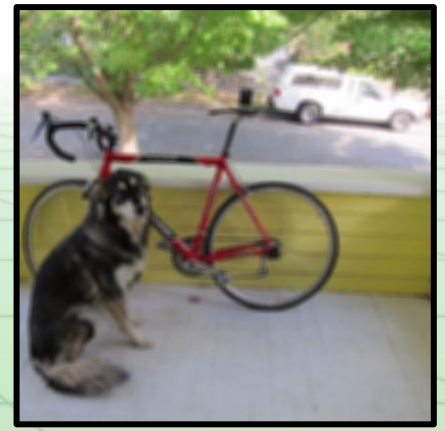
Box filters

N

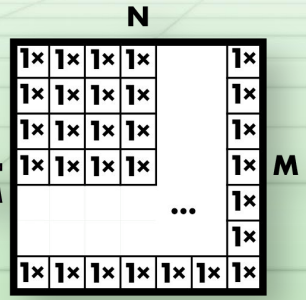
$\frac{1}{N*M}$ 1× 1× 1× 1× 1×
1× 1× 1× 1× 1×
1× 1× 1× 1× 1×
1× 1× 1× 1× 1× M
...
1×
1× 1× 1× 1× 1× 1× 1×

Box filters smooth image

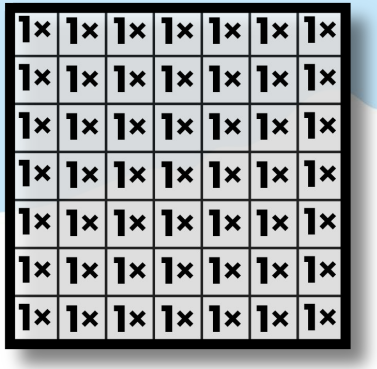
$\frac{1}{49}$ * 

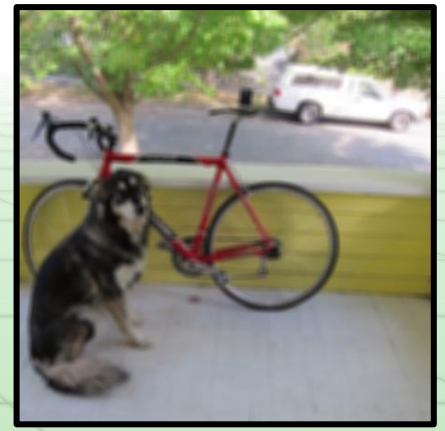


Box filters

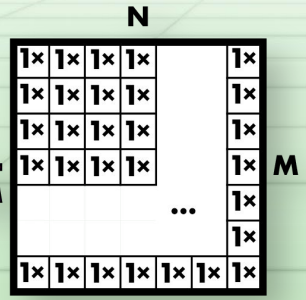
$\frac{1}{N*M}$ 

Box filters smooth image

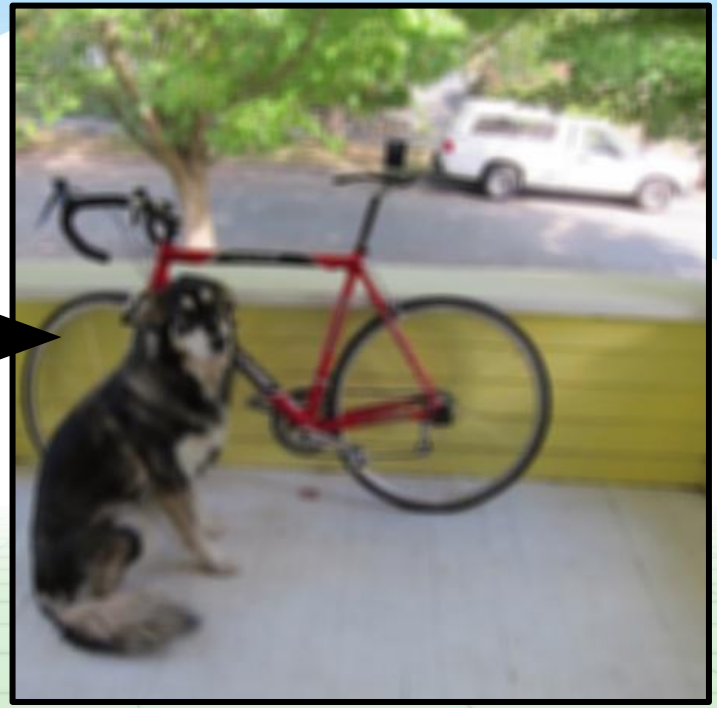
$\frac{1}{49}$ * 



Box filters

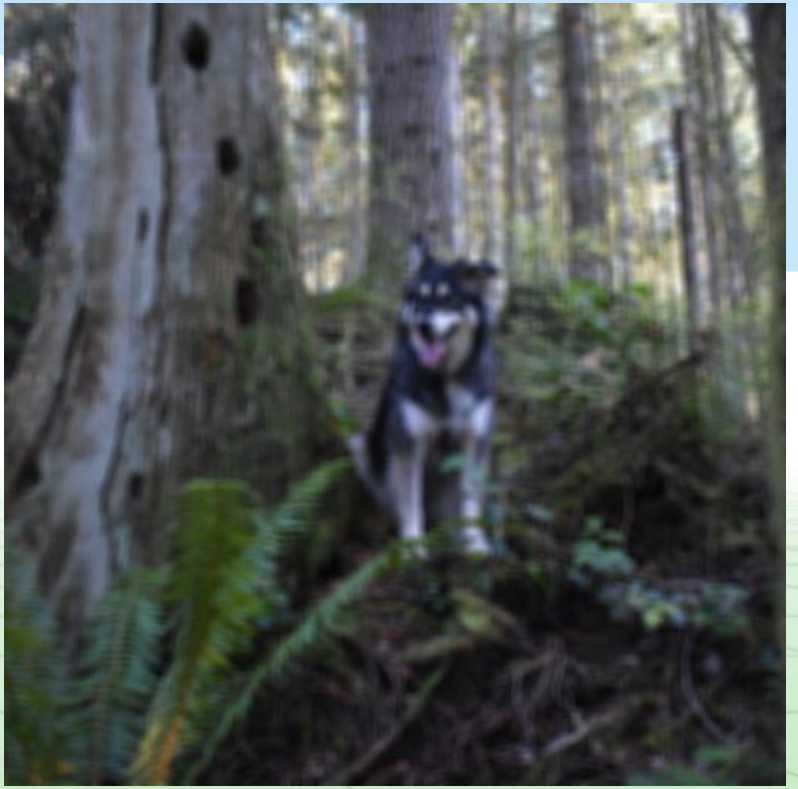
$\frac{1}{N*M}$ 

Now we resize our smoothed image



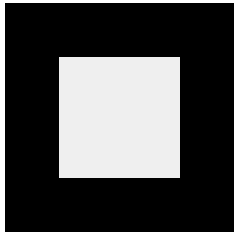
So much better!

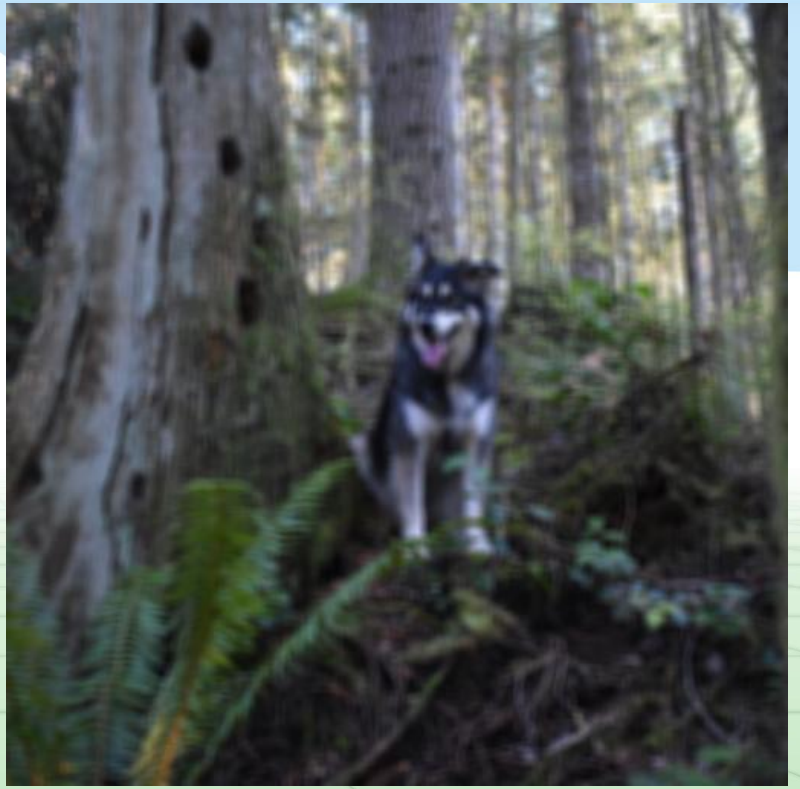
Box filters have artifacts



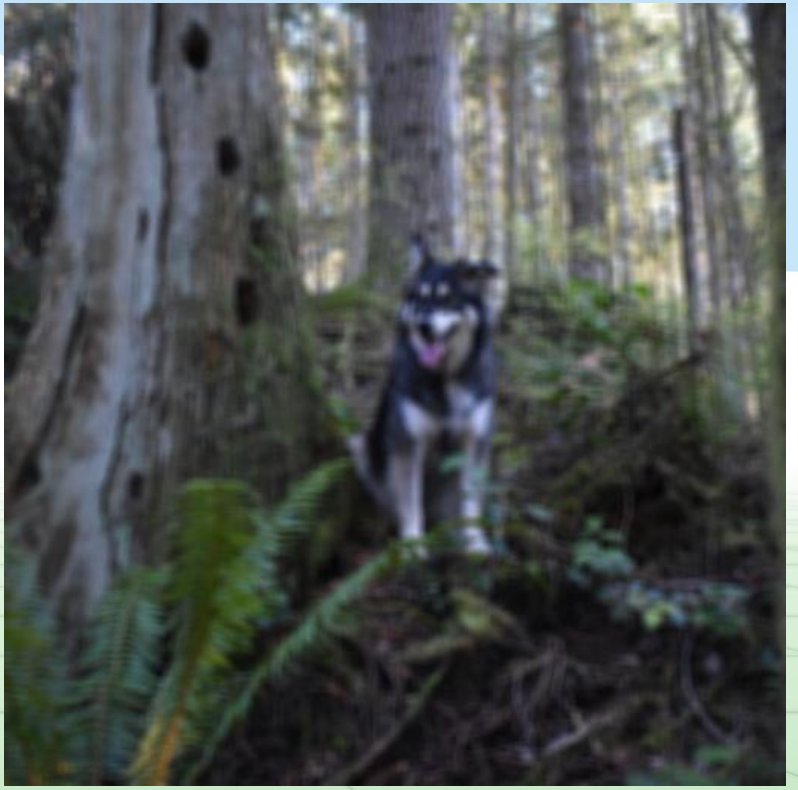
Box filters have artifacts

Box filters: vertical + horizontal streaking

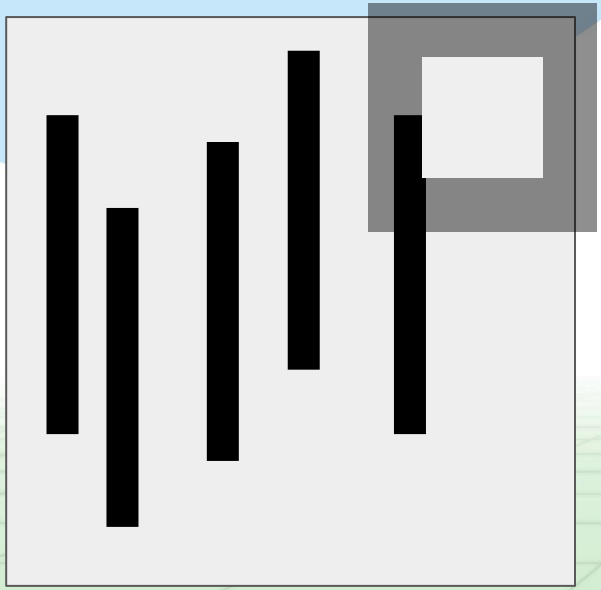


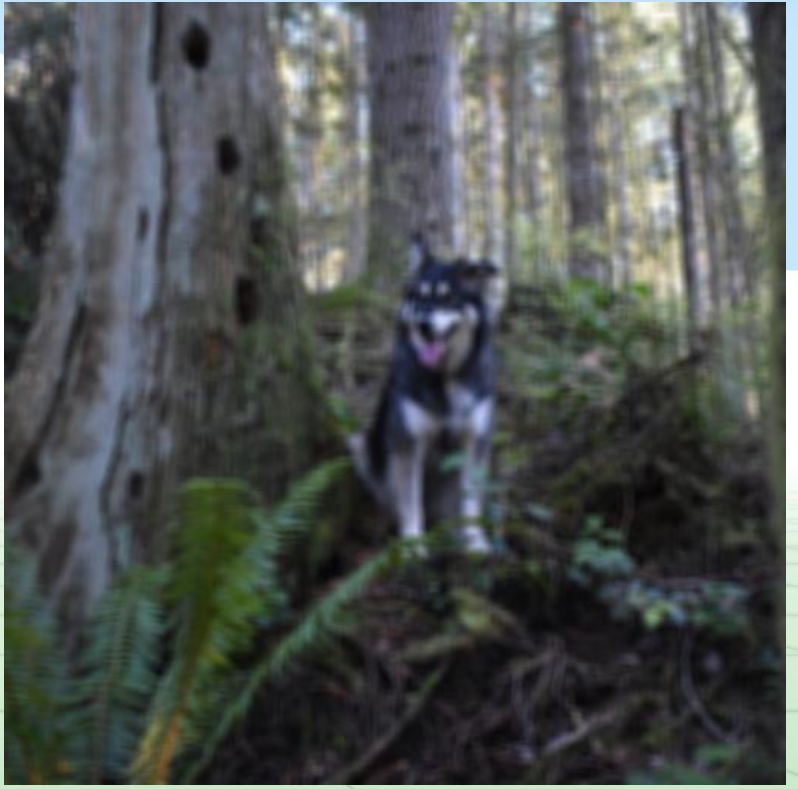


Box filters: vertical + horizontal streaking

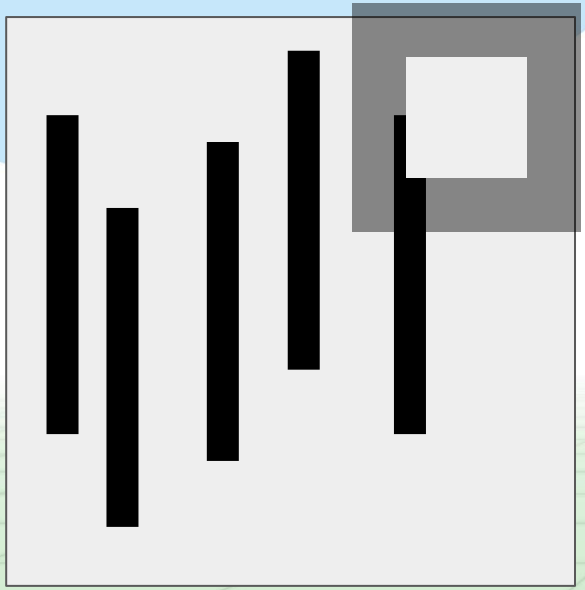


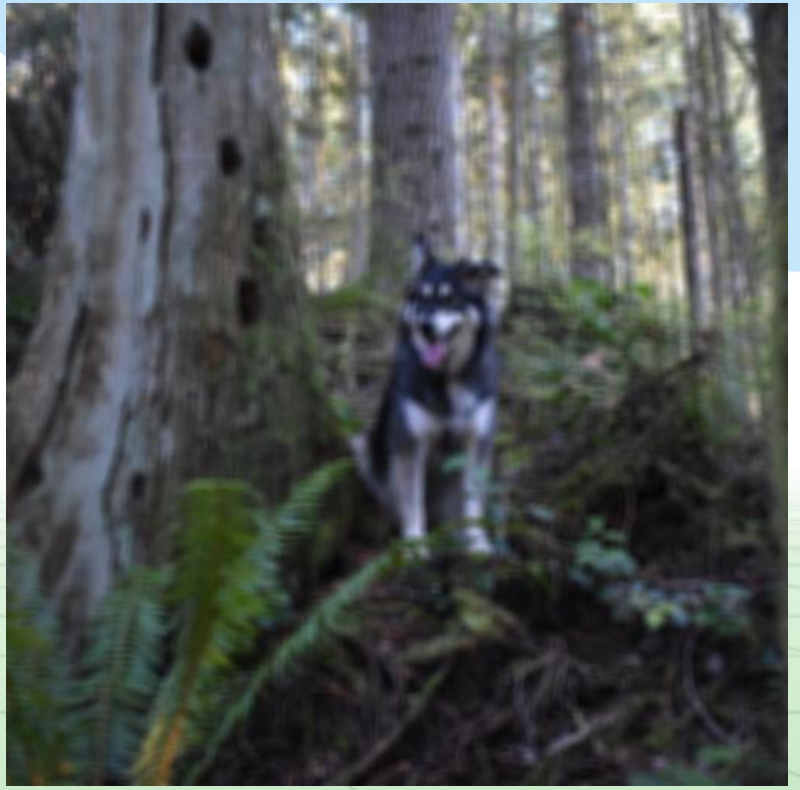
Box filters: vertical + horizontal streaking



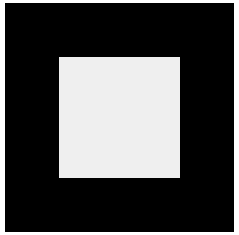


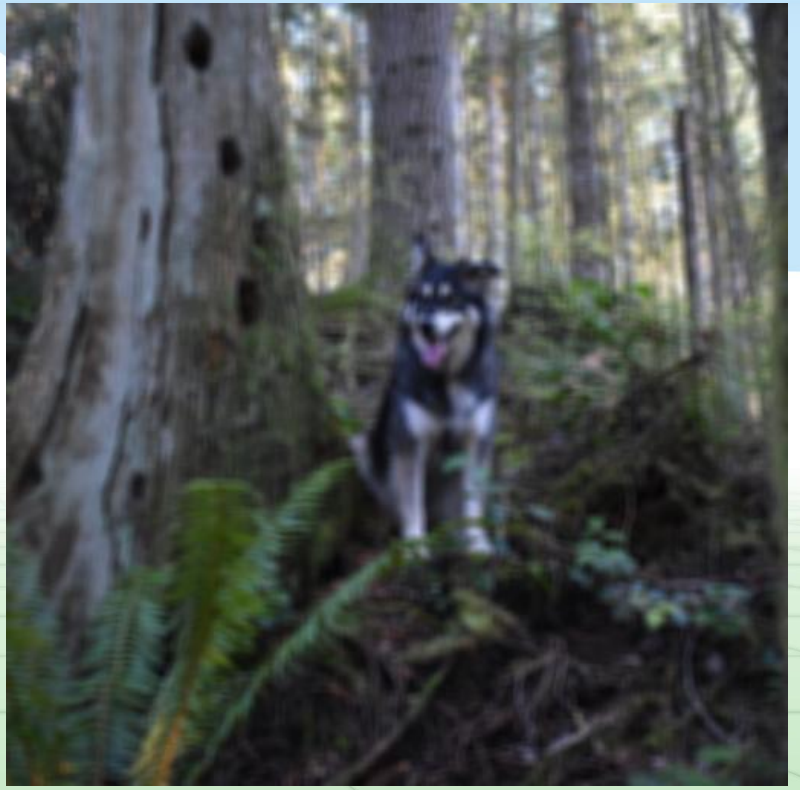
Box filters: vertical + horizontal streaking





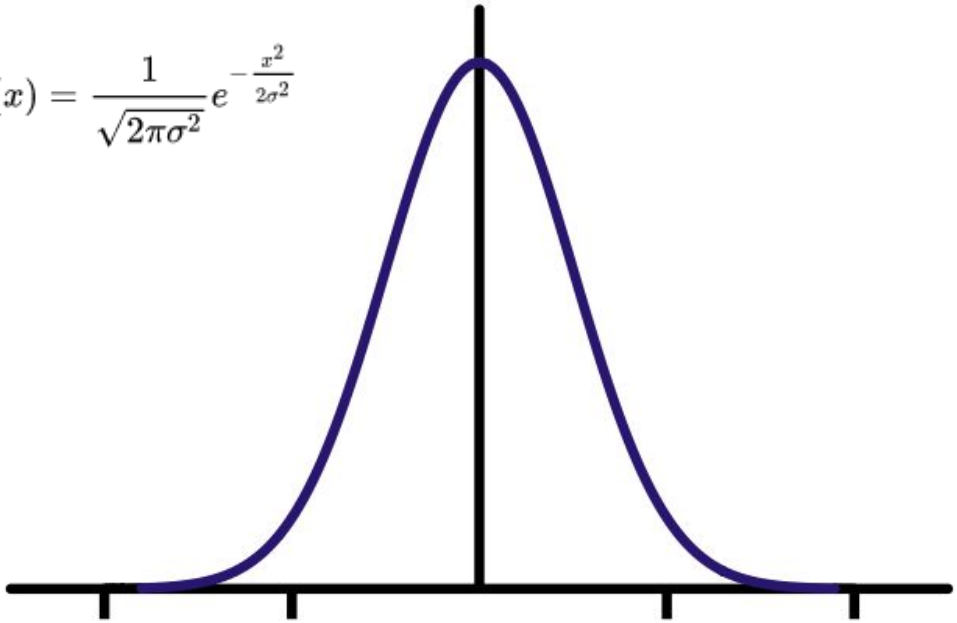
We want a smoothly weighted kernel





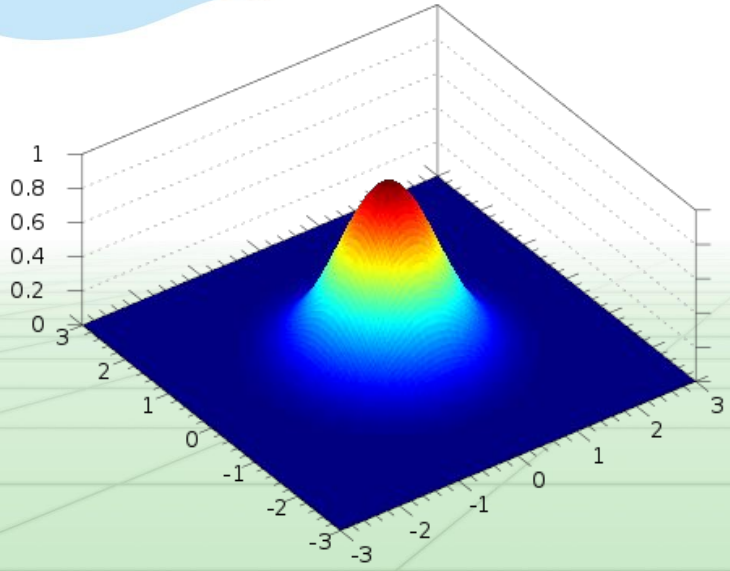
Gaussians

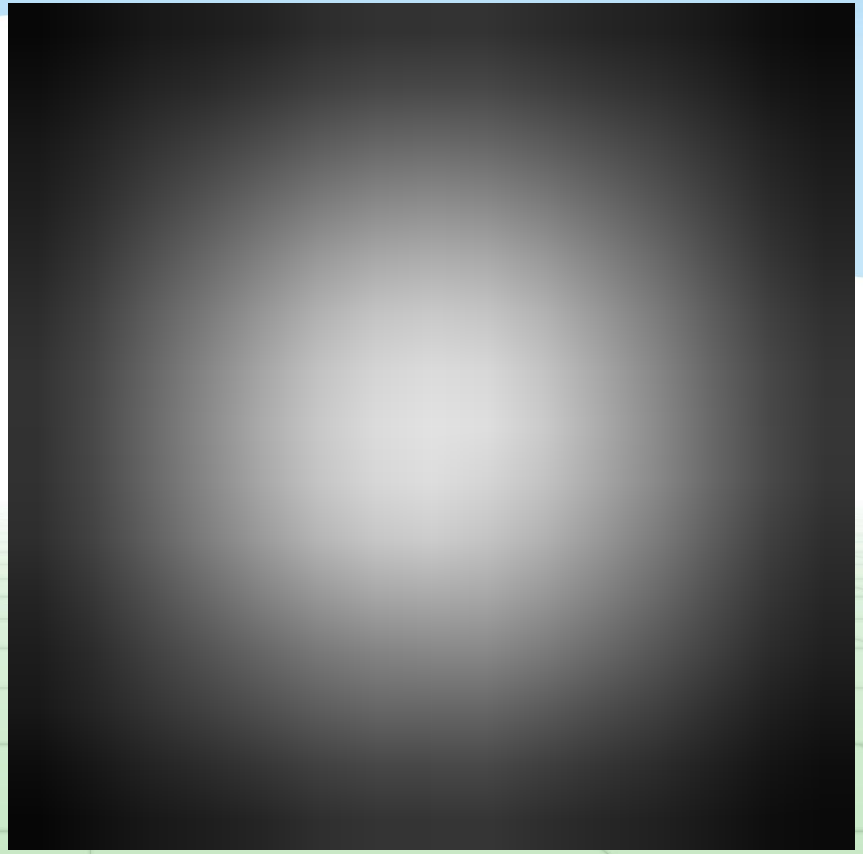
$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$



2d Gaussian

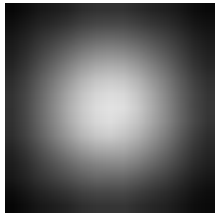
$$g(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$



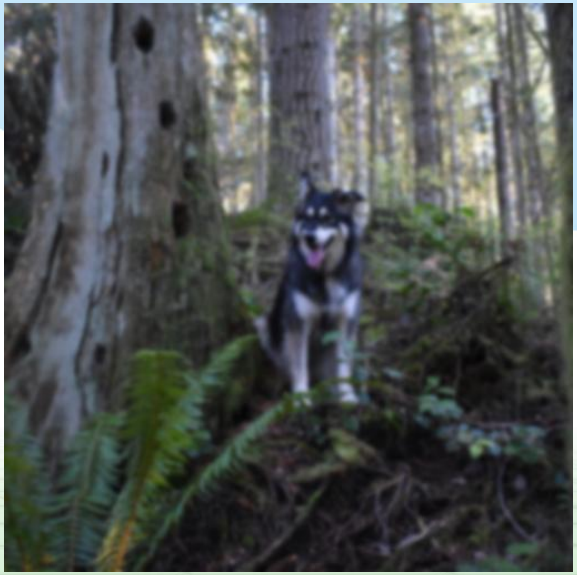


Better smoothing with Gaussians

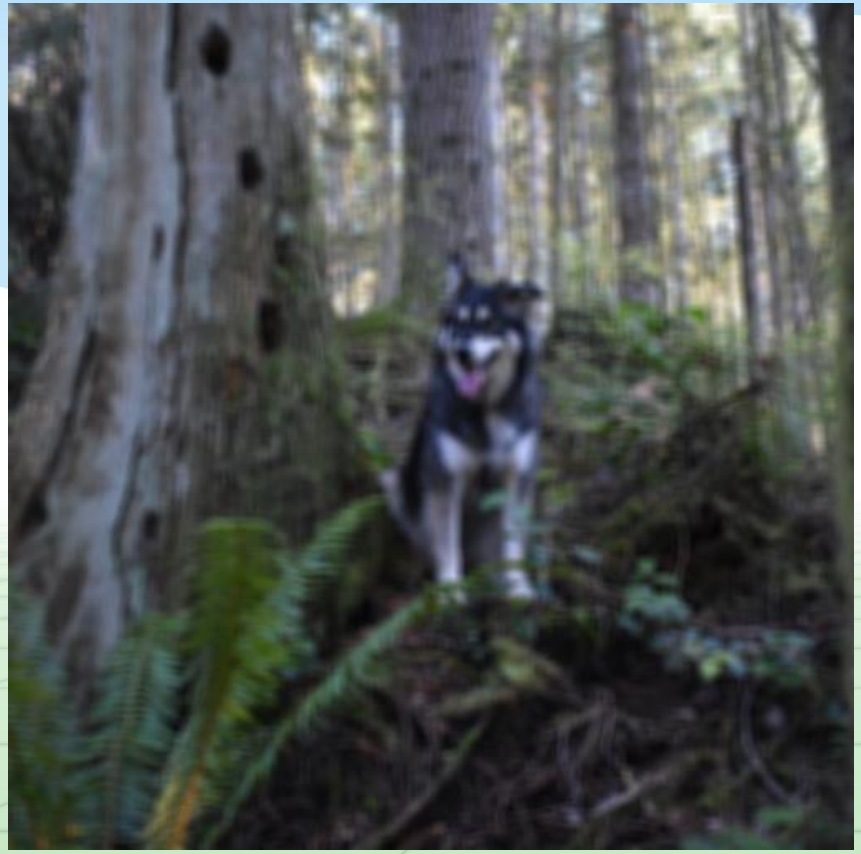
*

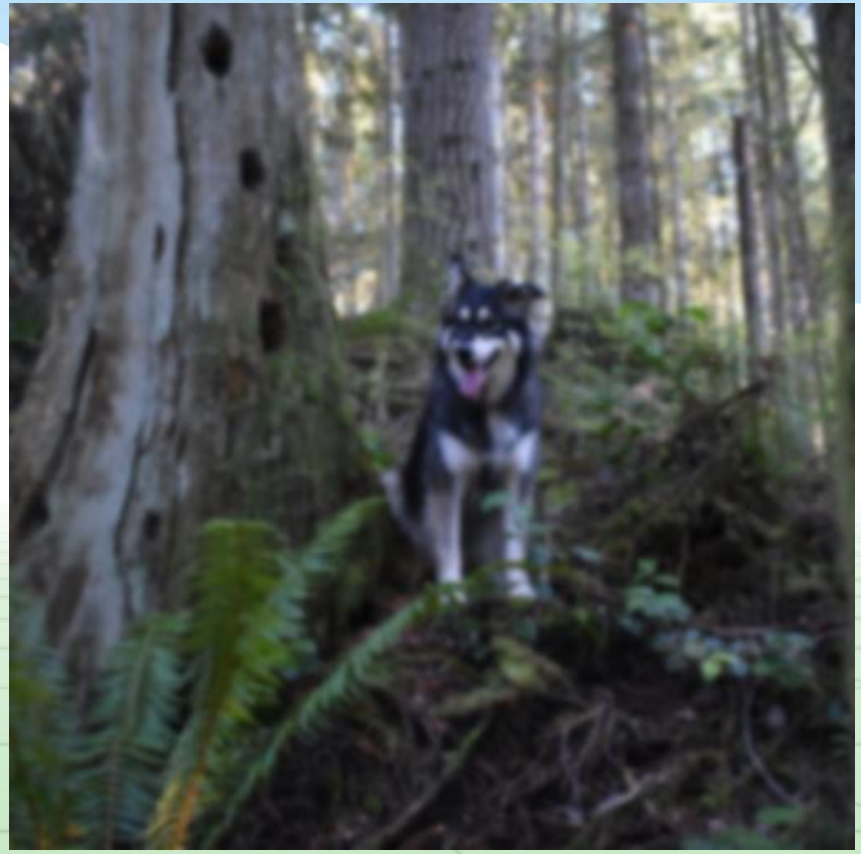


=

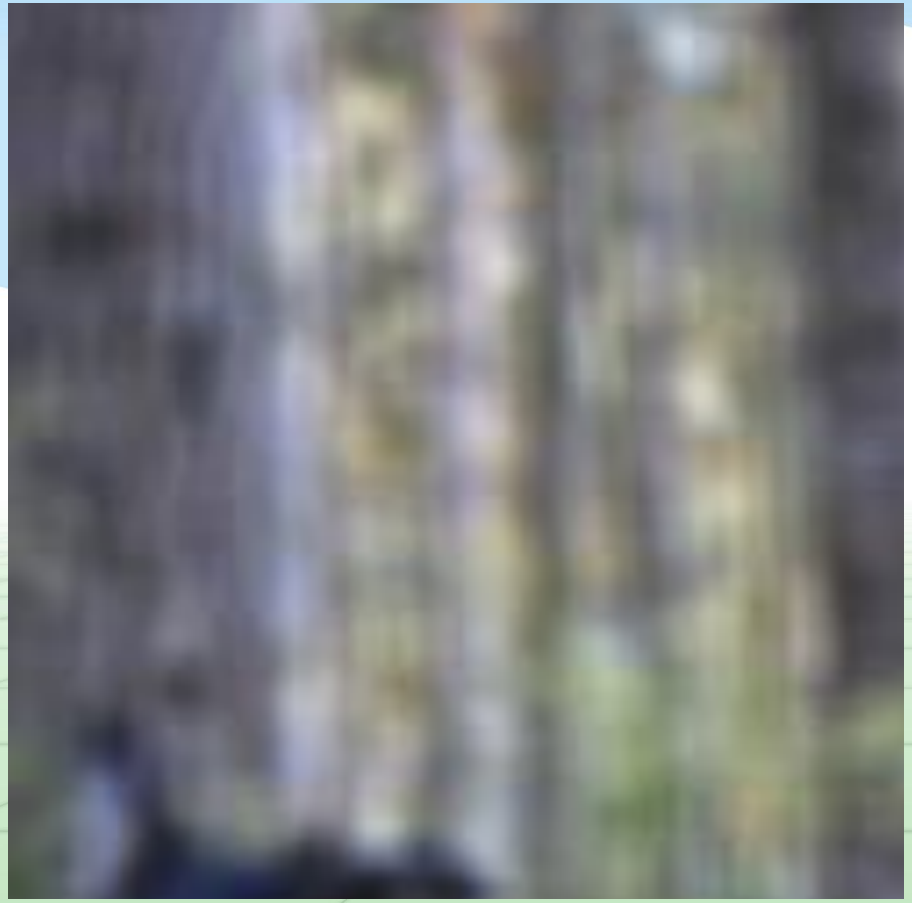


Better smoothing with Gaussians

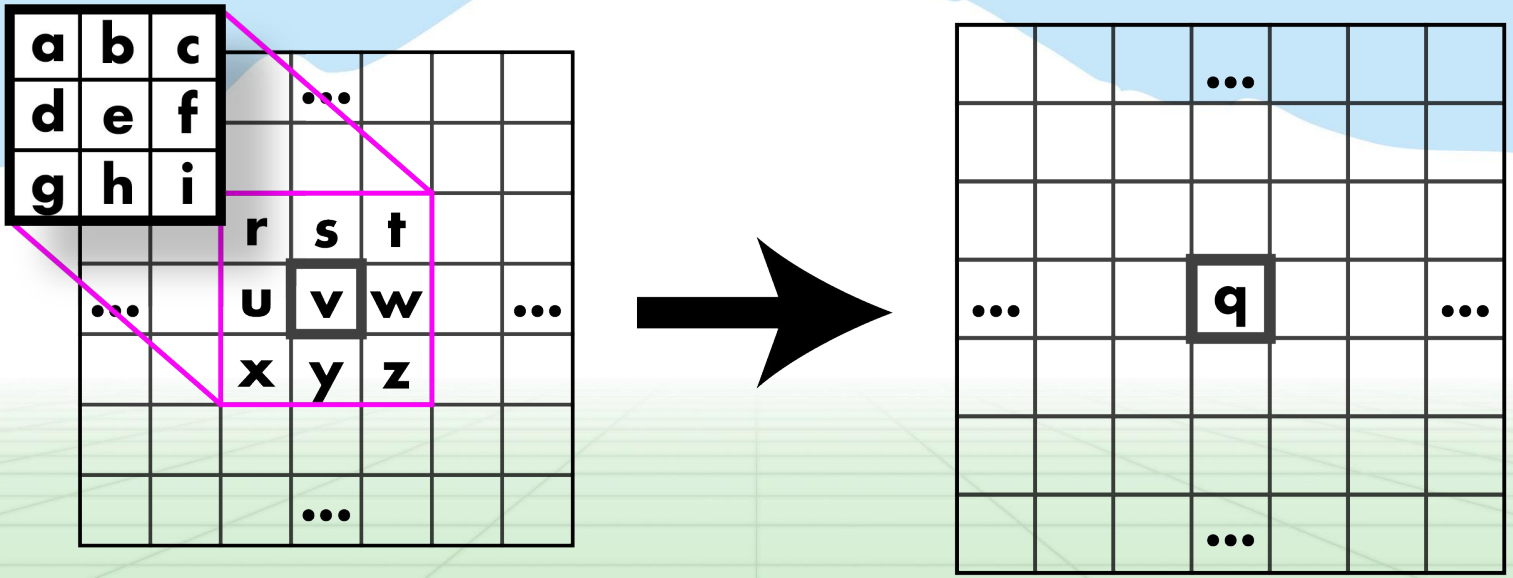




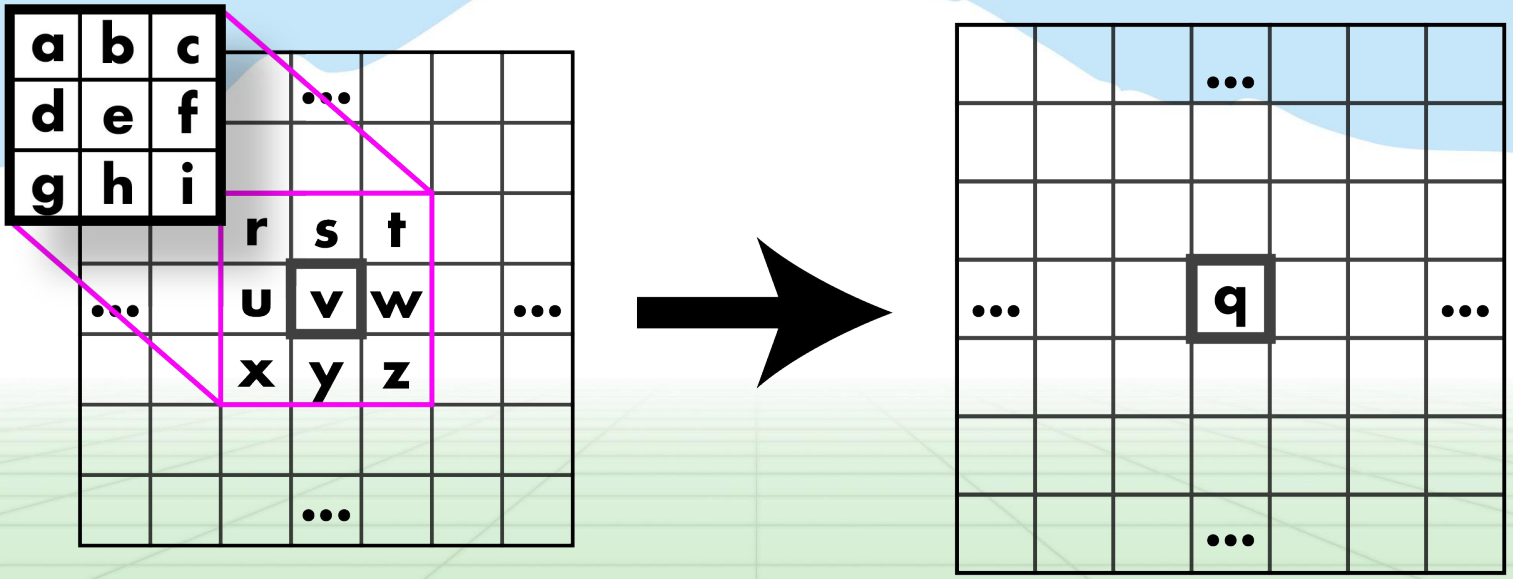
Better smoothing with Gaussians



Wow, so what was that convolution thing??

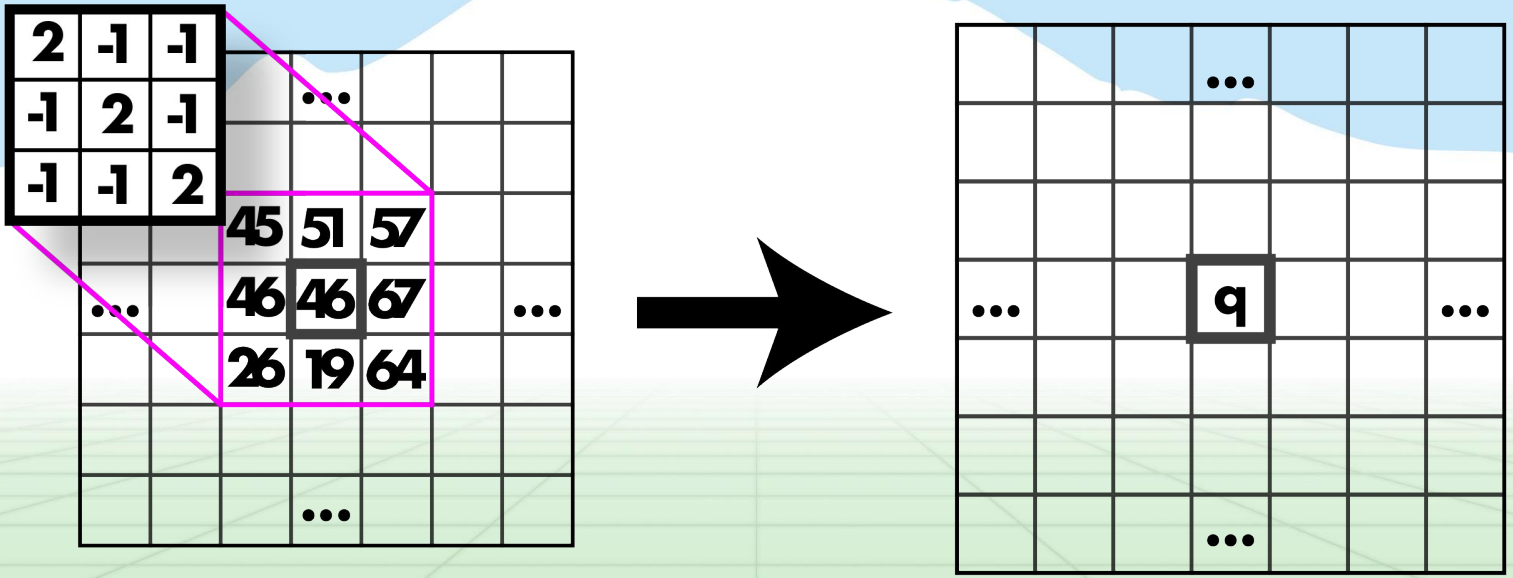


Wow, so what was that convolution thing??

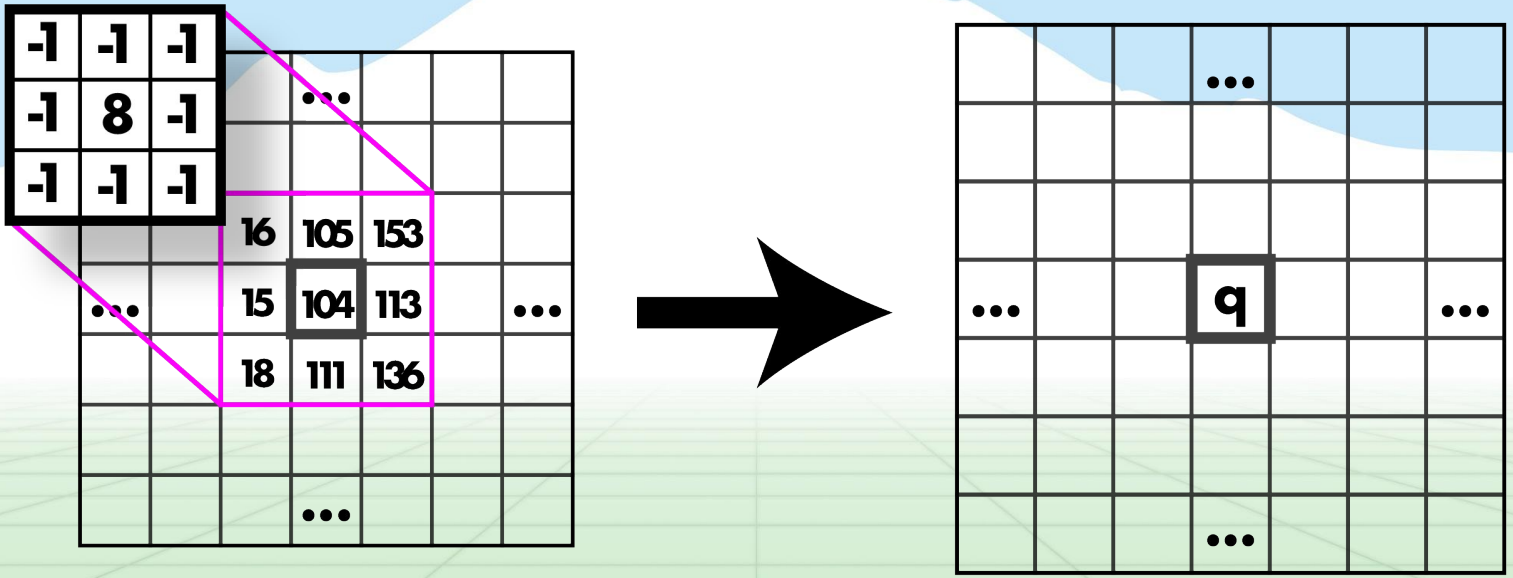


$$q = a \times r + b \times s + c \times t + d \times u + e \times v + f \times w + g \times x + h \times y + i \times z$$

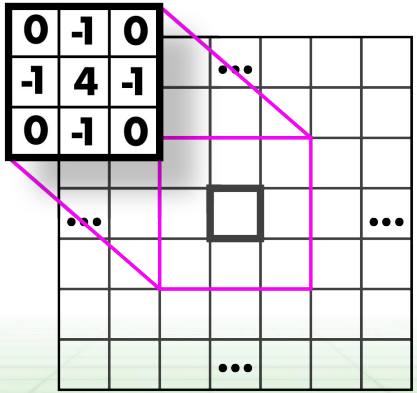
Calculate it, go!



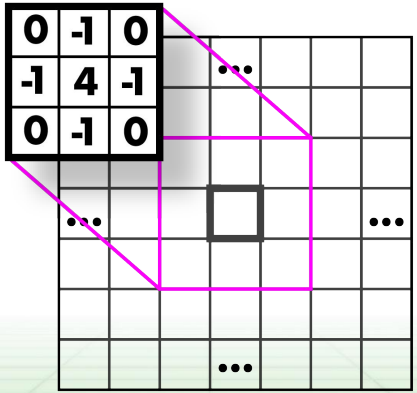
Calculate it, go!



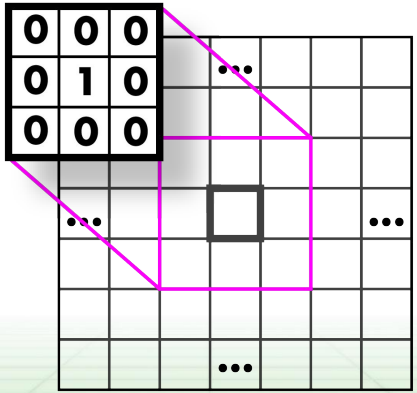
Guess that kernel!



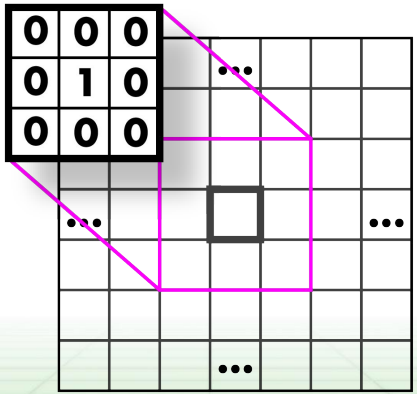
Highpass Kernel: finds edges



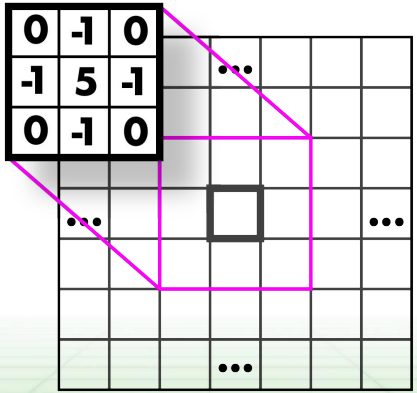
Guess that kernel!



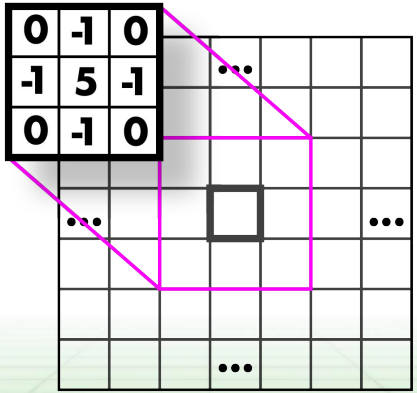
Identity Kernel: Does nothing!



Guess that kernel!

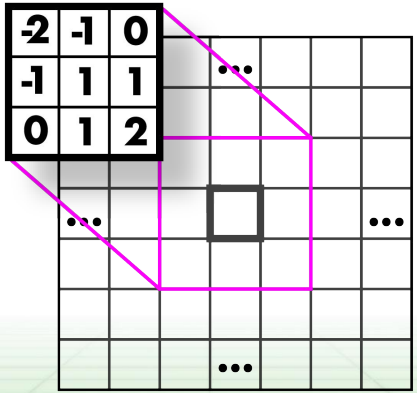


Sharpen Kernel: sharpens!

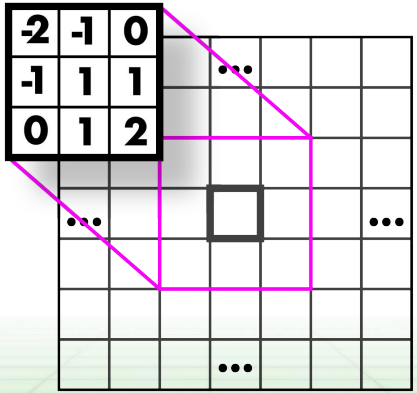


Note: sharpen = highpass + identity!

Guess that kernel!

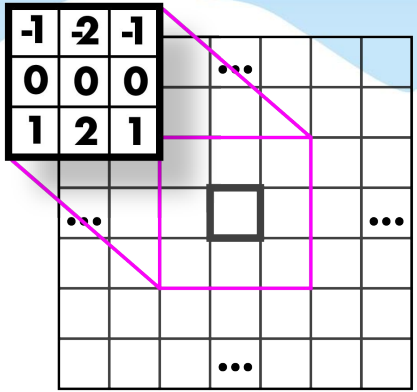


Emboss Kernel: stylin’

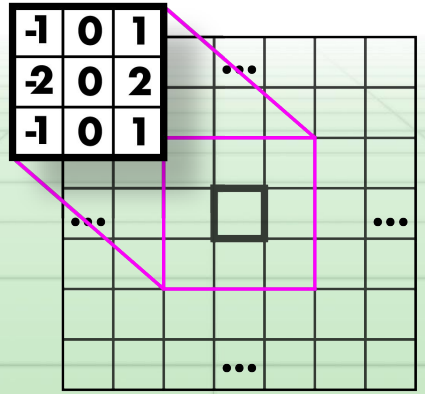


Guess those kernels!

-1	-2	-1
0	0	0
1	2	1

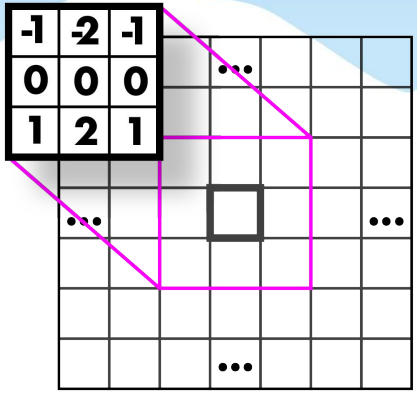


-1	0	1
-2	0	2
-1	0	1

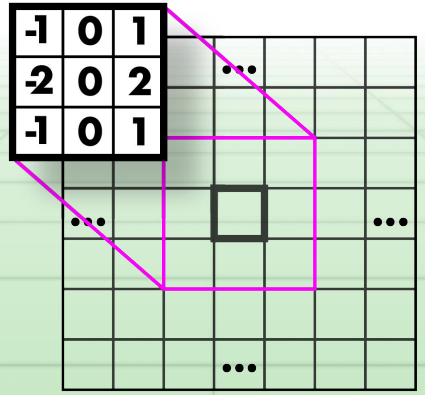


Sobel Kernels: edges and...

-1	-2	-1
0	0	0
1	2	1

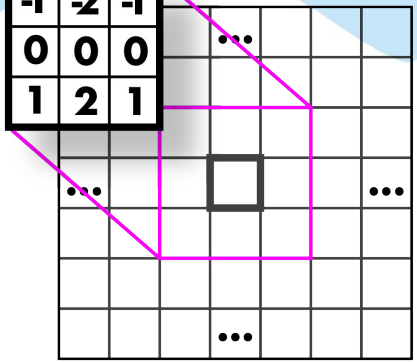


-1	0	1
-2	0	2
-1	0	1

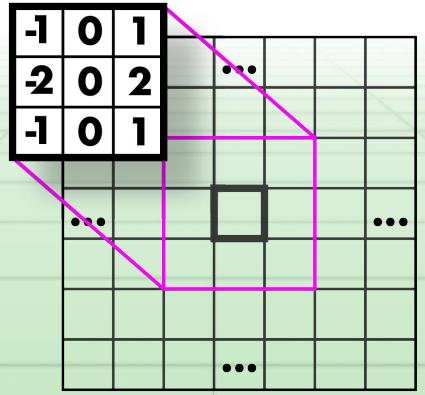


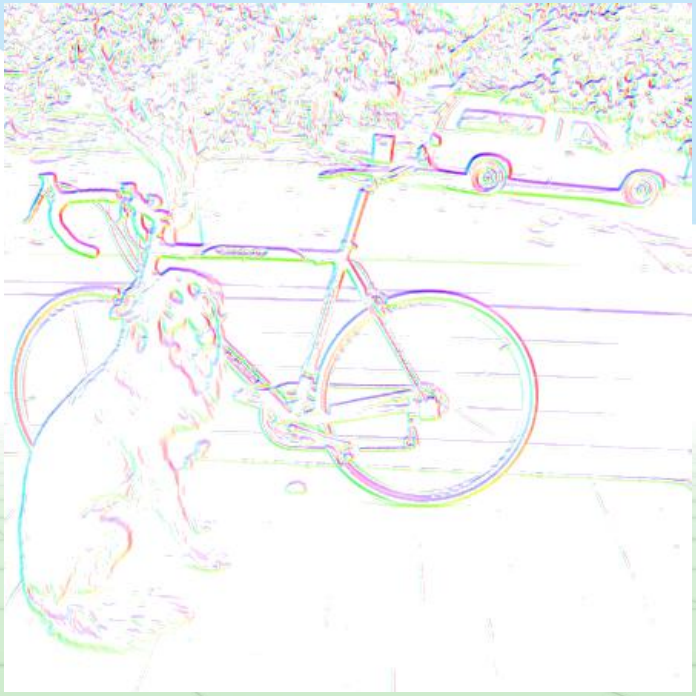
Sobel Kernels: edges and gradient!

-1	-2	-1
0	0	0
1	2	1

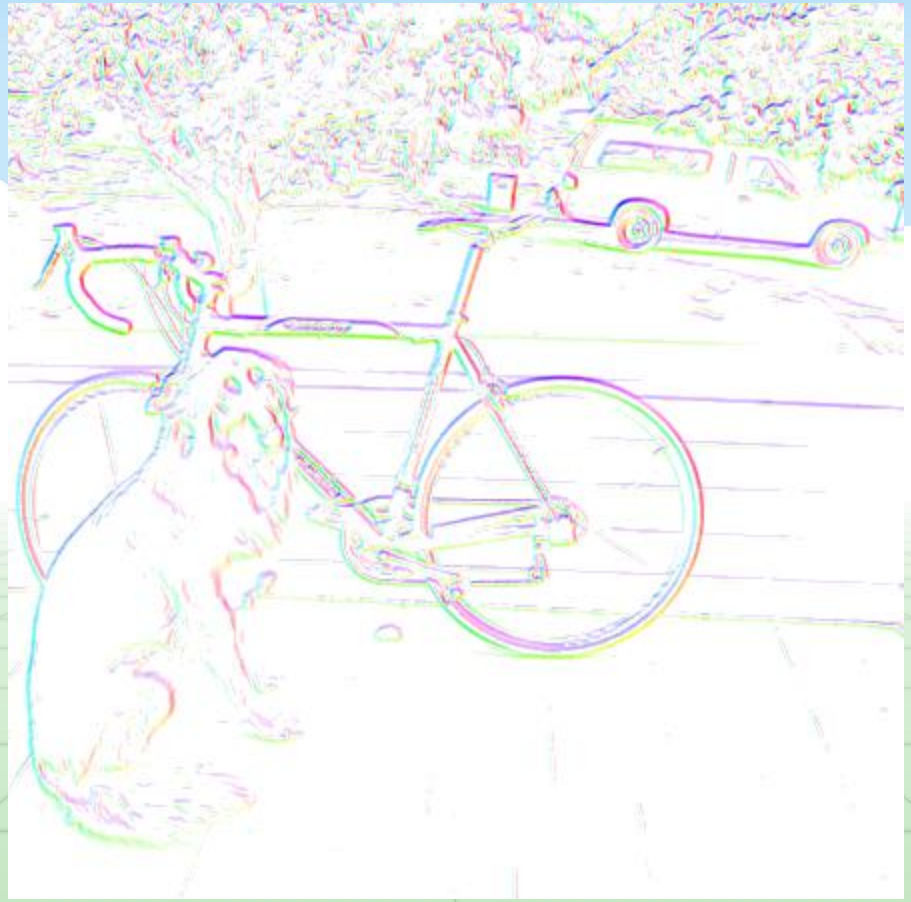


-1	0	1
-2	0	2
-1	0	1





Sobel Kernels: edges and gradient!



And so much more!!