

# Computer Vision

ECE/CSE 576

Filters

Linda Shapiro

Professor of Computer Science & Engineering  
Professor of Electrical & Computer Engineering

Let's do something interesting already!!

---

# Want to make image smaller



---

448x448 -> 64x64



---

448x448 -> 64x64



---

448x448 -> 64x64



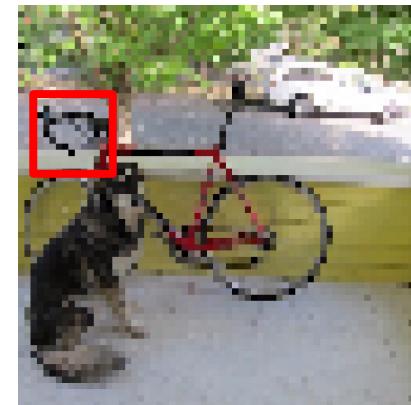
---

448x448 -> 64x64



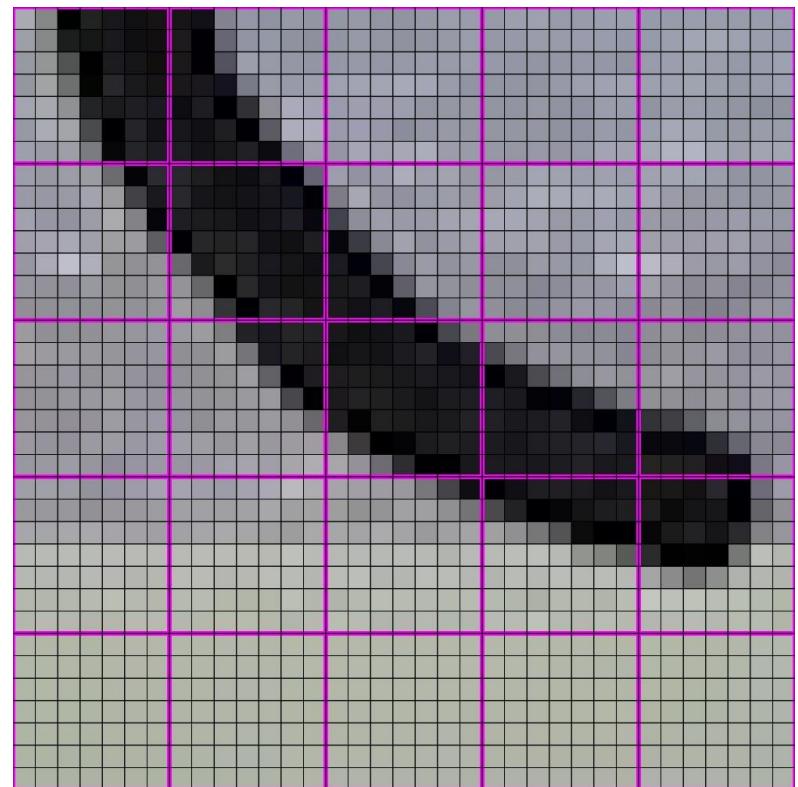
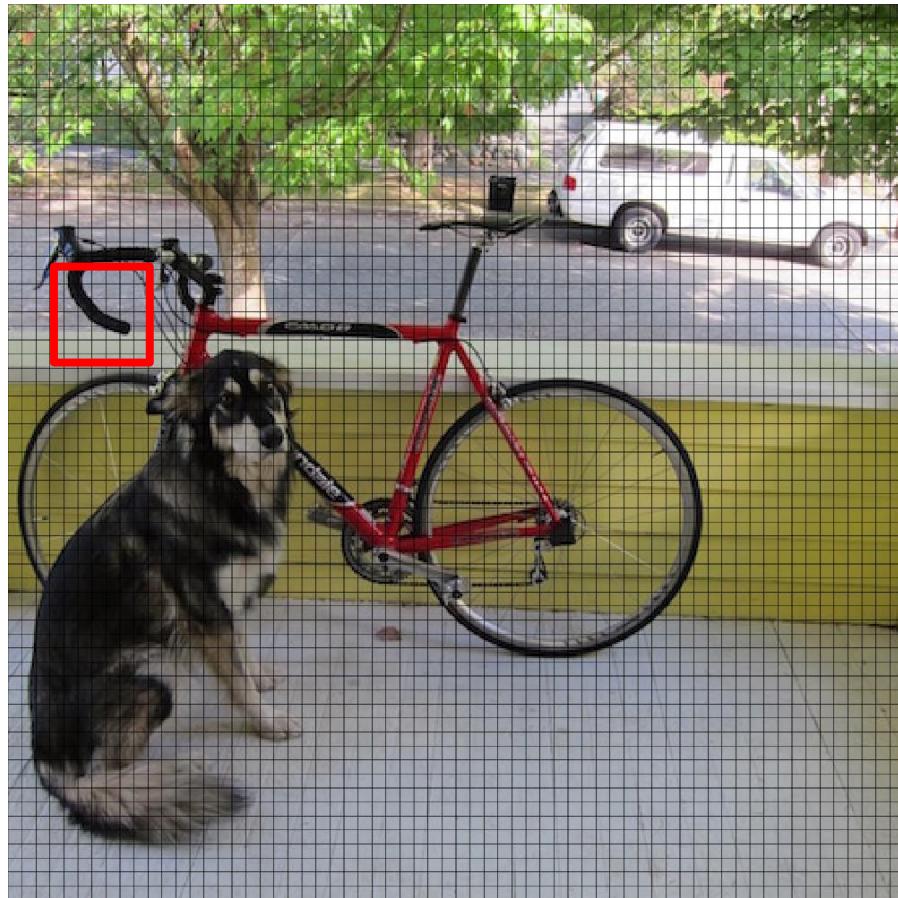
---

448x448 -> 64x64



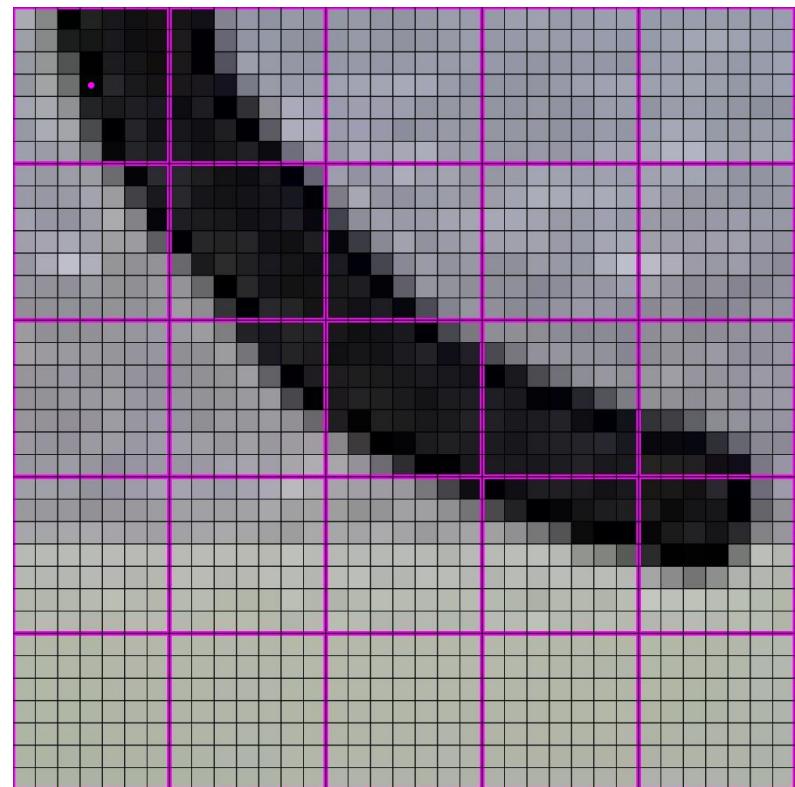
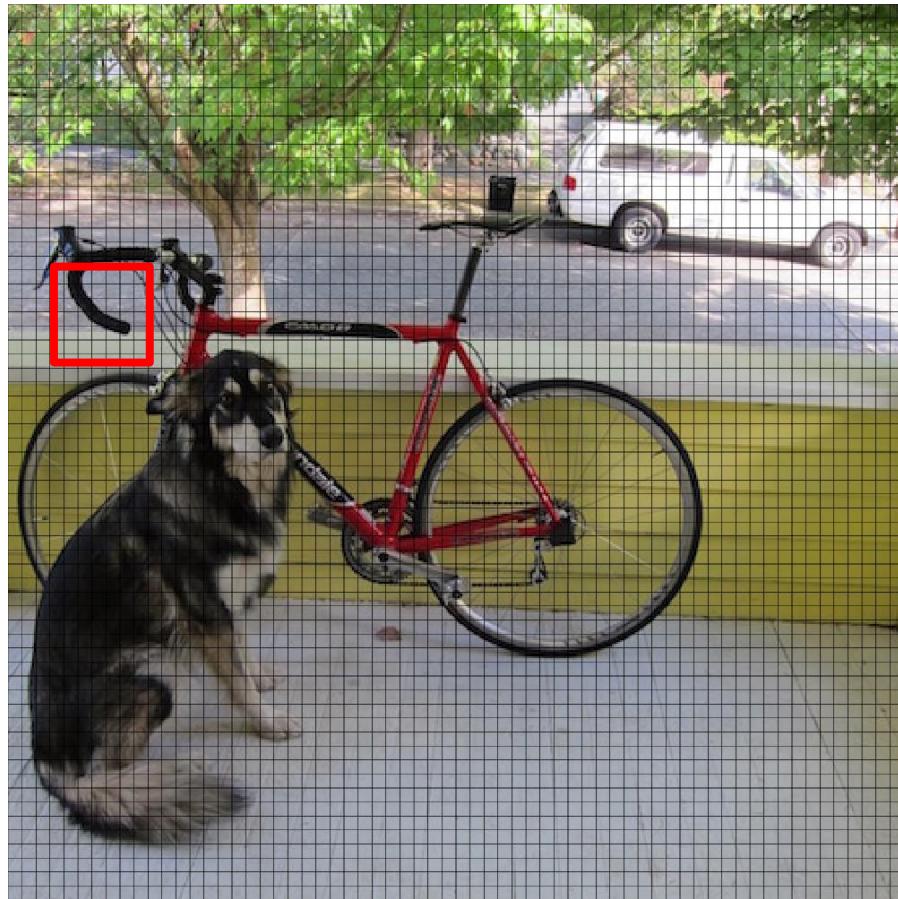
---

448x448 -> 64x64



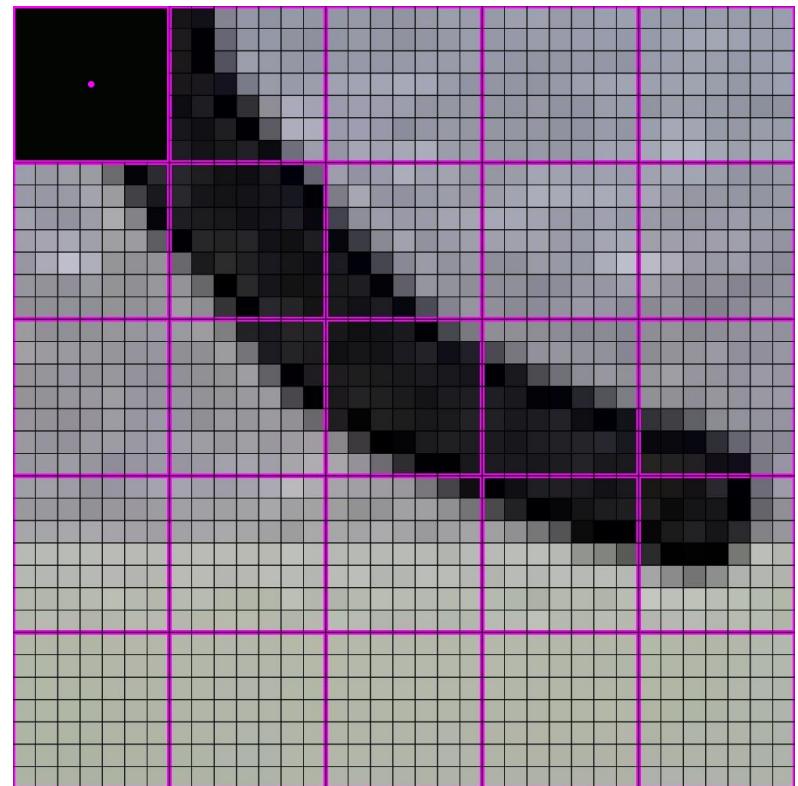
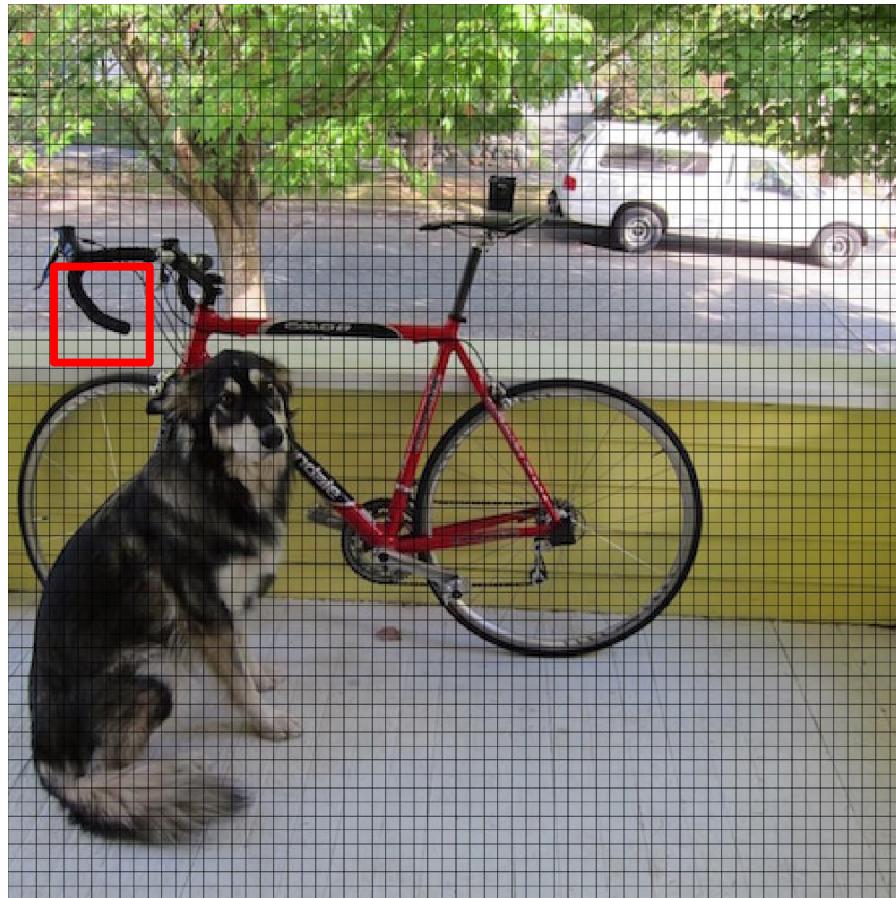
---

448x448 -> 64x64



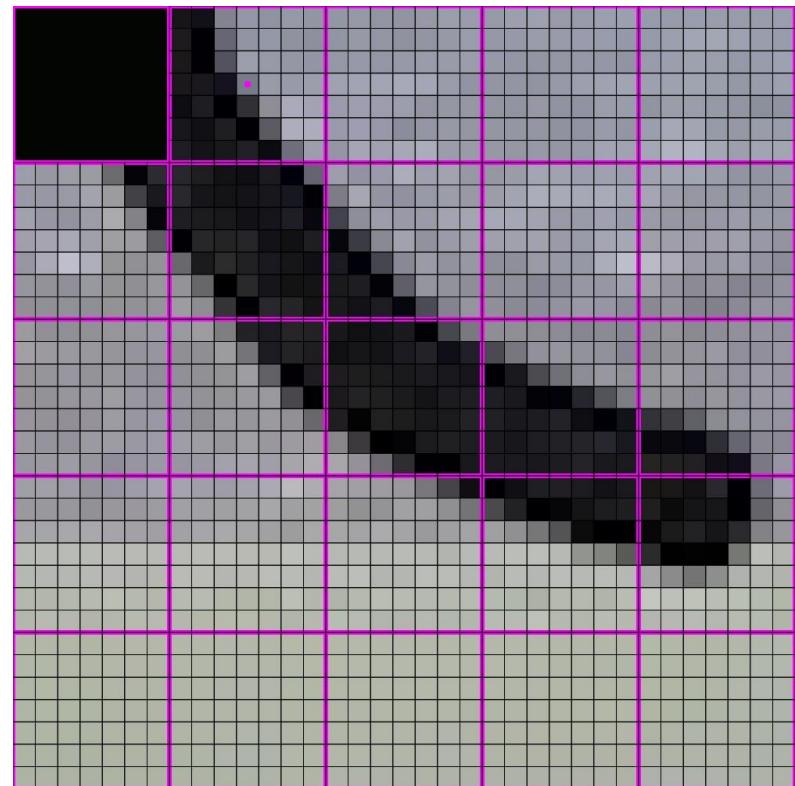
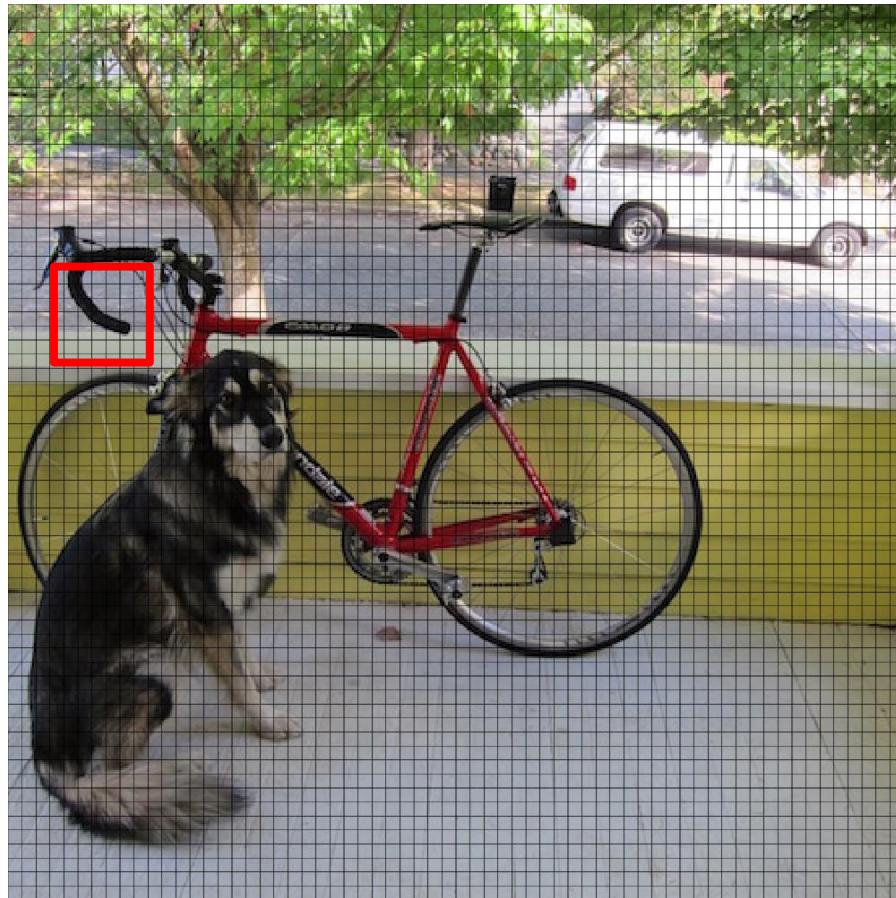
---

448x448 -> 64x64



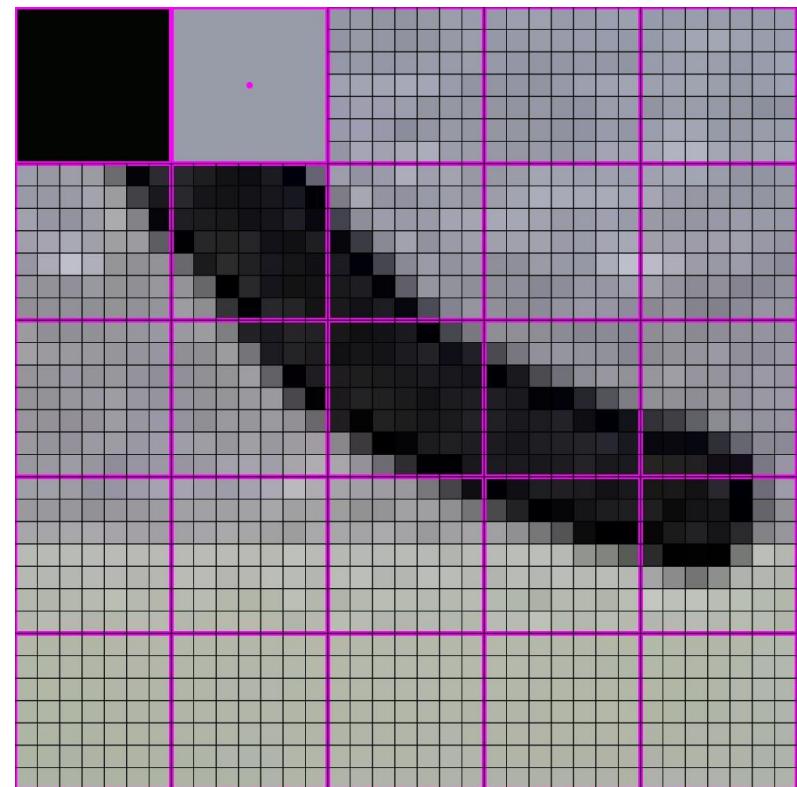
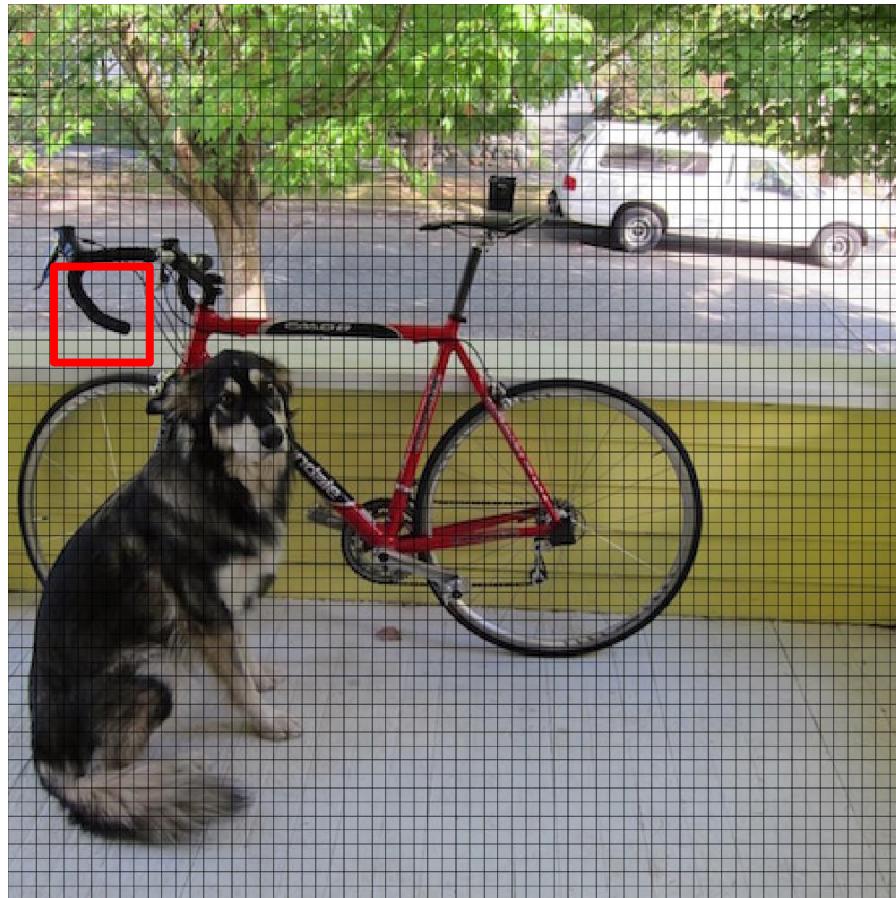
---

448x448 -> 64x64



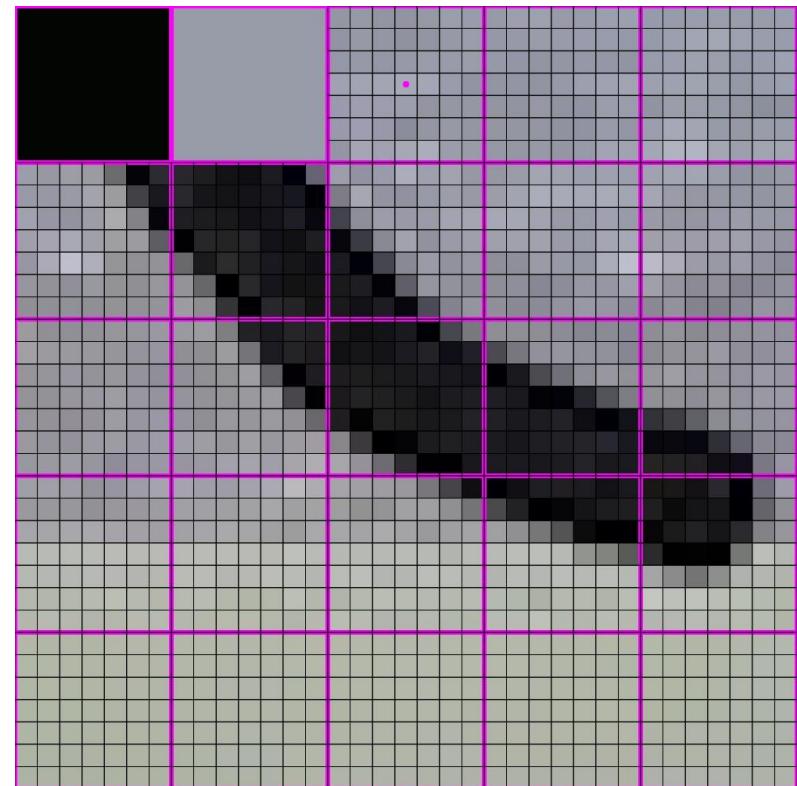
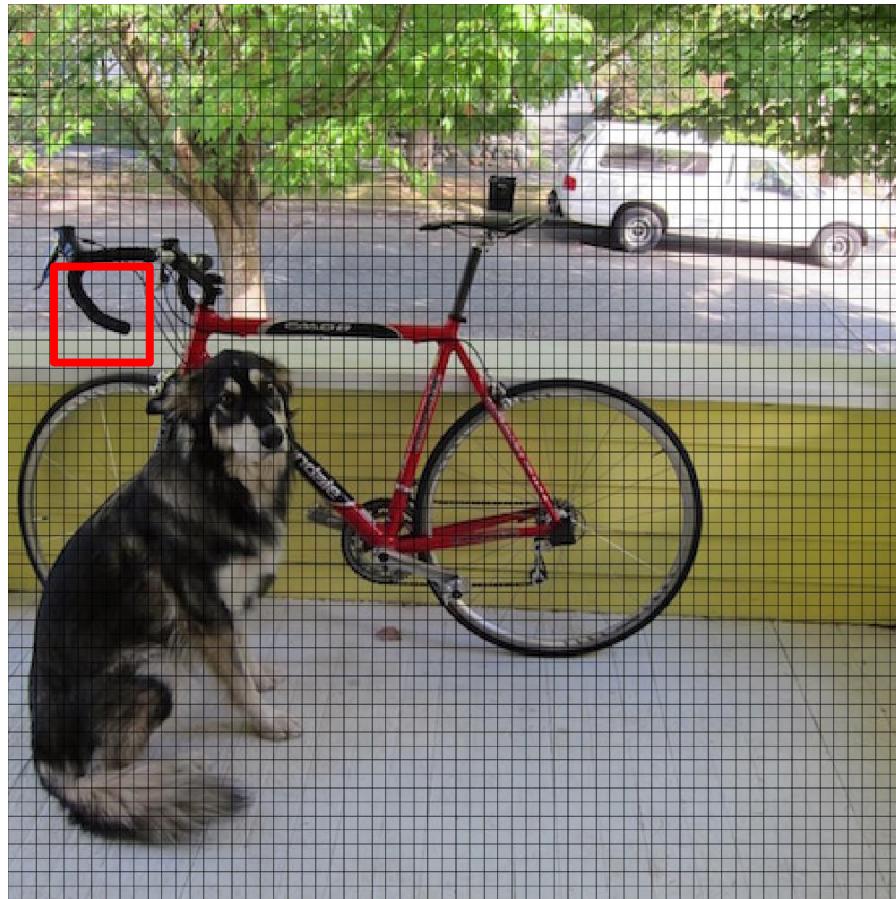
---

448x448 -> 64x64



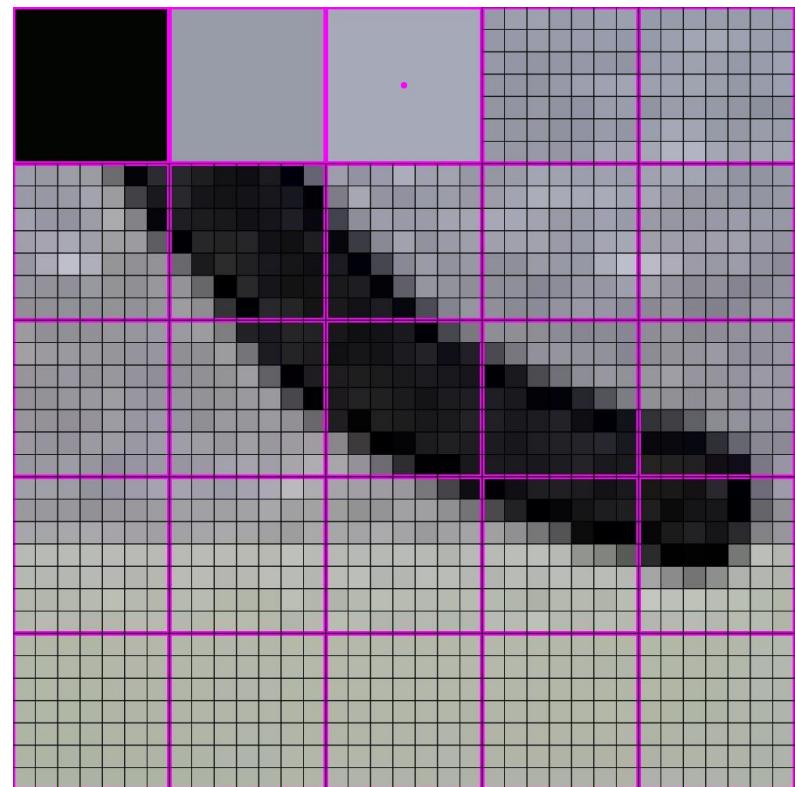
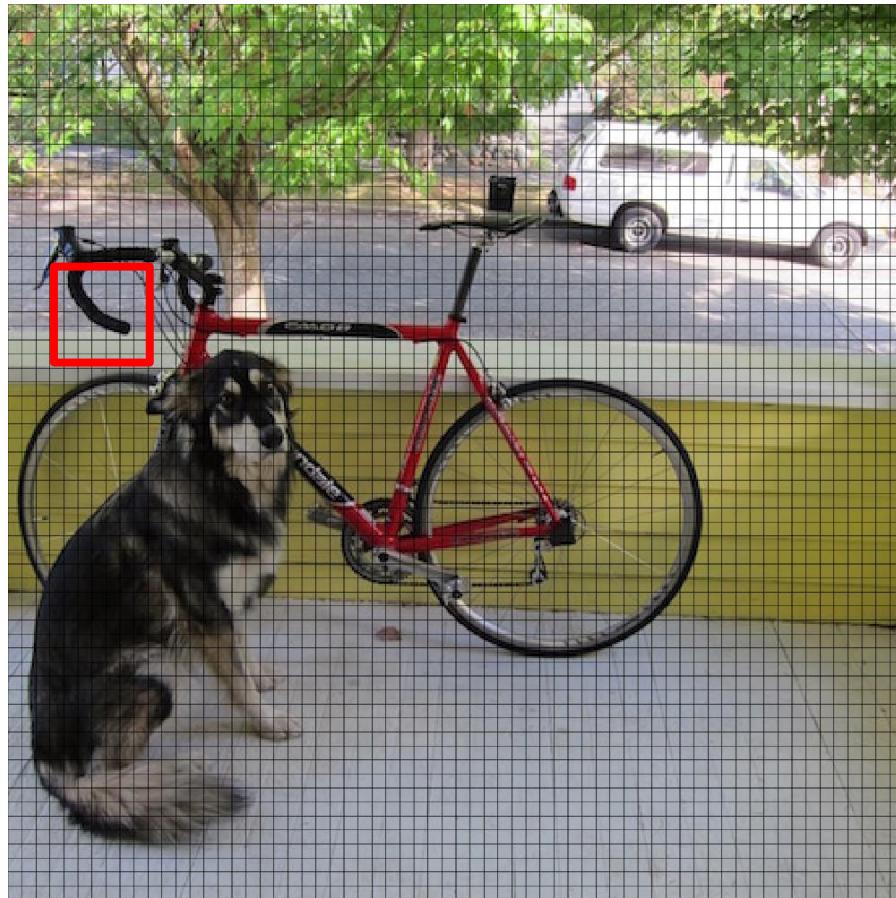
---

448x448 -> 64x64



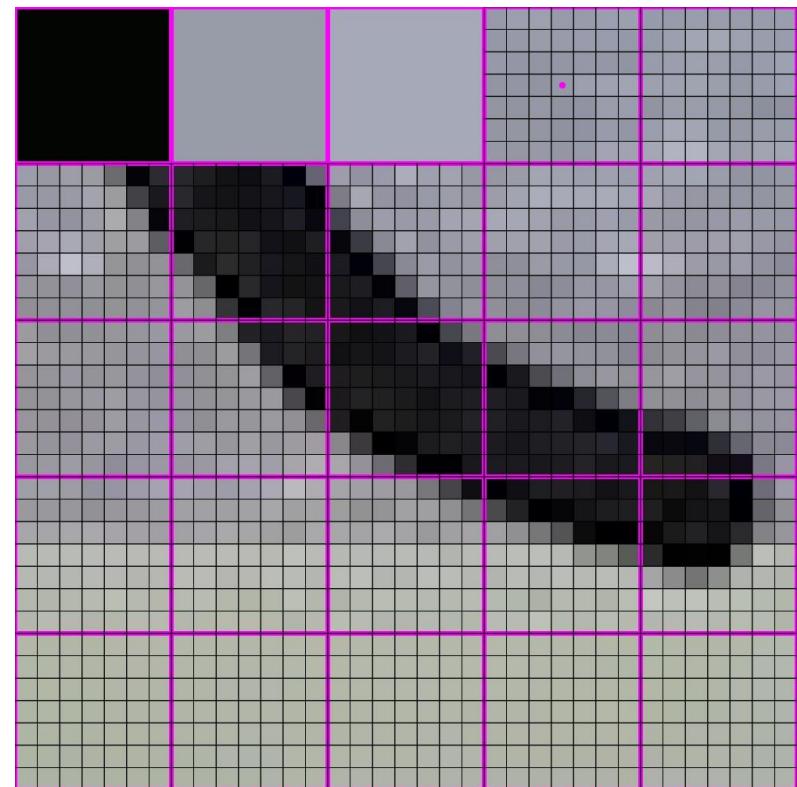
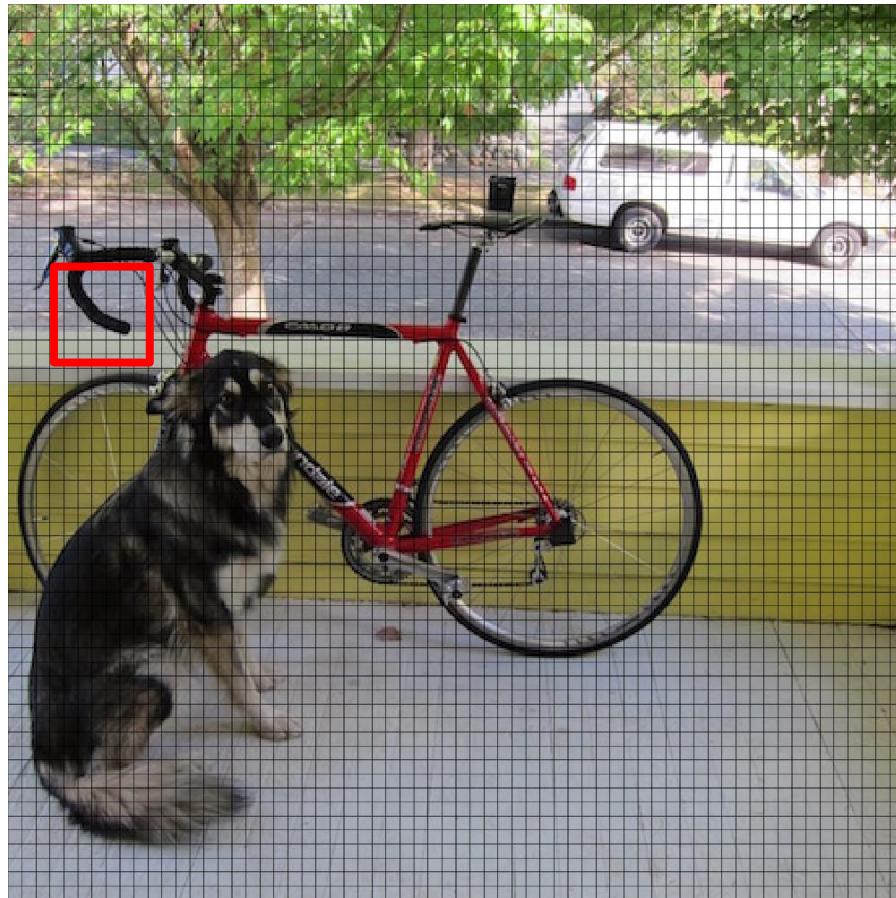
---

448x448 -> 64x64



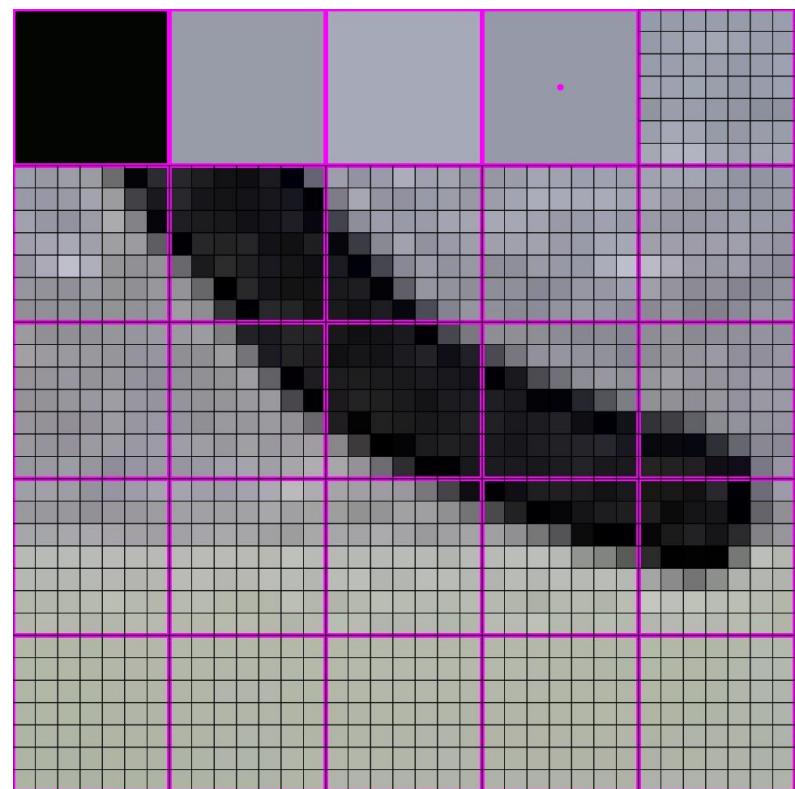
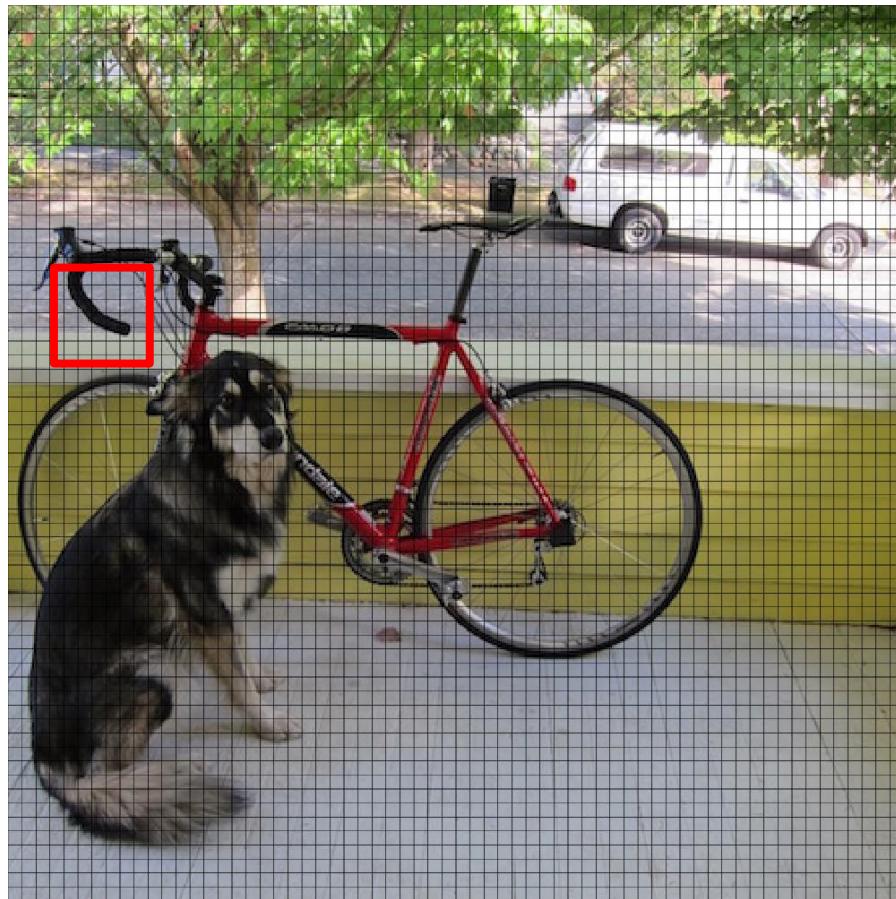
---

448x448 -> 64x64



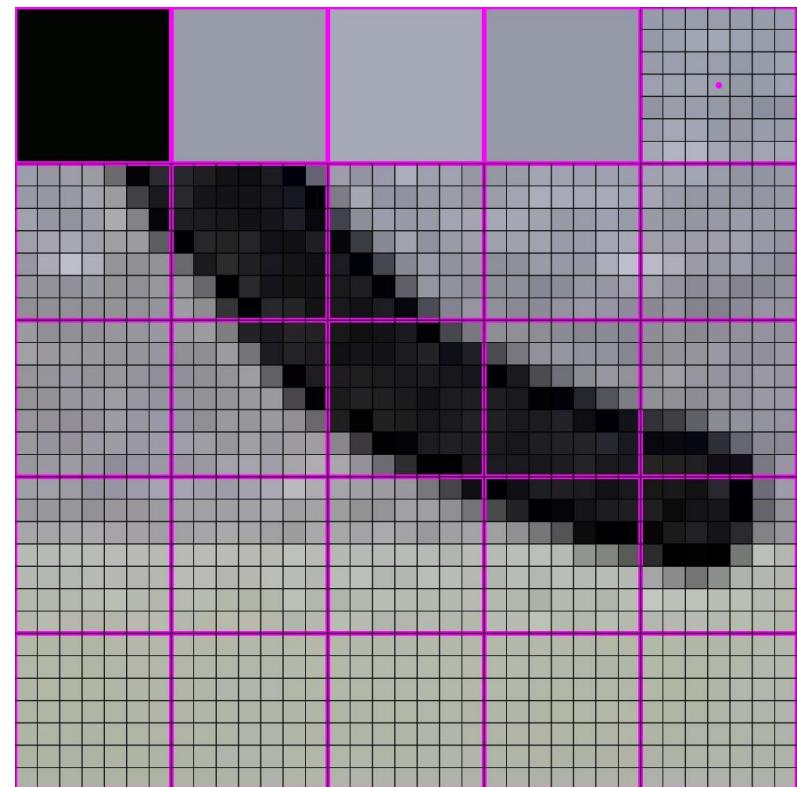
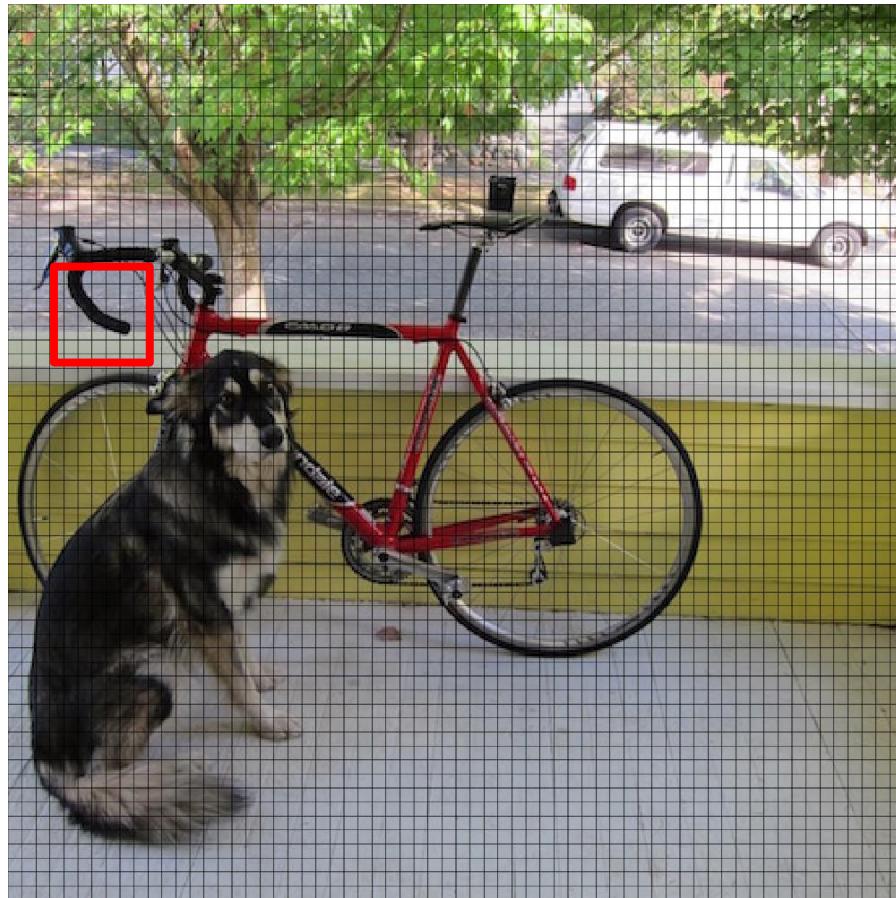
---

448x448 -> 64x64



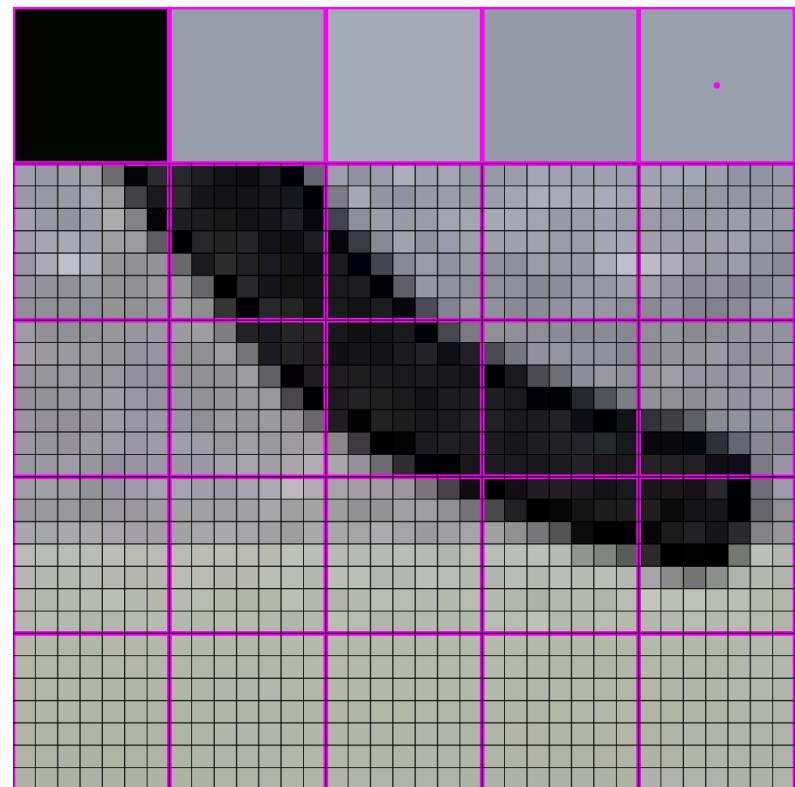
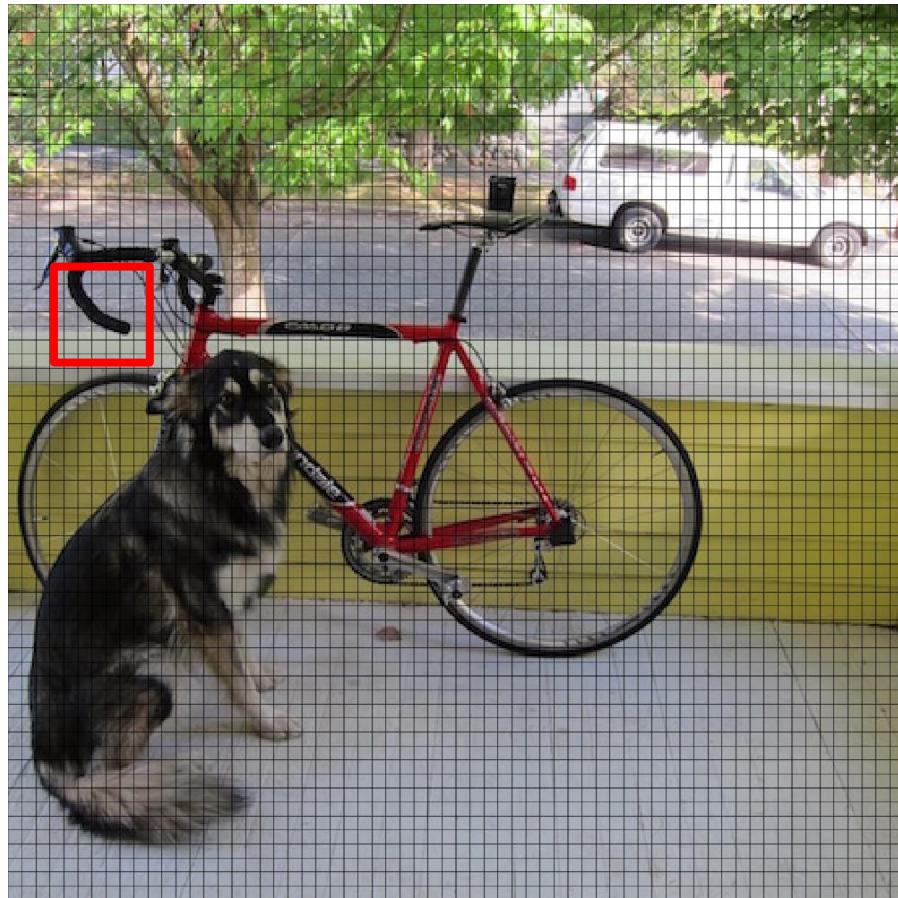
---

448x448 -> 64x64



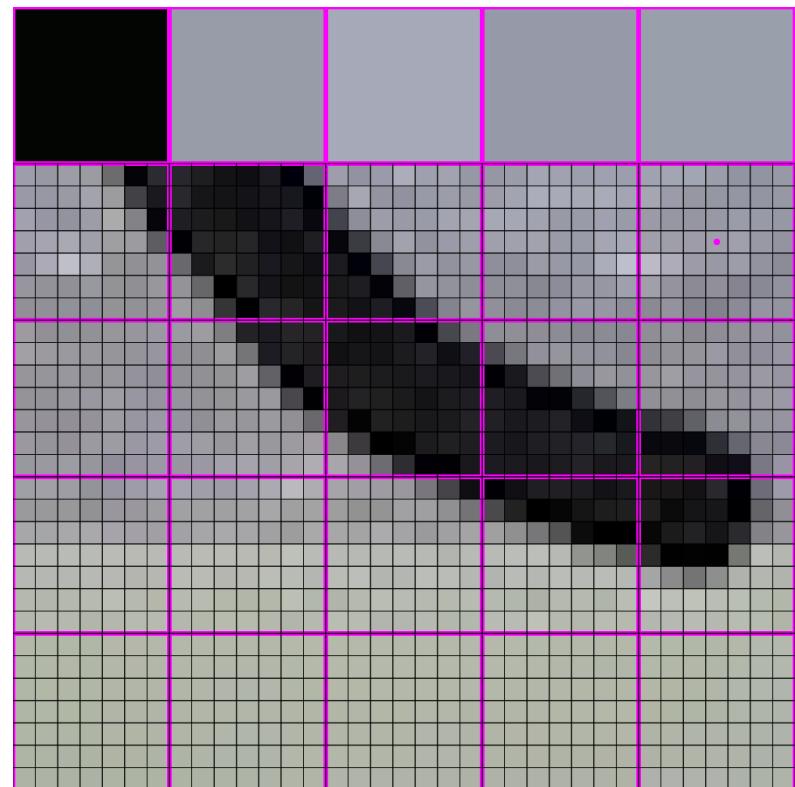
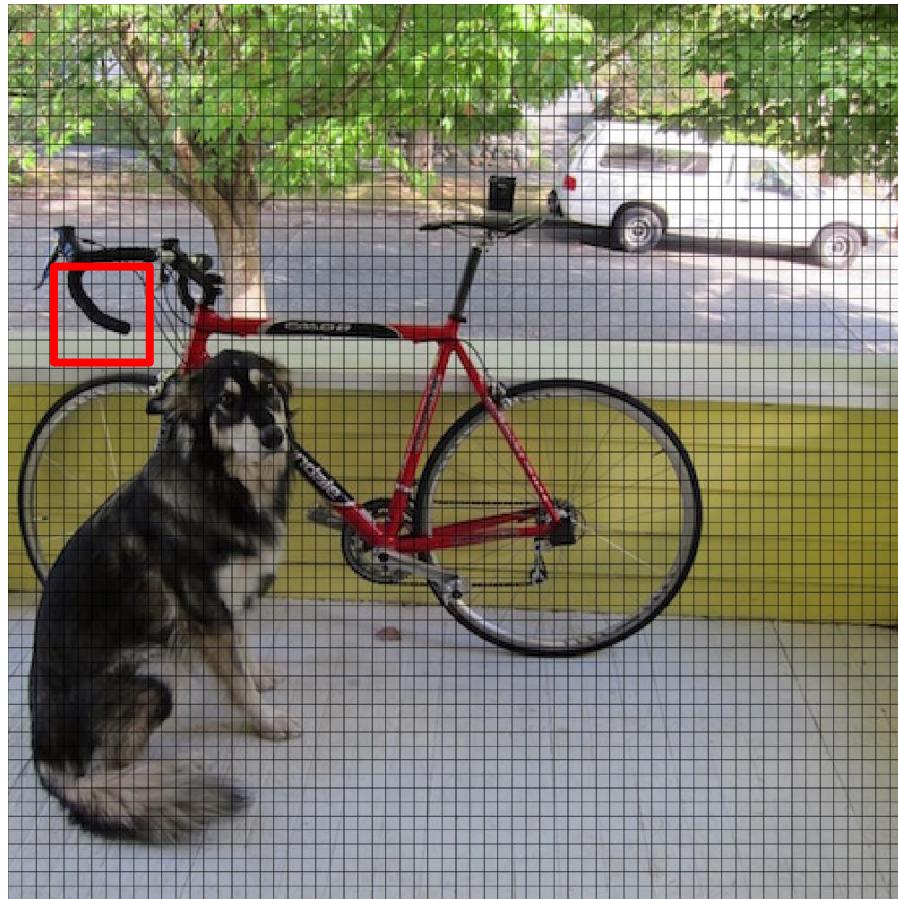
---

448x448 -> 64x64



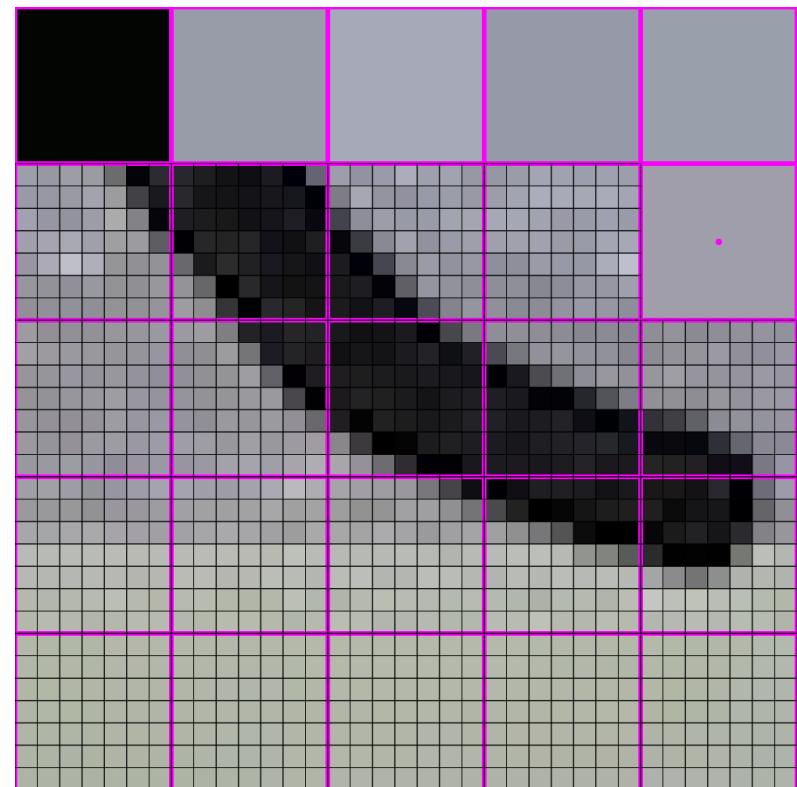
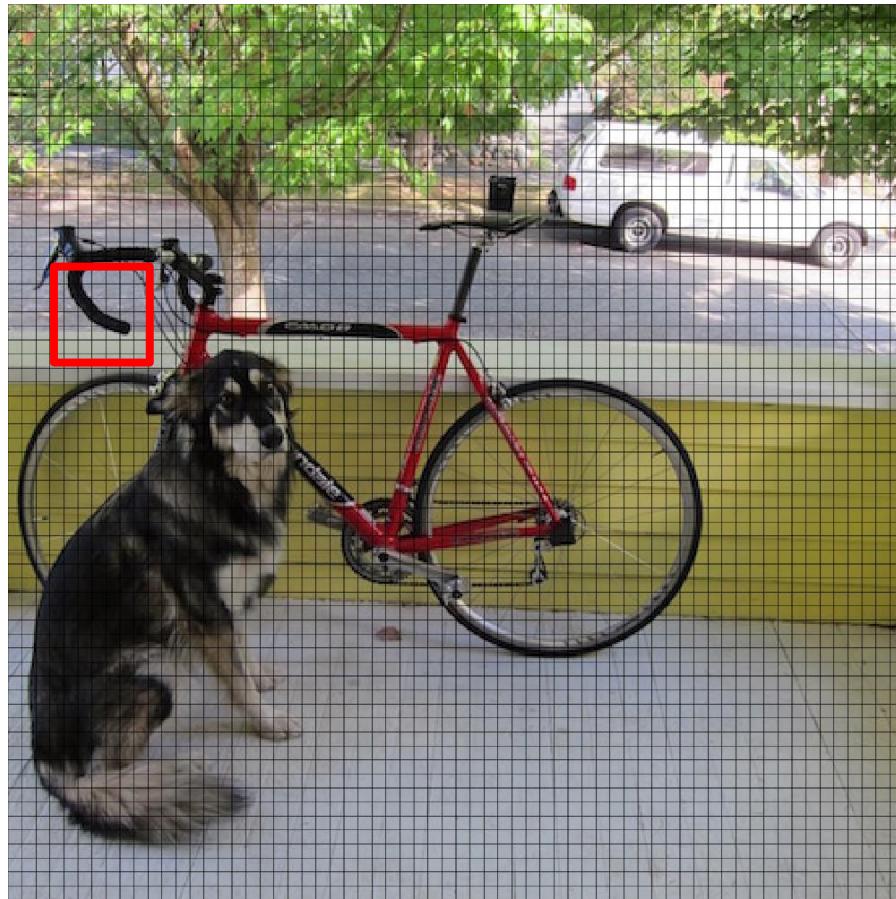
---

448x448 -> 64x64



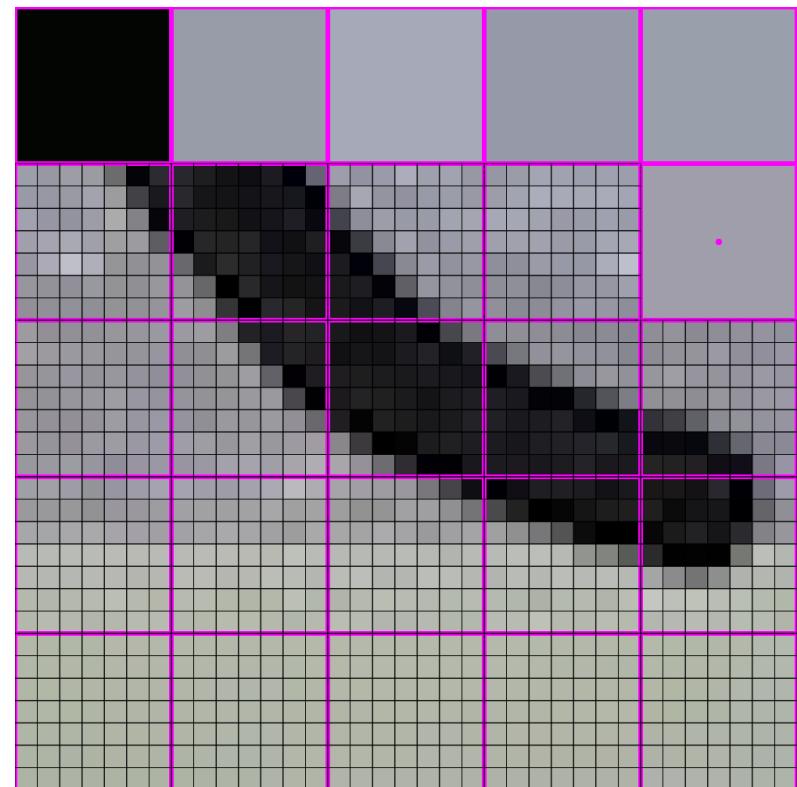
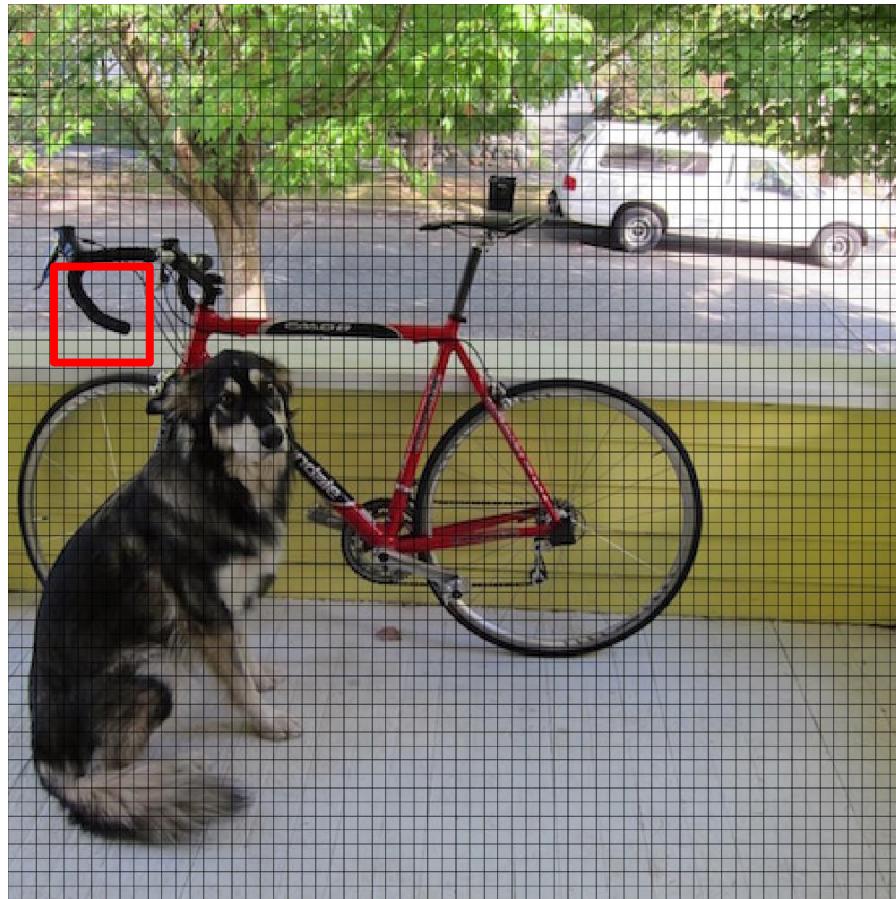
---

448x448 -> 64x64



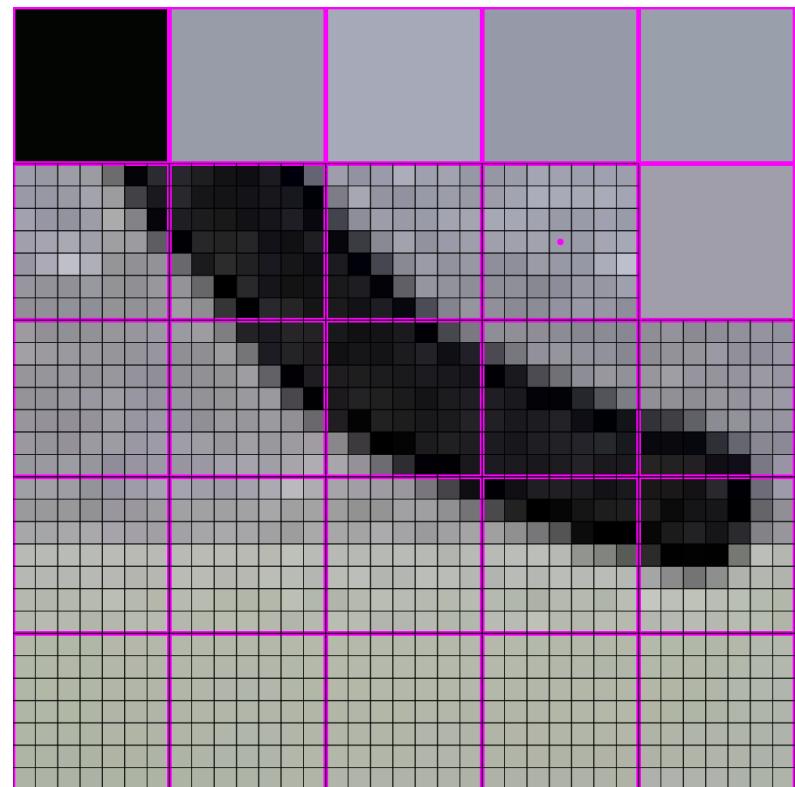
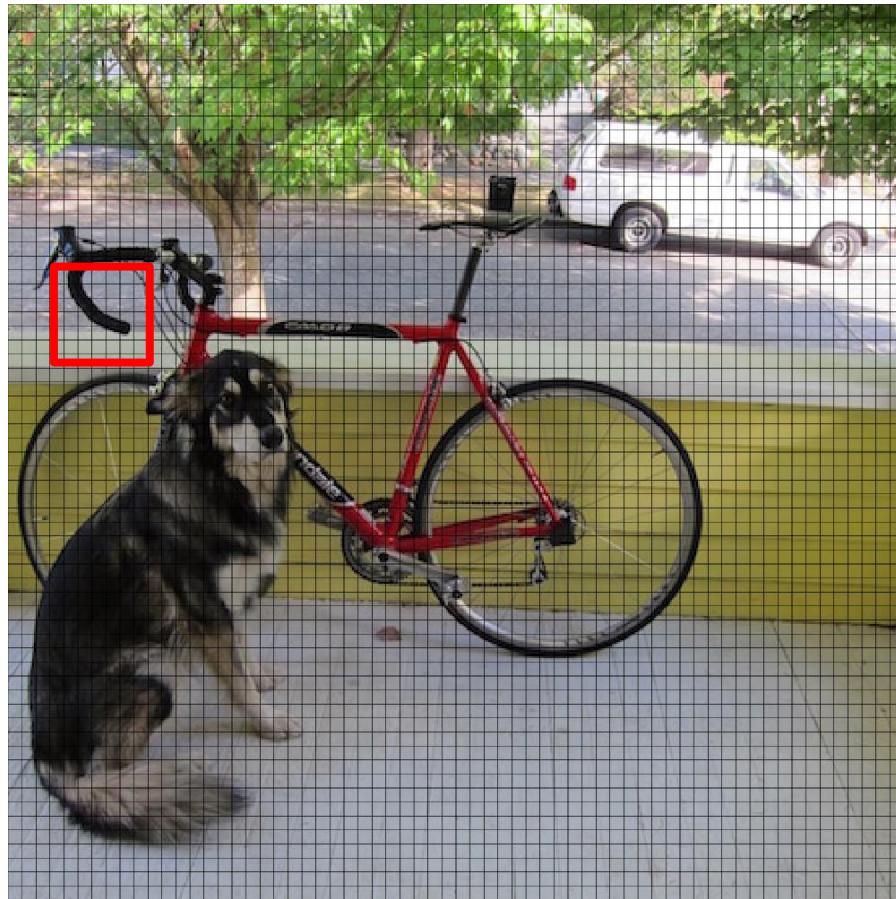
---

448x448 -> 64x64



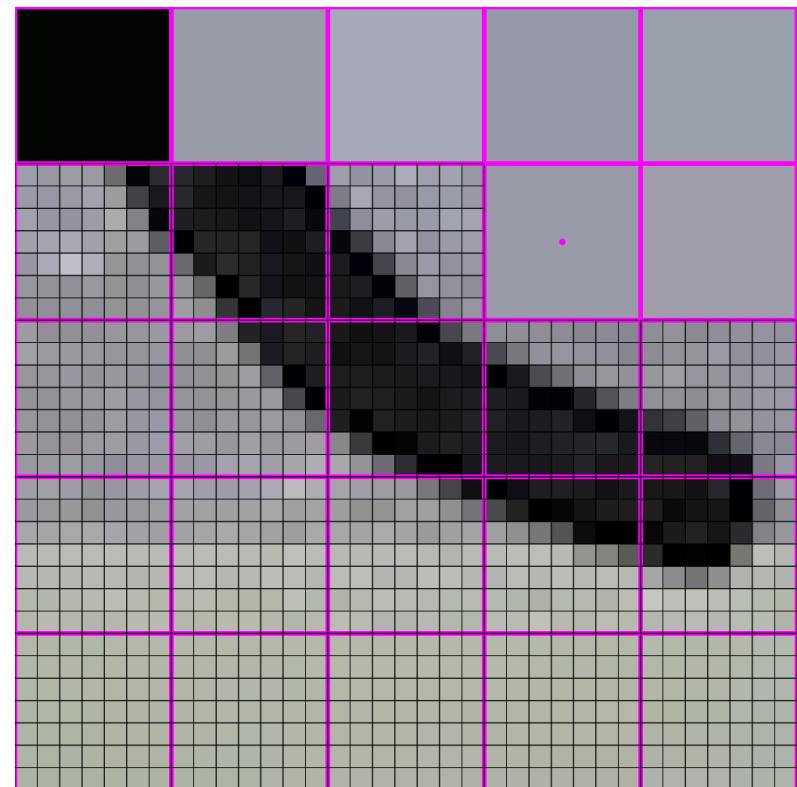
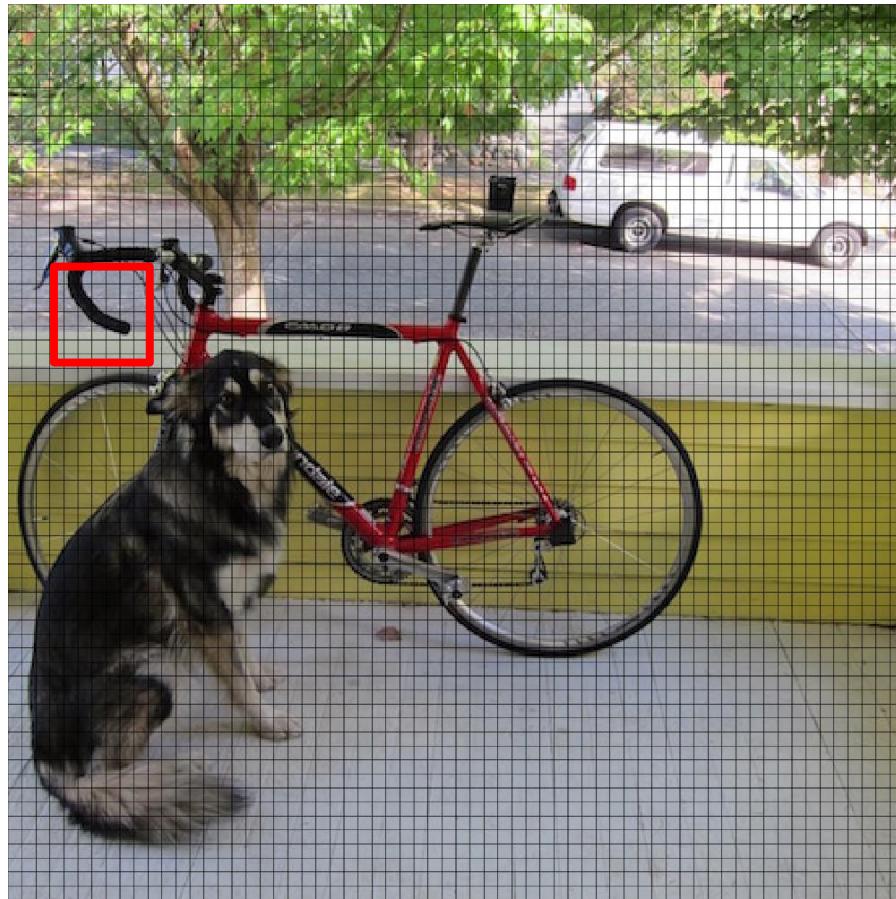
---

448x448 -> 64x64



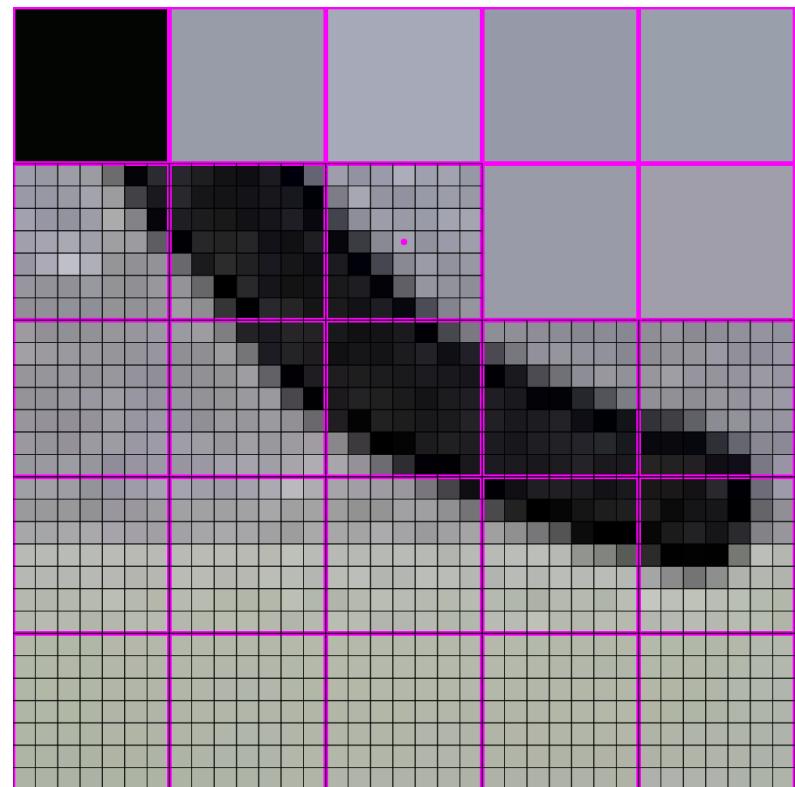
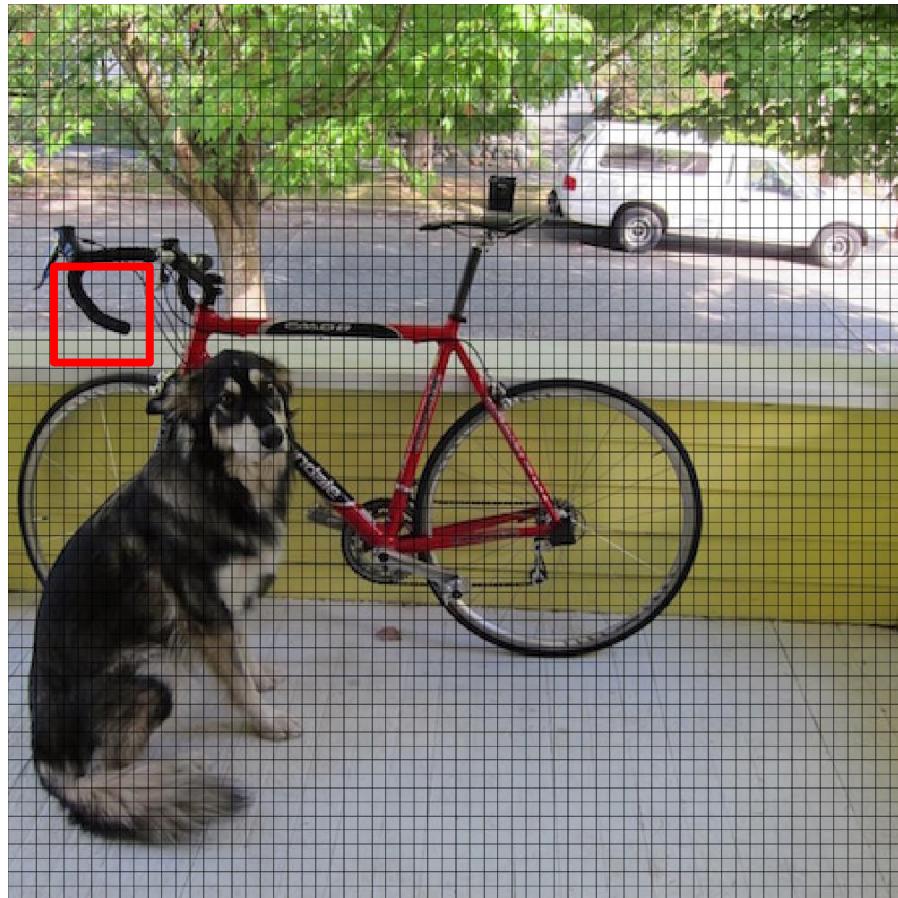
---

448x448 -> 64x64



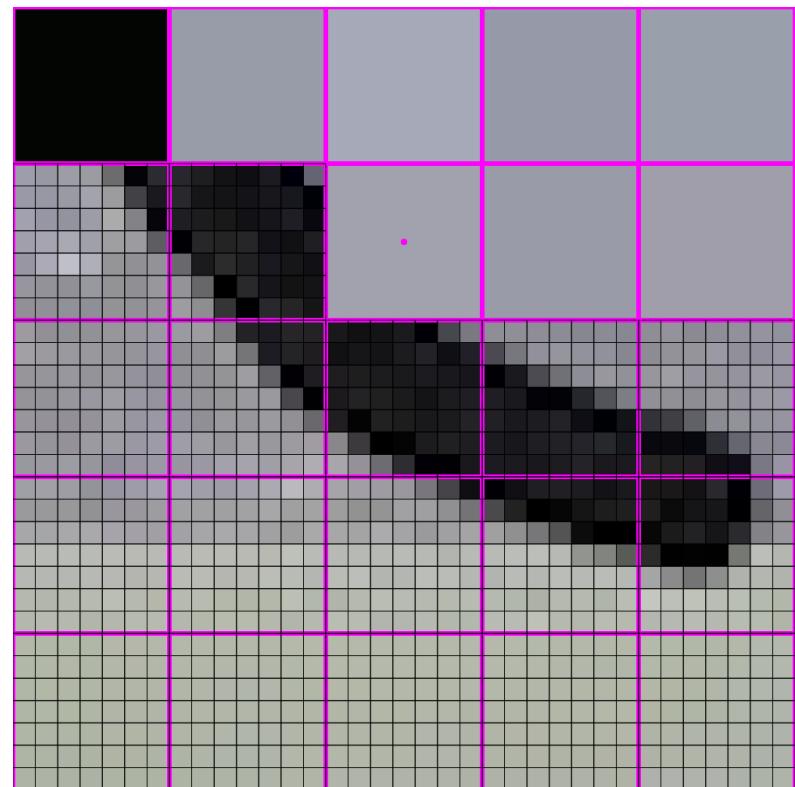
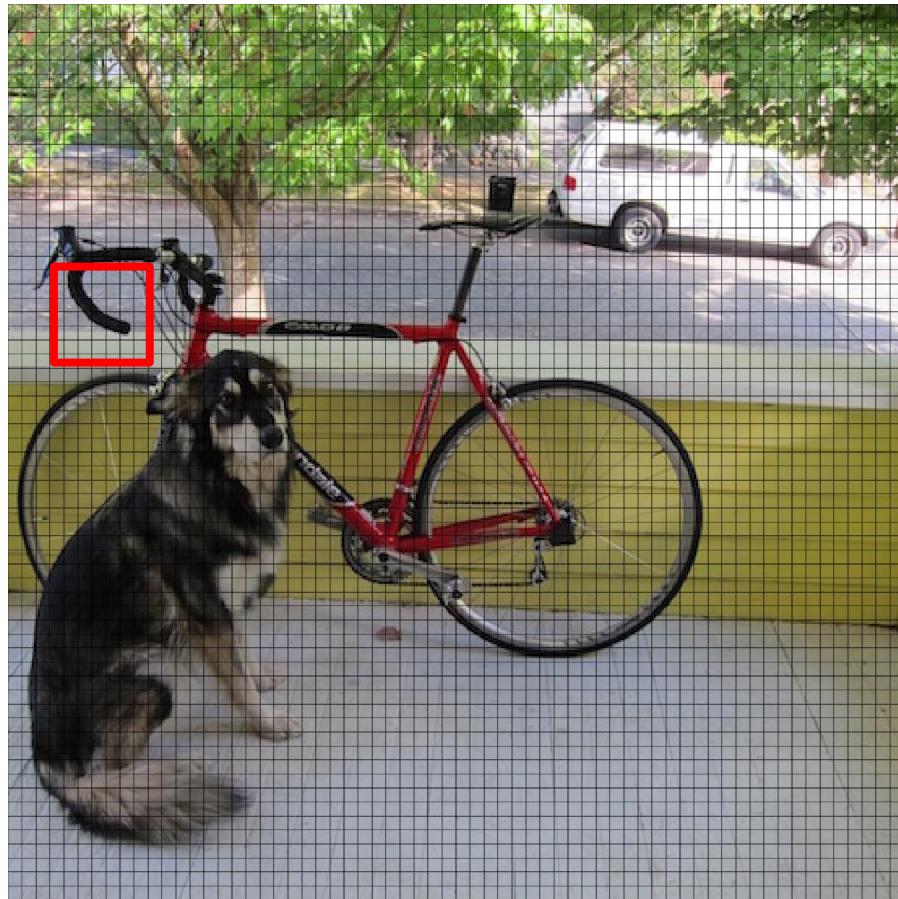
---

448x448 -> 64x64



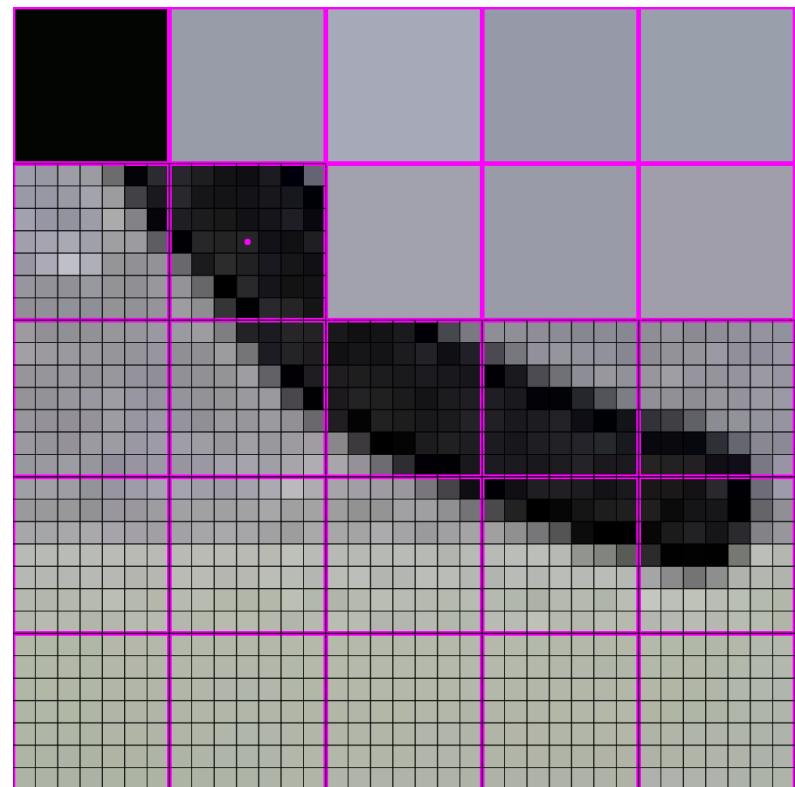
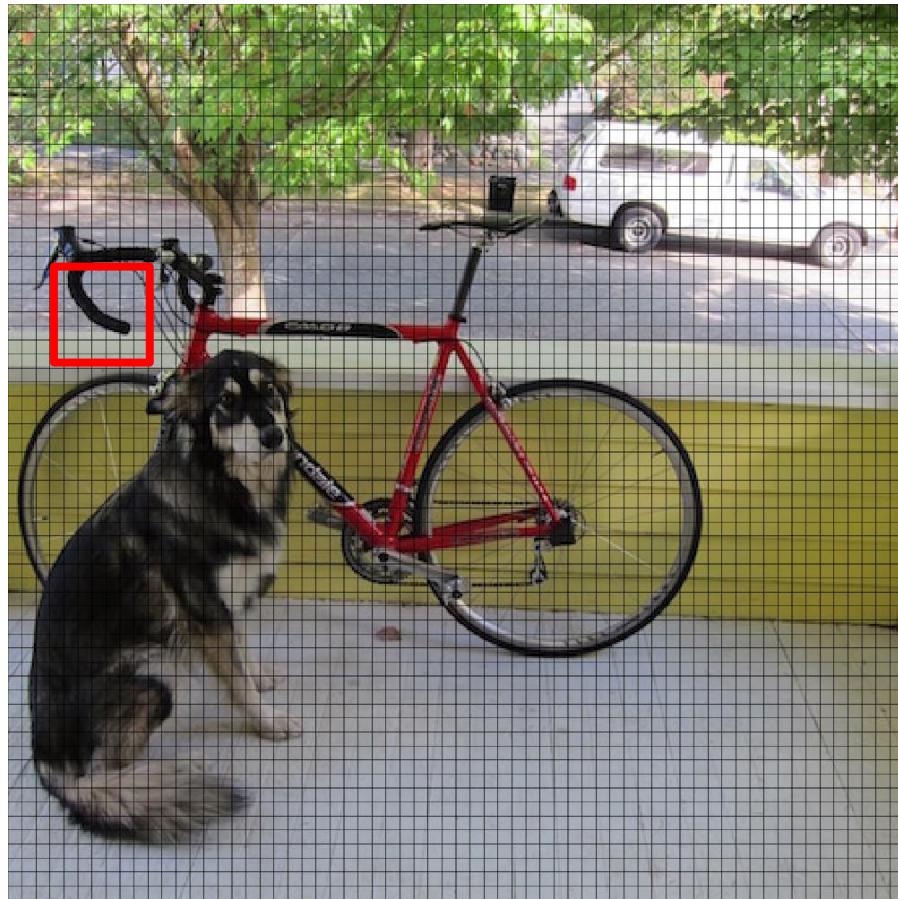
---

448x448 -> 64x64



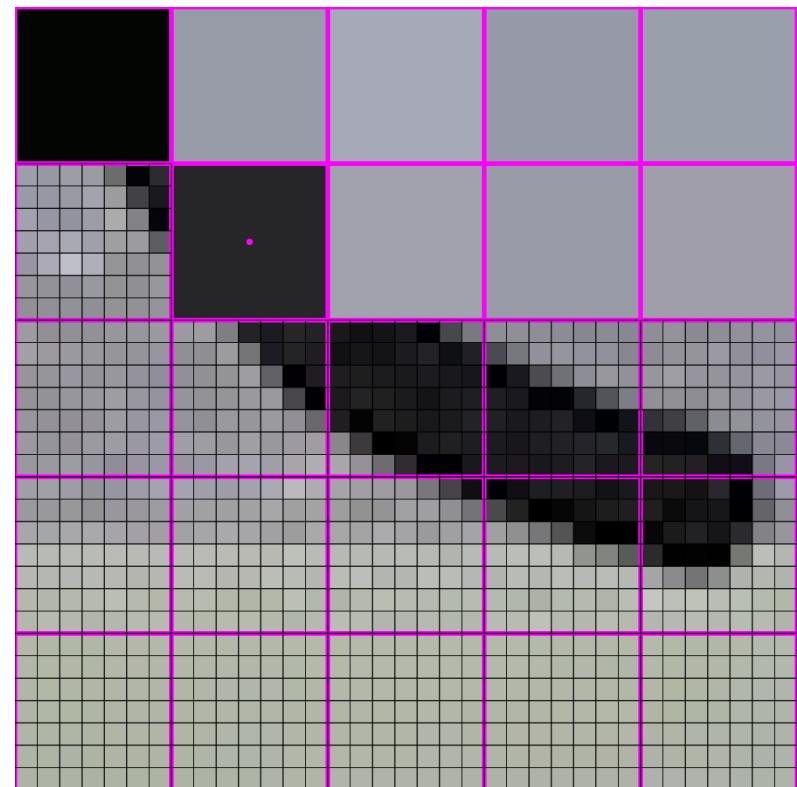
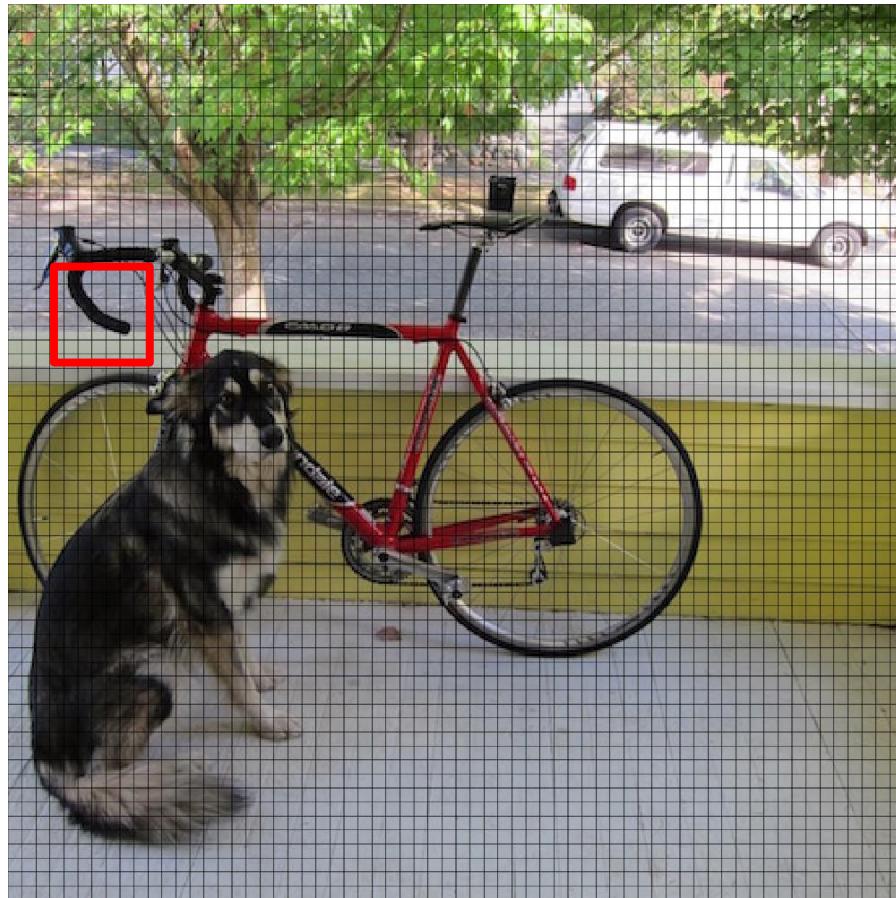
---

448x448 -> 64x64



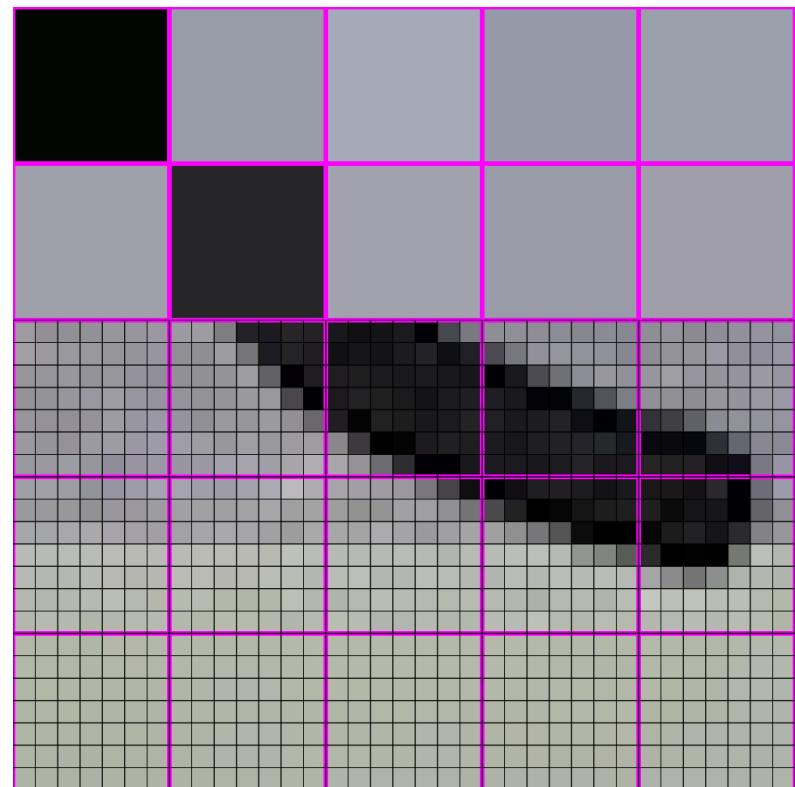
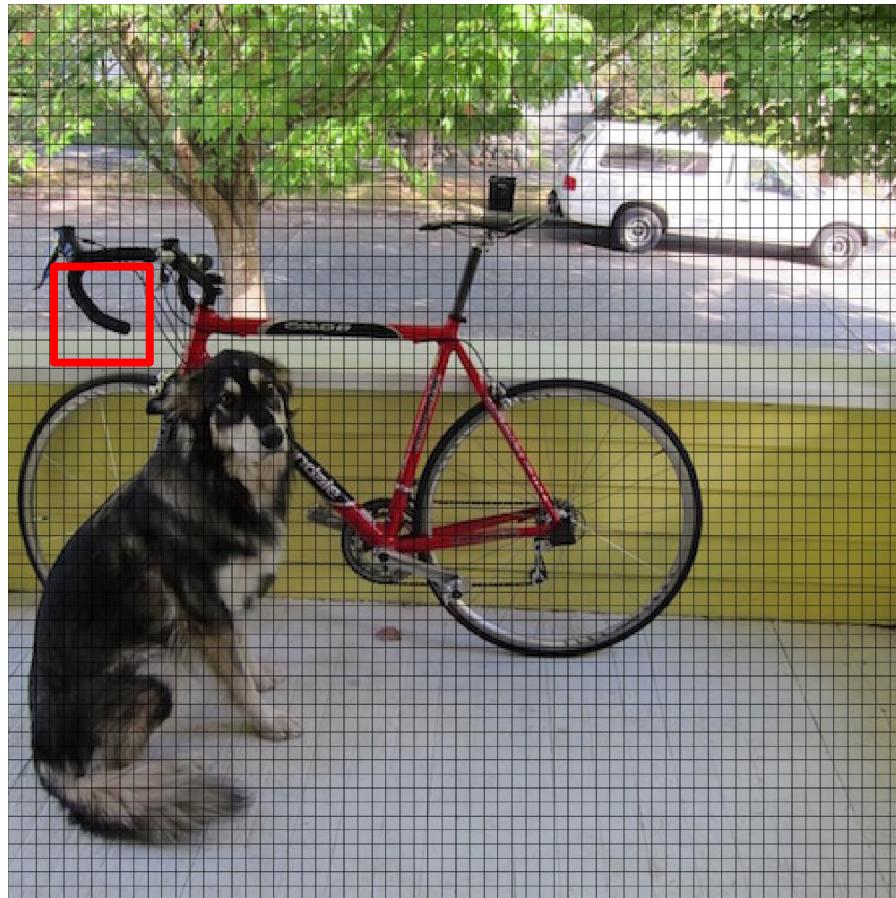
---

448x448 -> 64x64



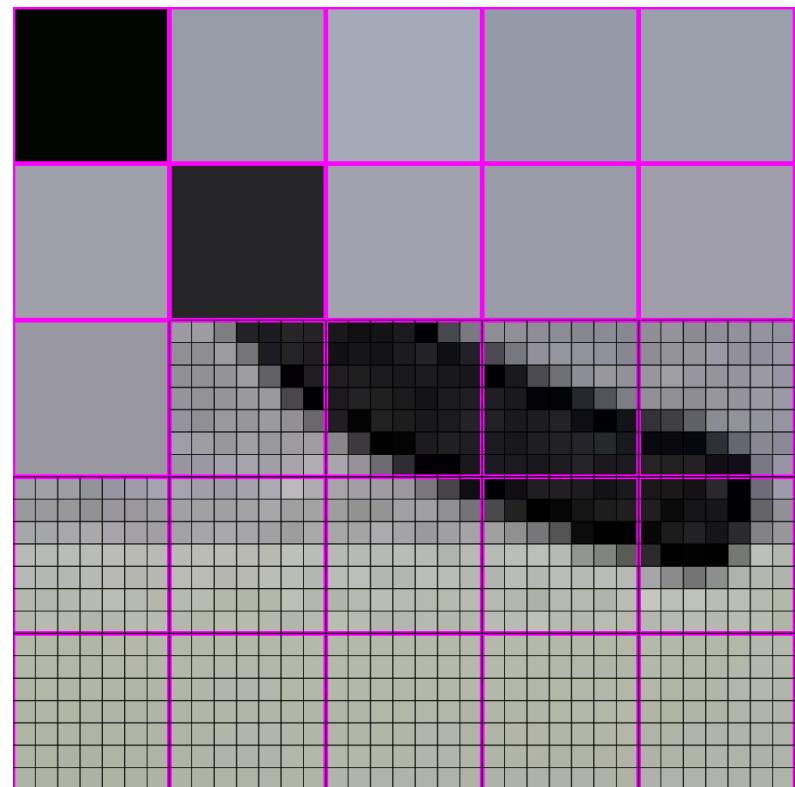
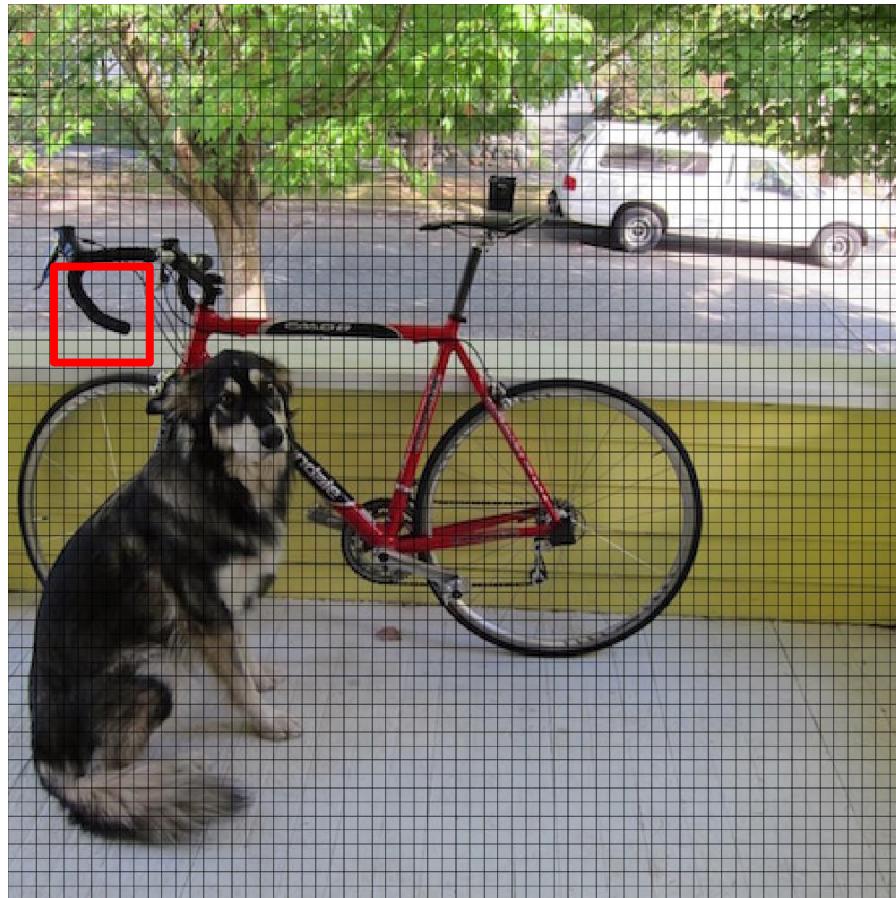
---

448x448 -> 64x64



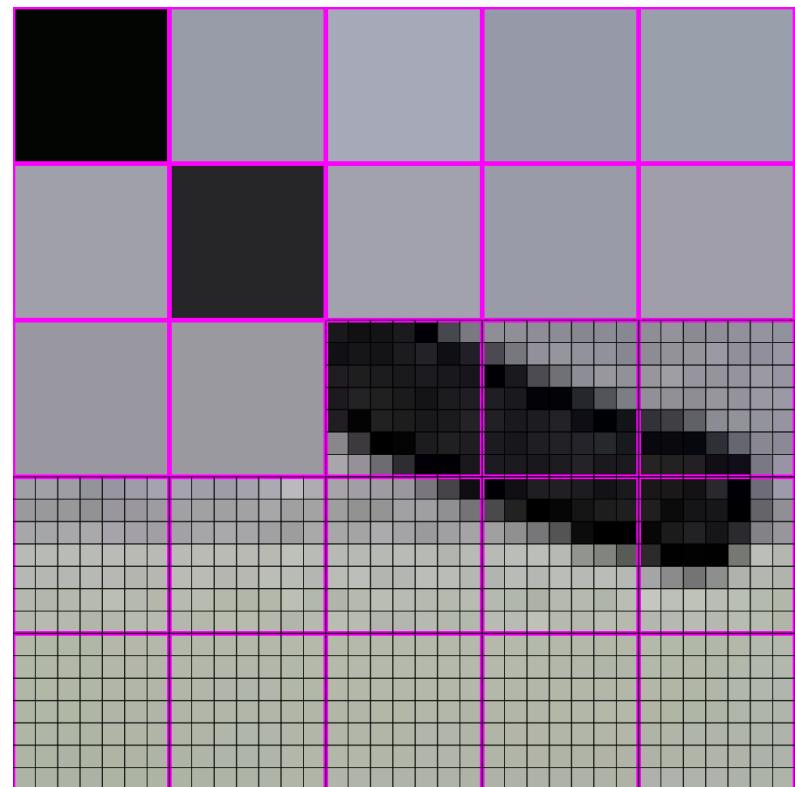
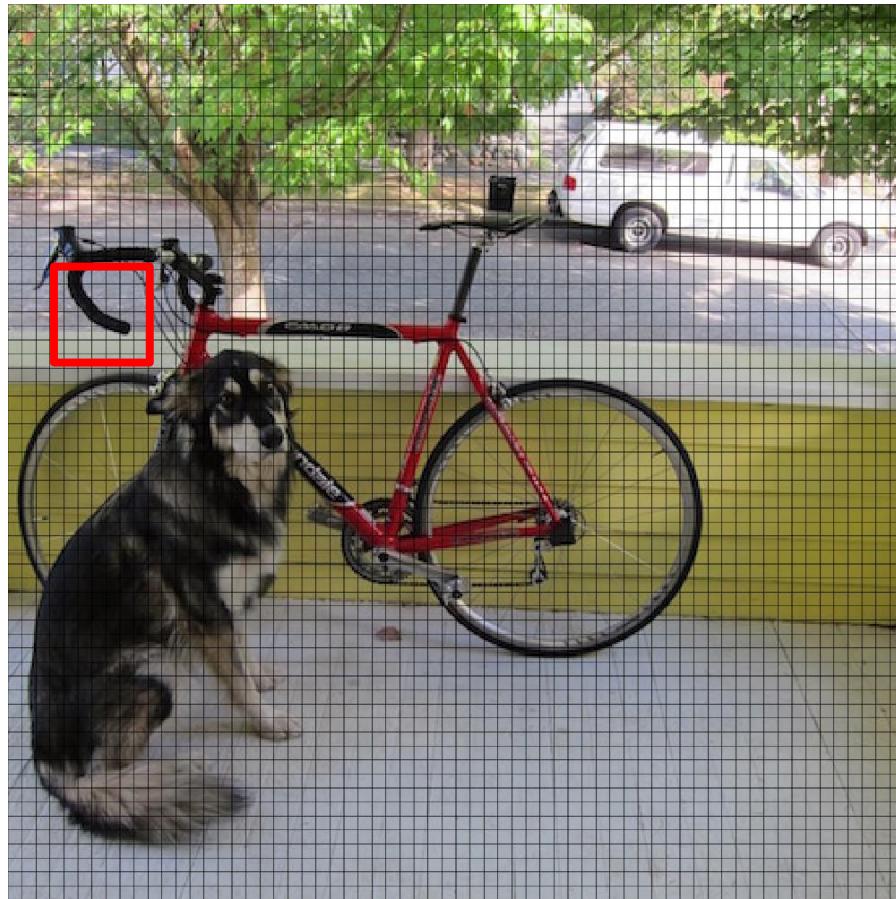
---

448x448 -> 64x64



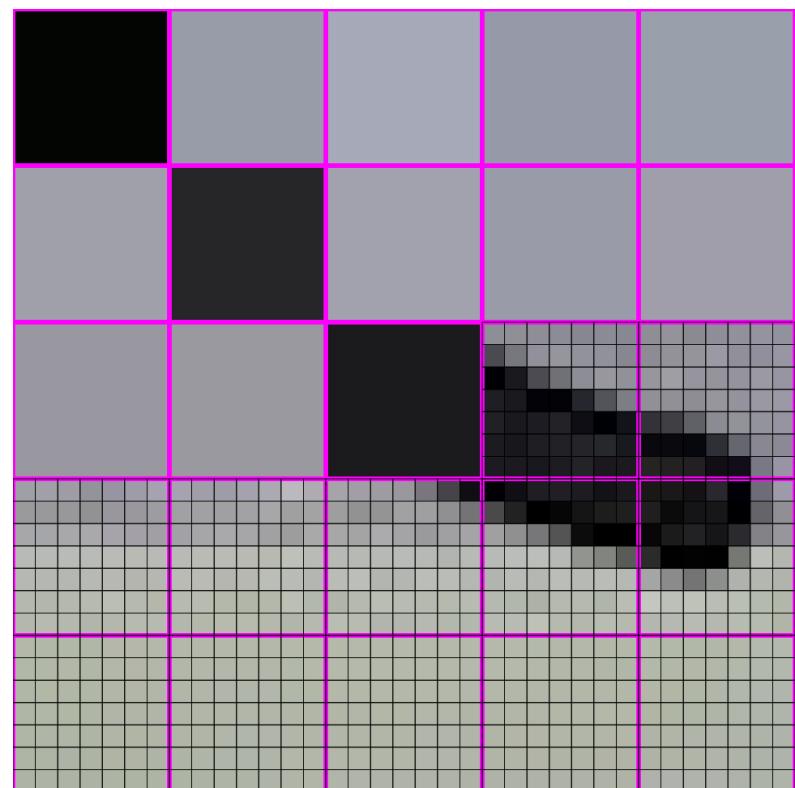
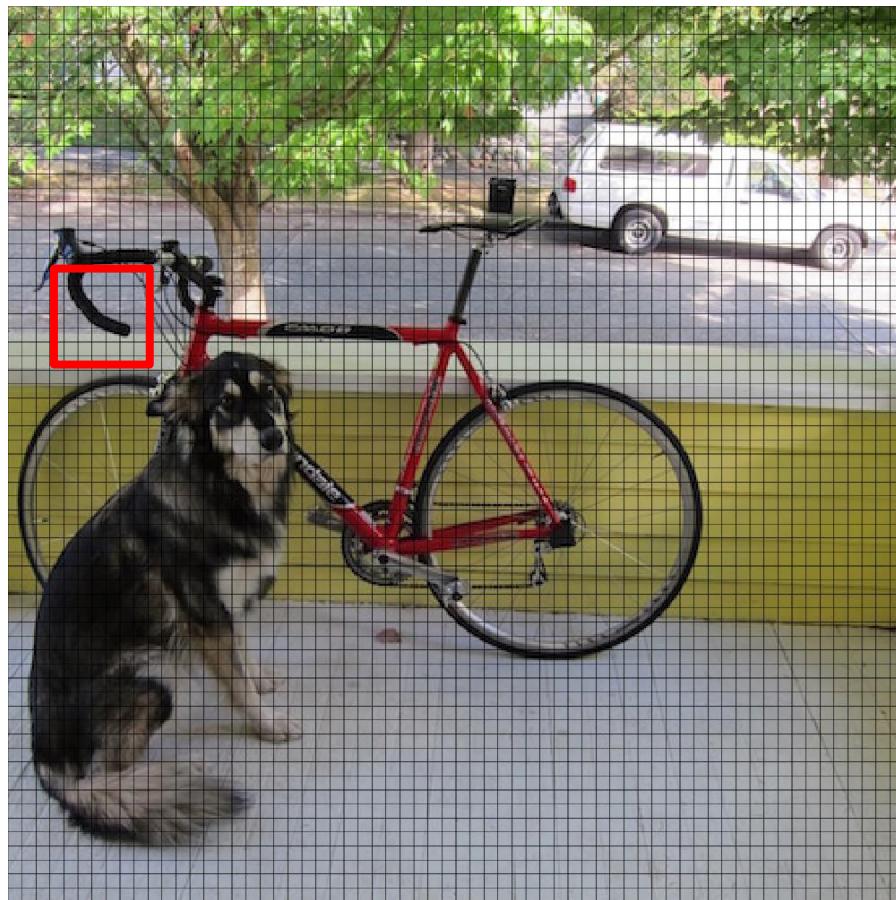
---

448x448 -> 64x64



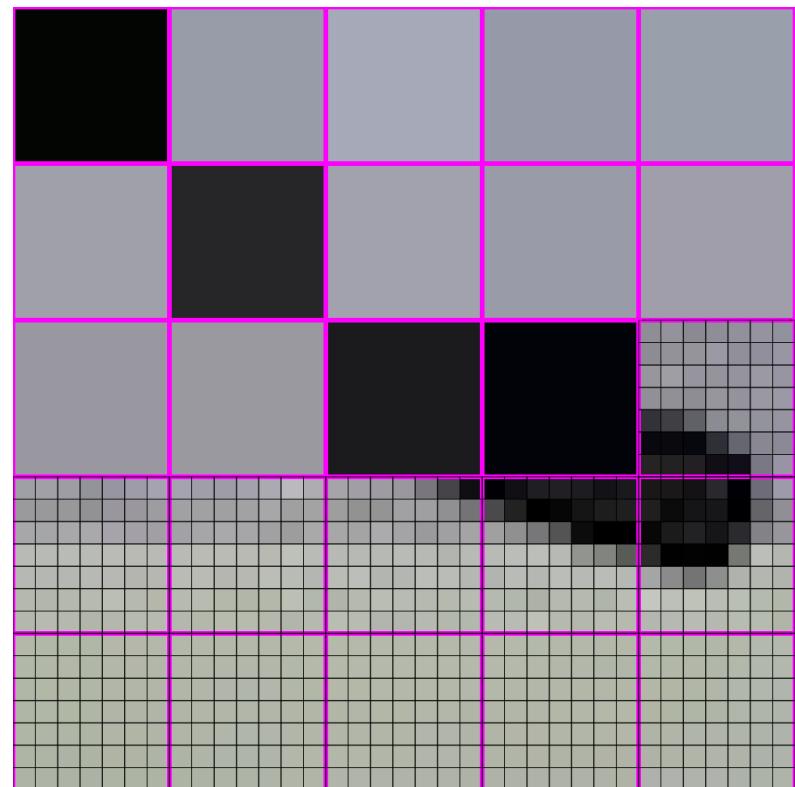
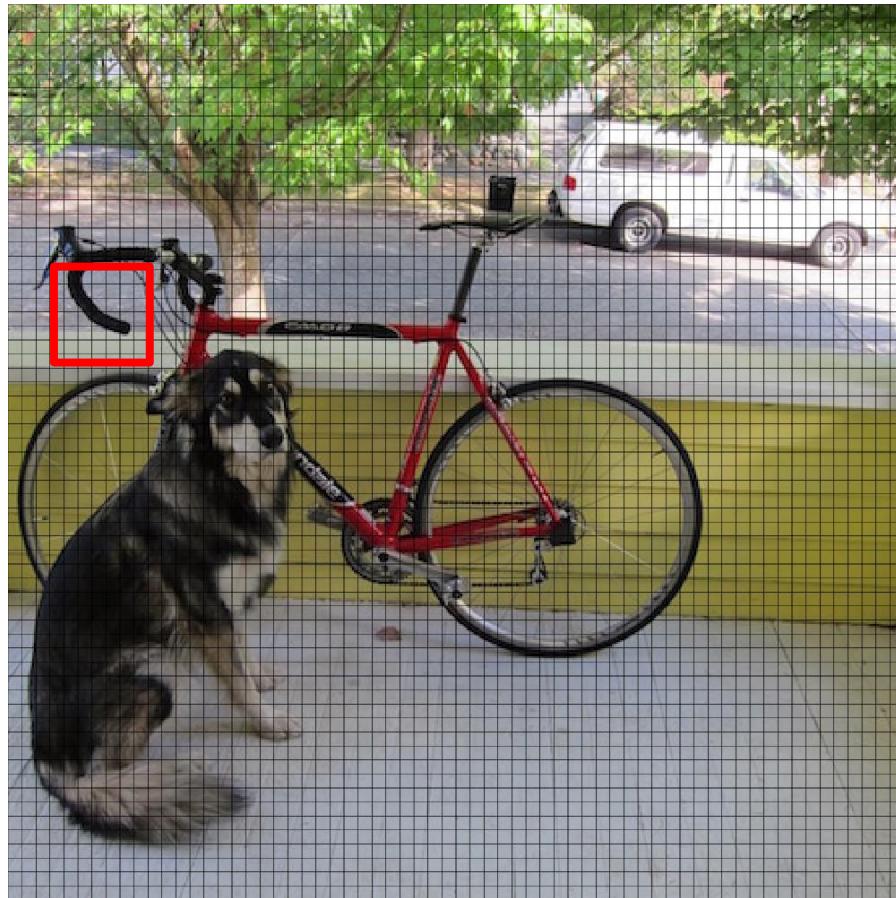
---

448x448 -> 64x64



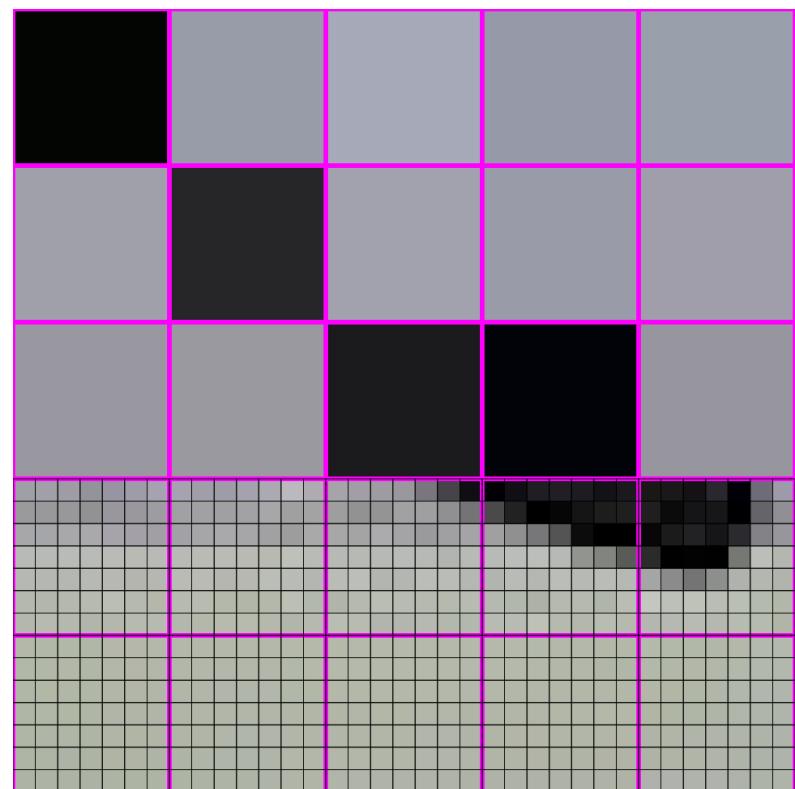
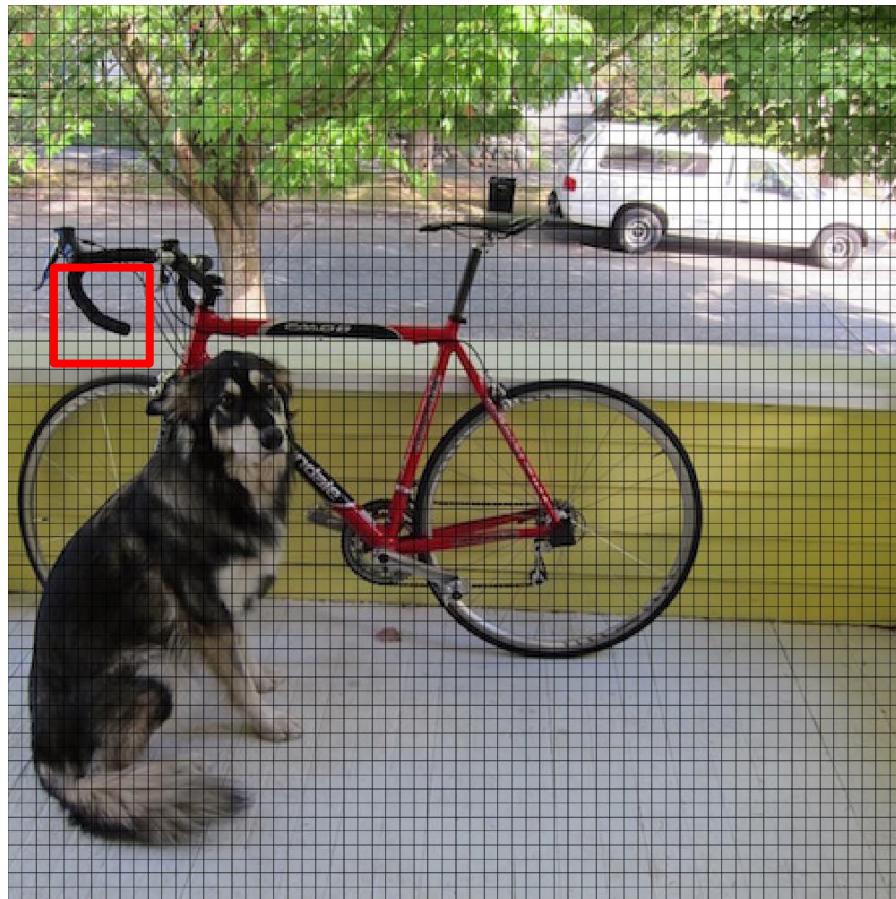
---

448x448 -> 64x64



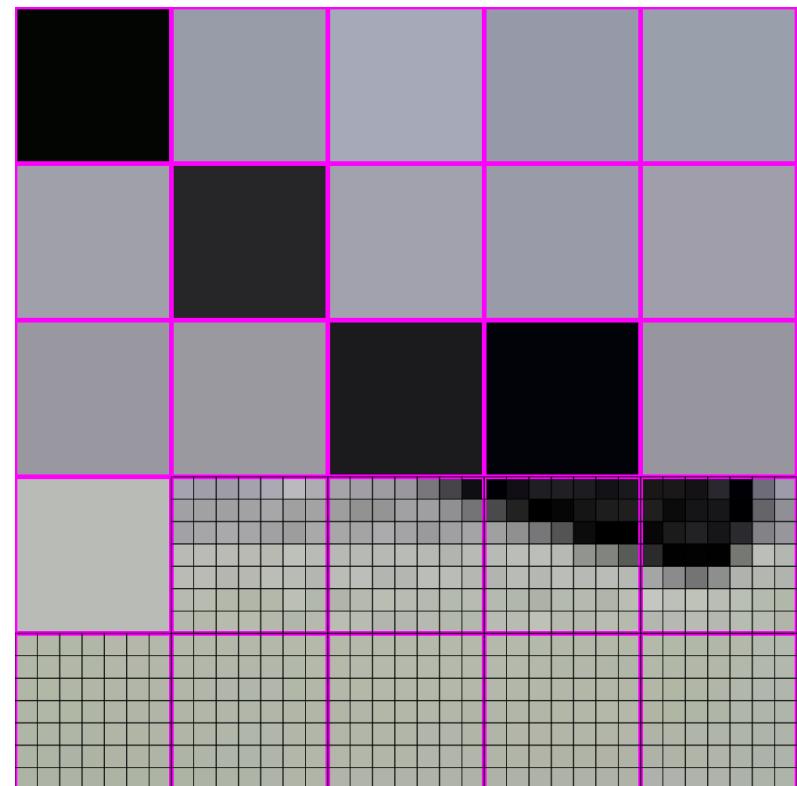
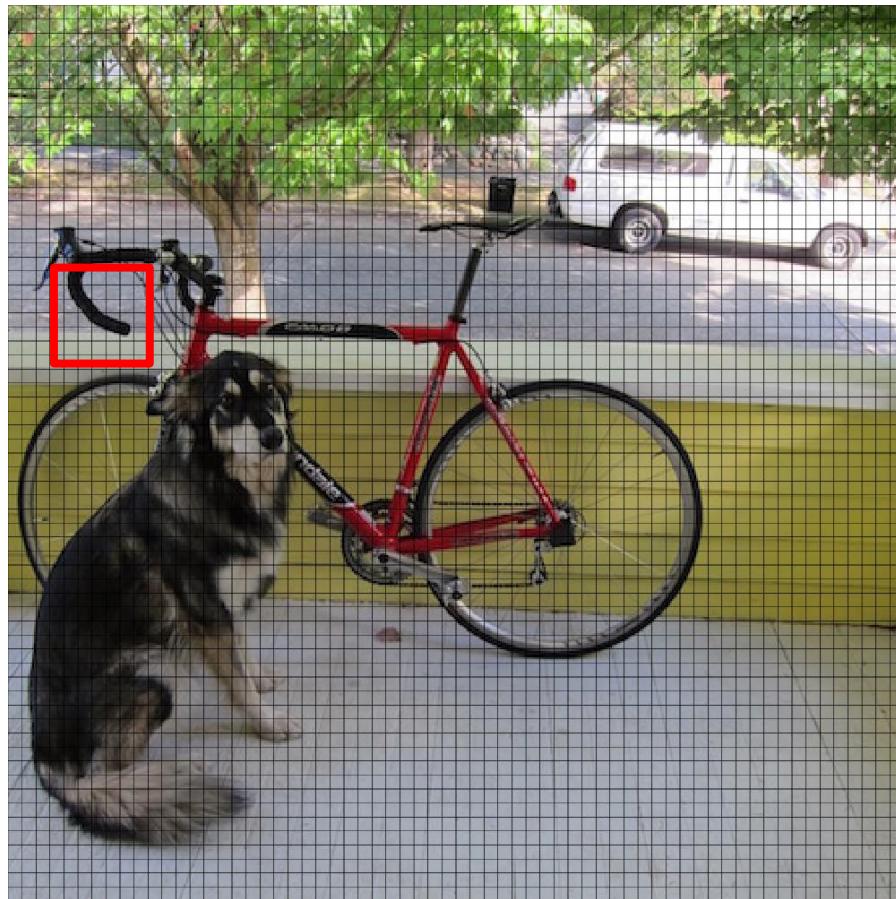
---

448x448 -> 64x64



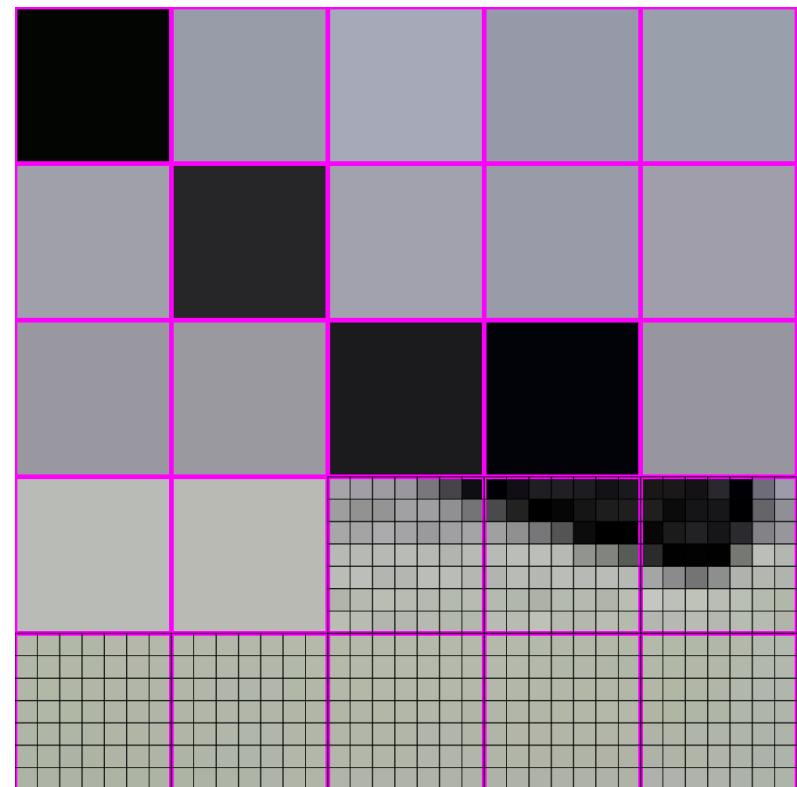
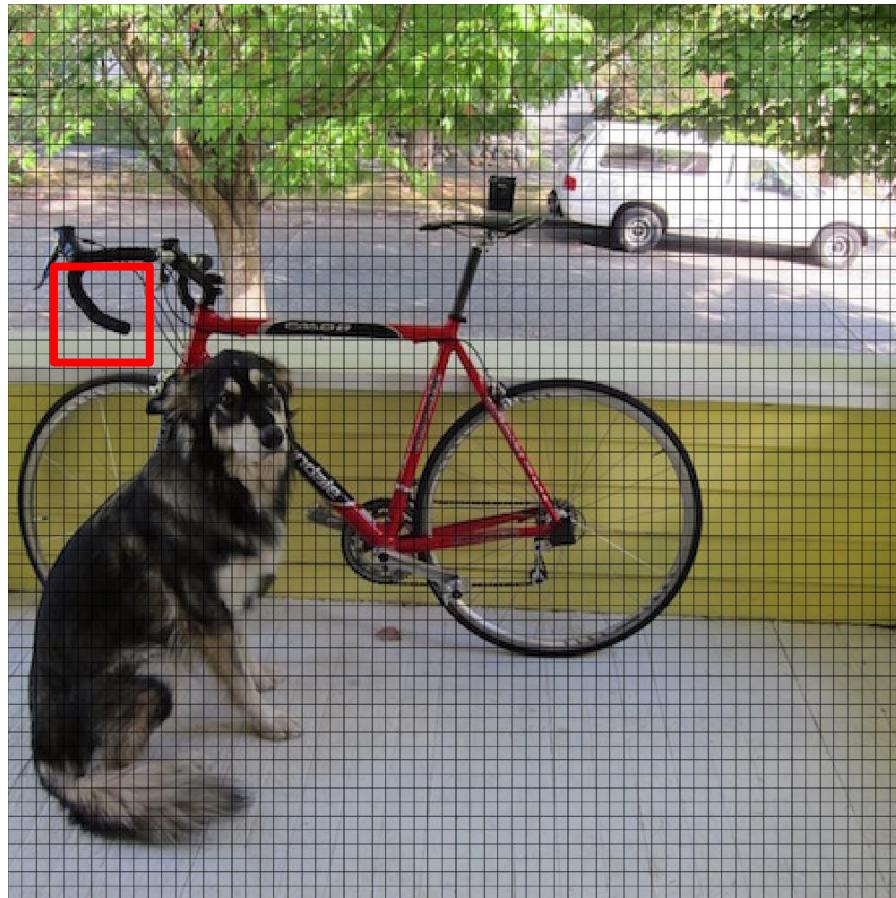
---

448x448 -> 64x64



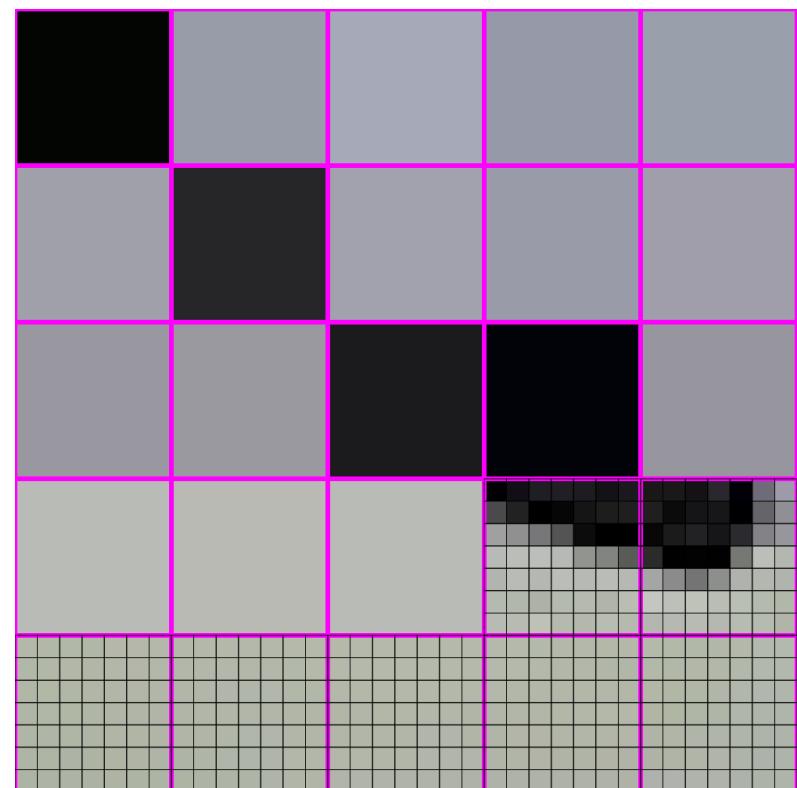
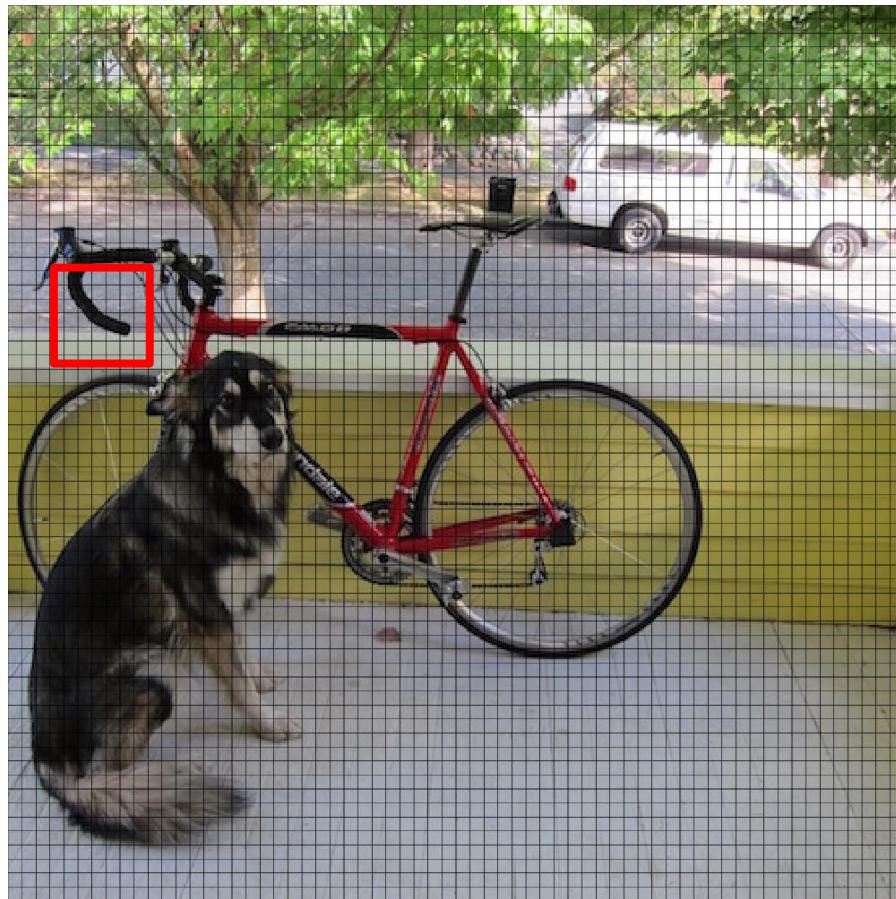
---

448x448 -> 64x64



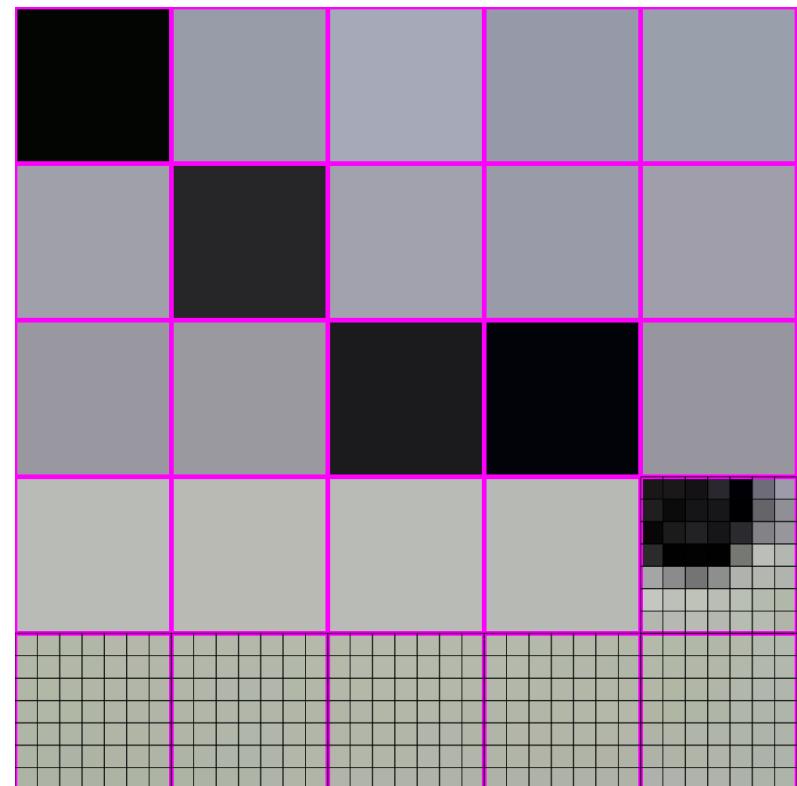
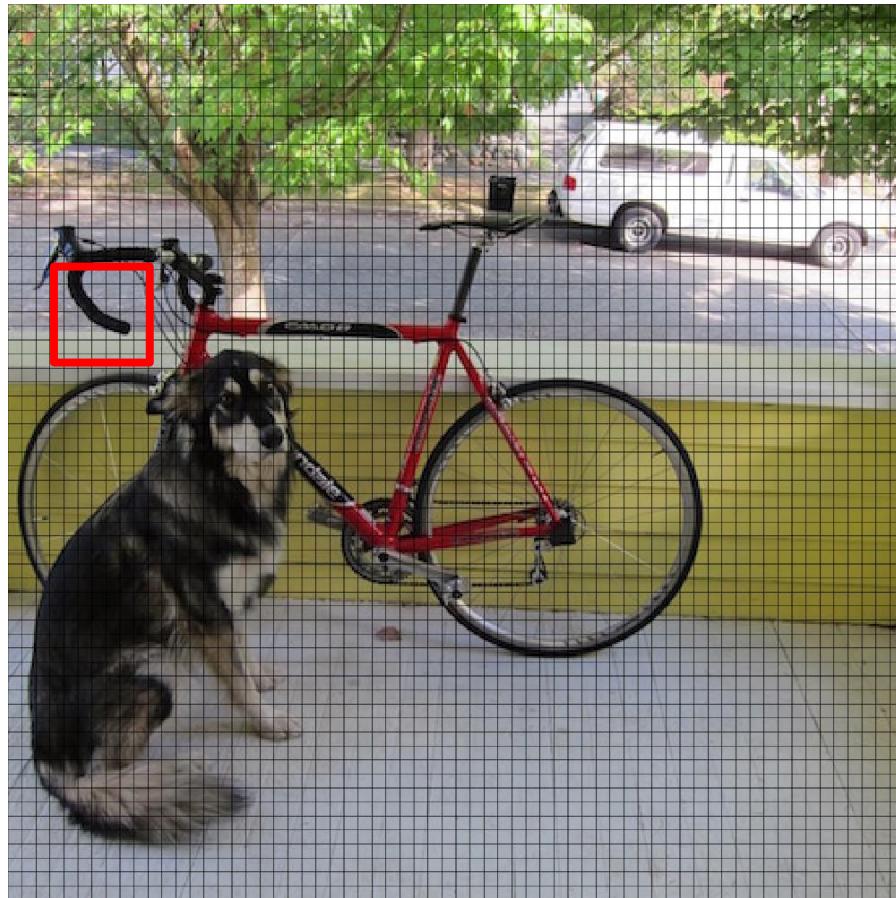
---

448x448 -> 64x64



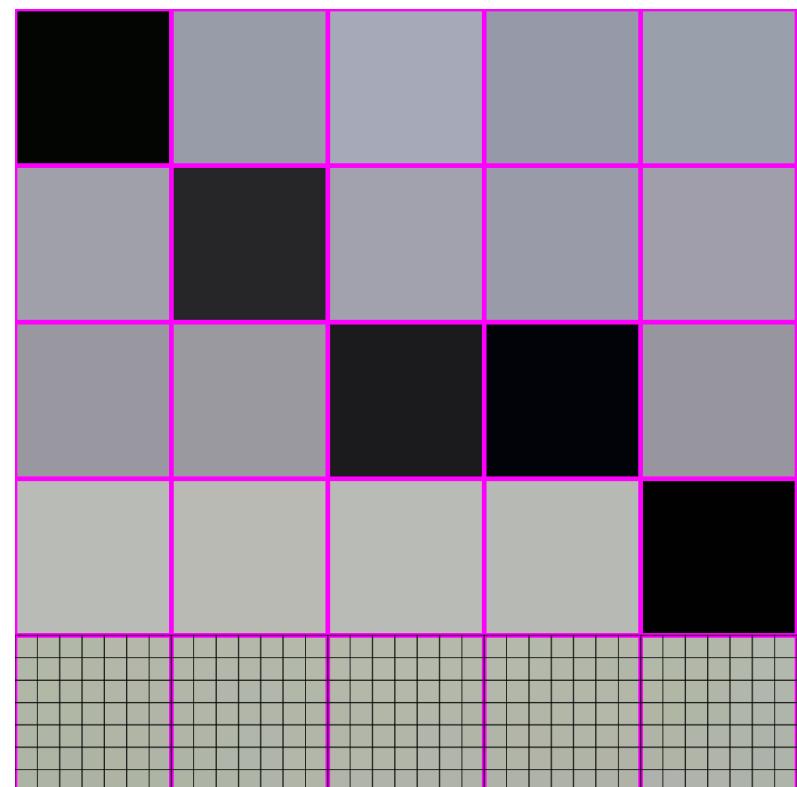
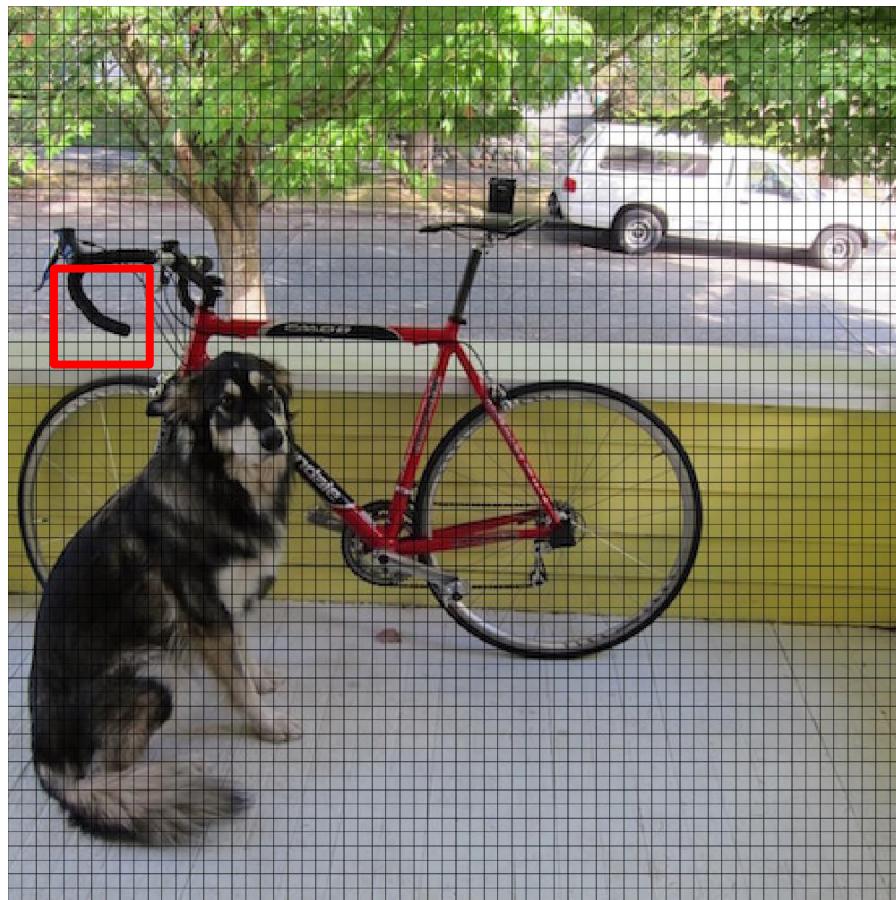
---

448x448 -> 64x64



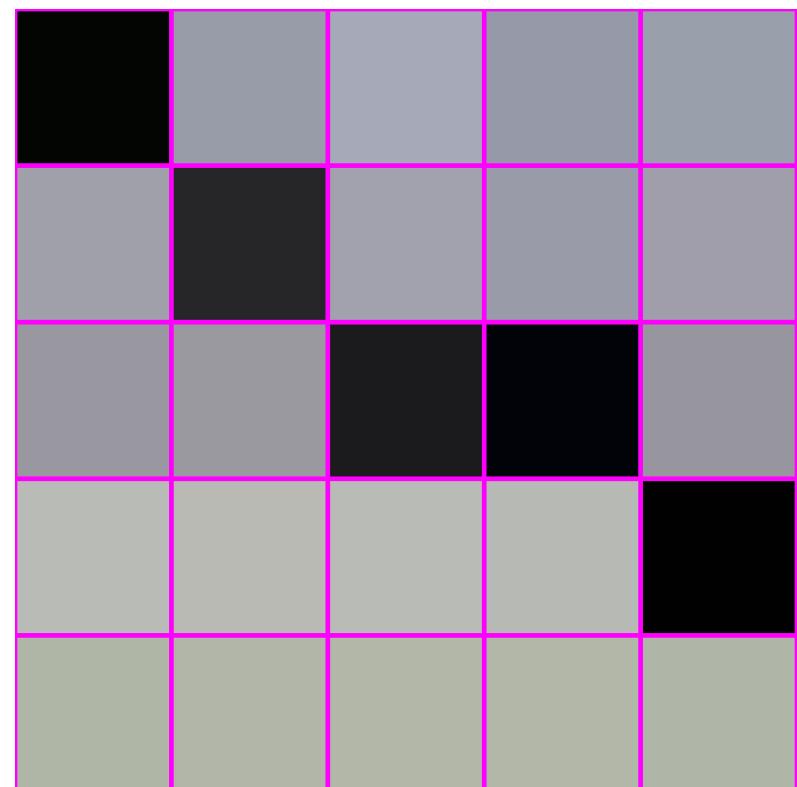
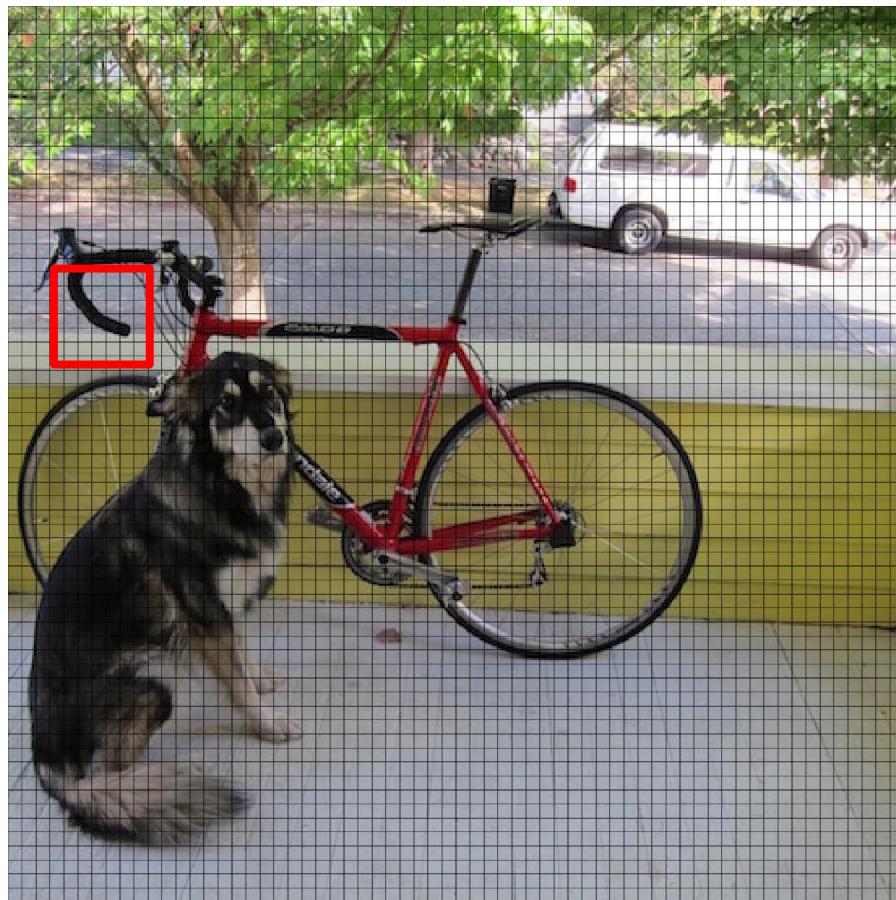
---

448x448 -> 64x64



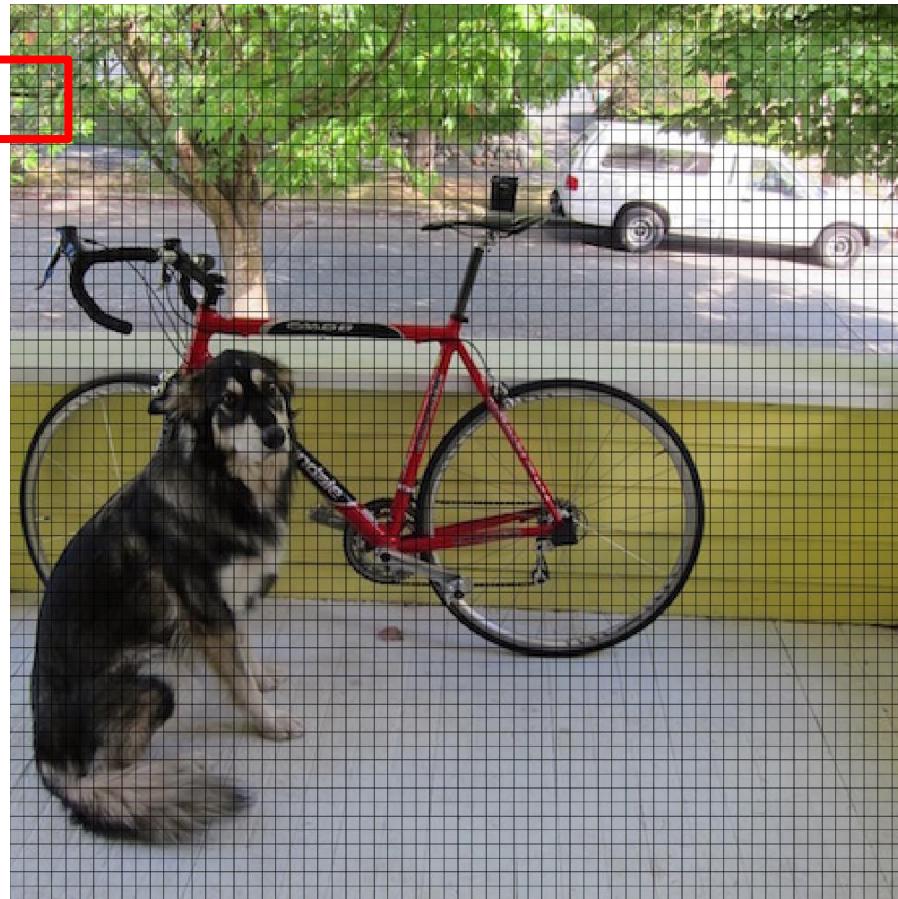
---

448x448 -> 64x64



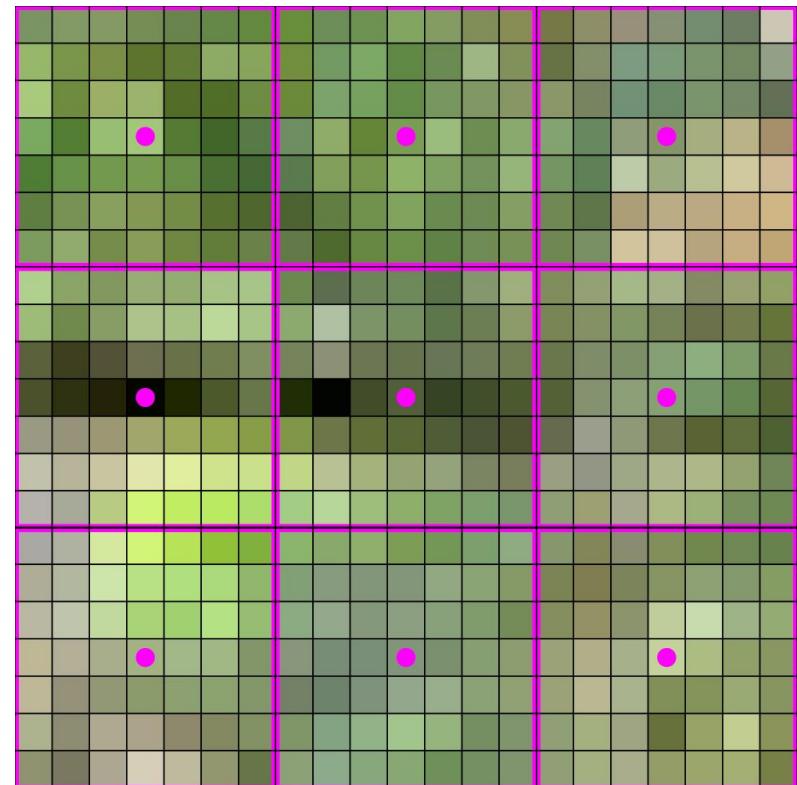
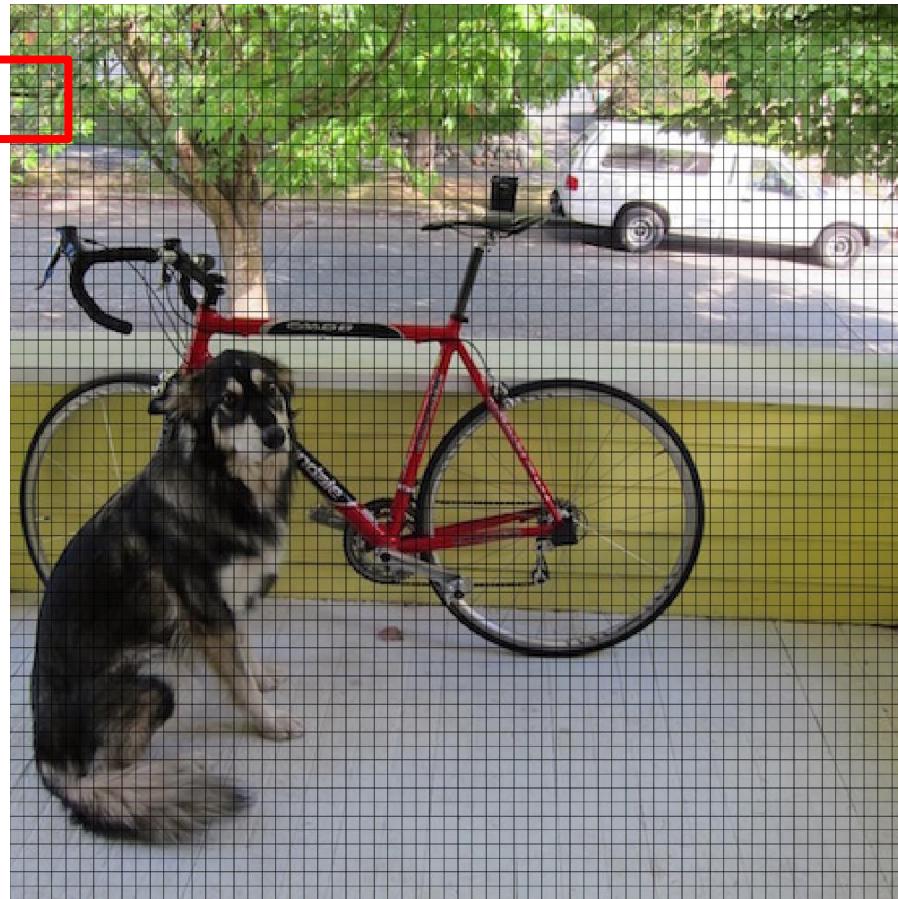
---

448x448 -> 64x64



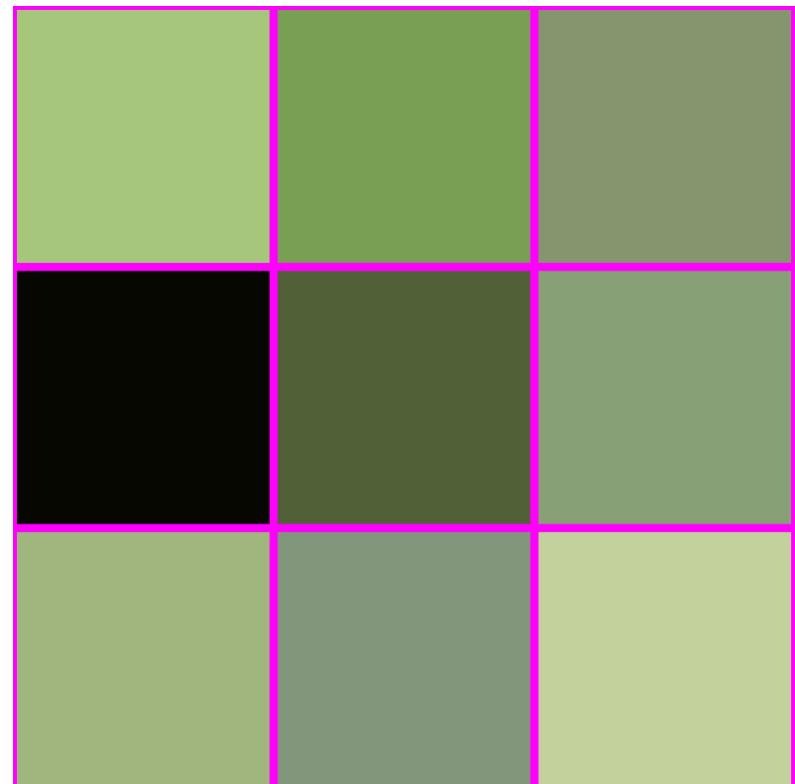
---

448x448 -> 64x64



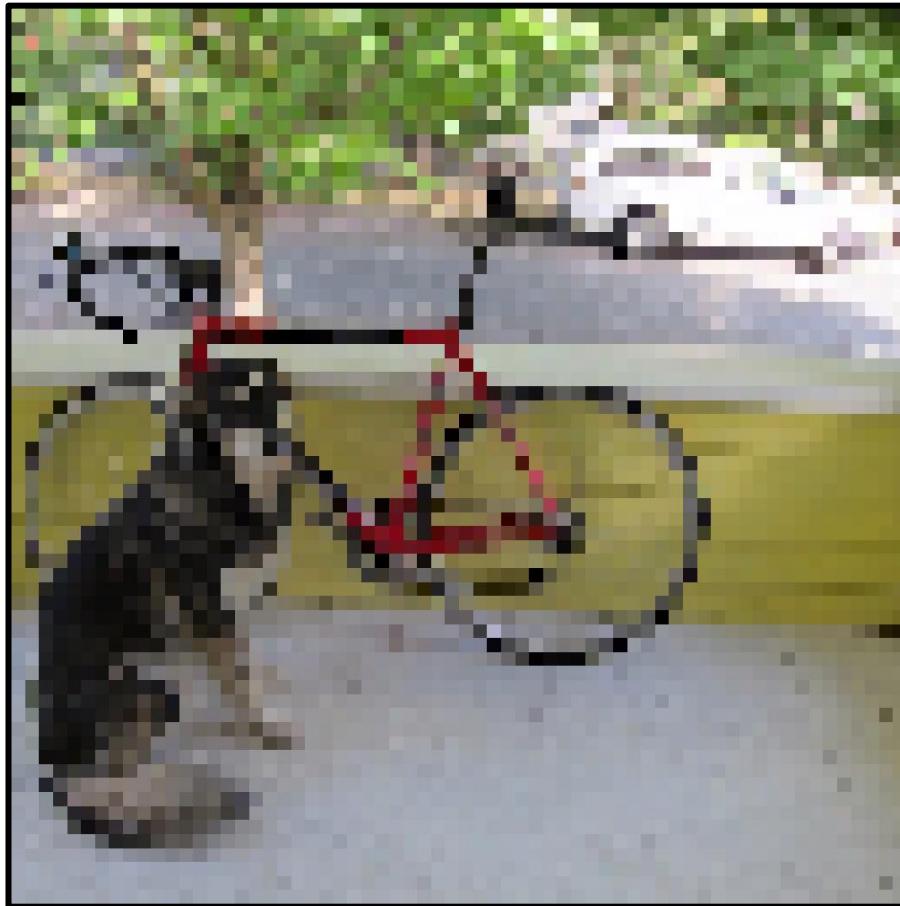
---

448x448 -> 64x64



---

# IS THIS ALL THERE IS??



---

# THERE IS A BETTER WAY!



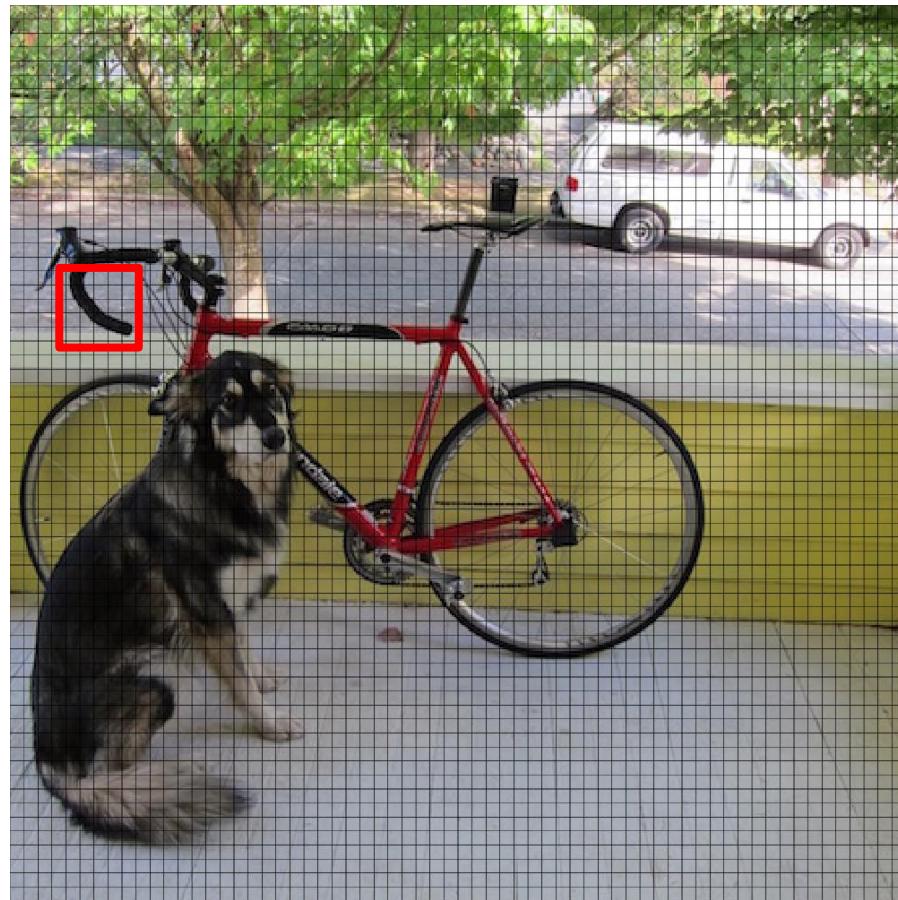
---

# LOOK AT HOW MUCH BETTER



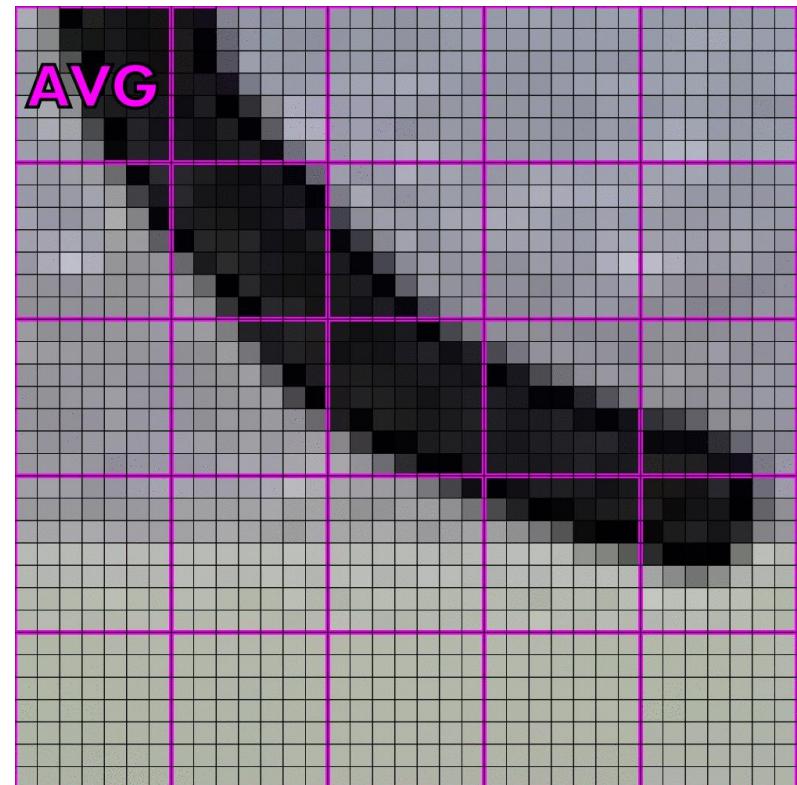
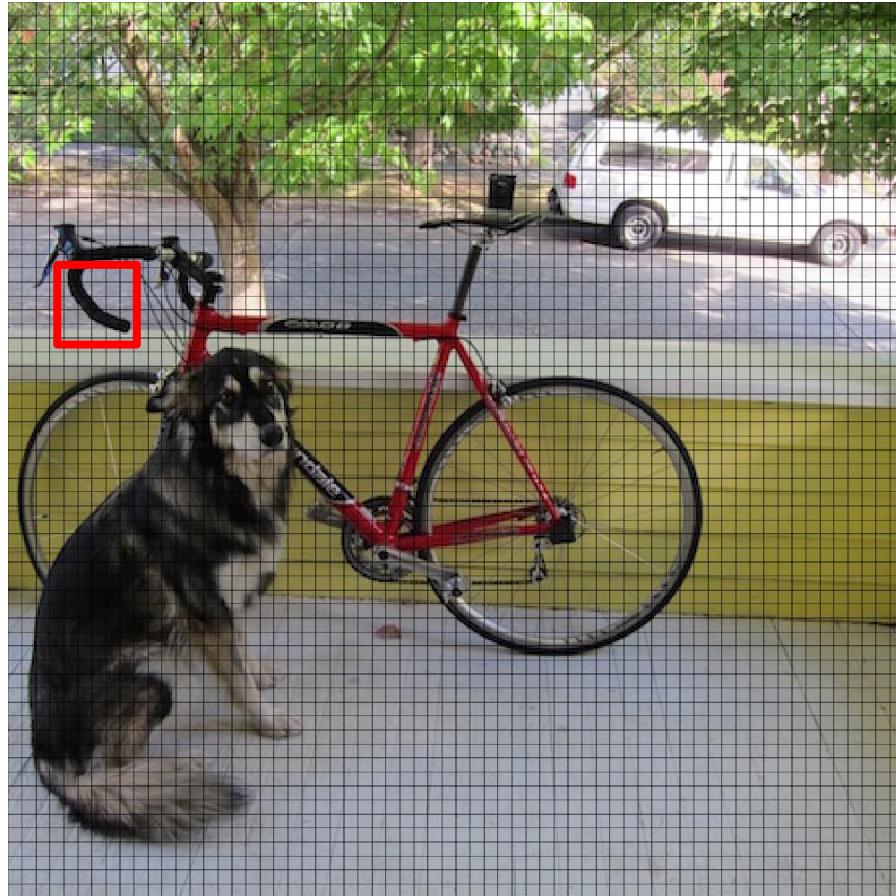
---

# How do?

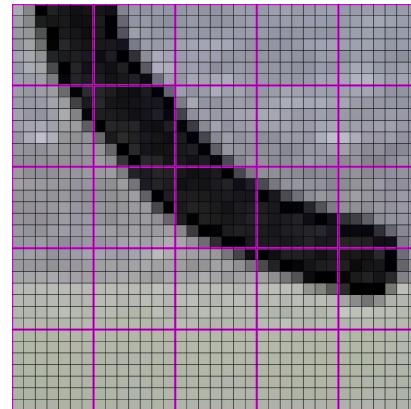


---

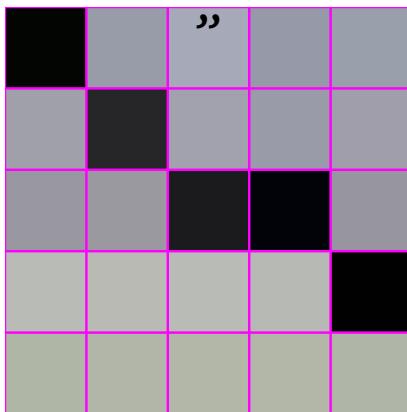
# How do? Averaging!



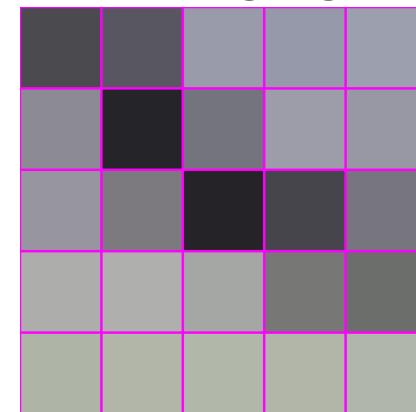
# How do? Averaging!



“interpolation



averaging



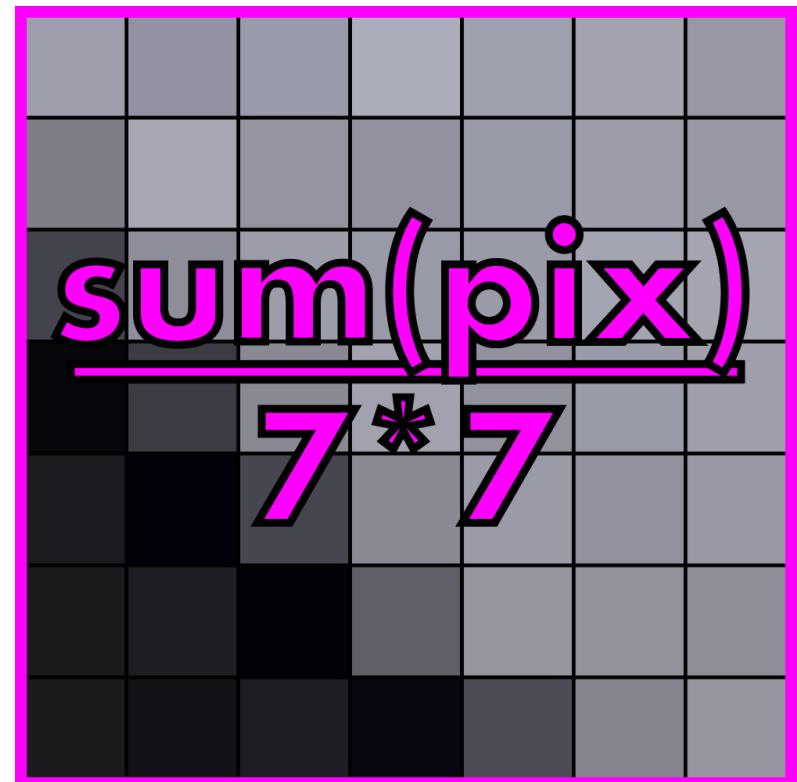
---

# What is averaging?



---

# What is averaging? A weighted sum



# What is averaging? A weighted sum

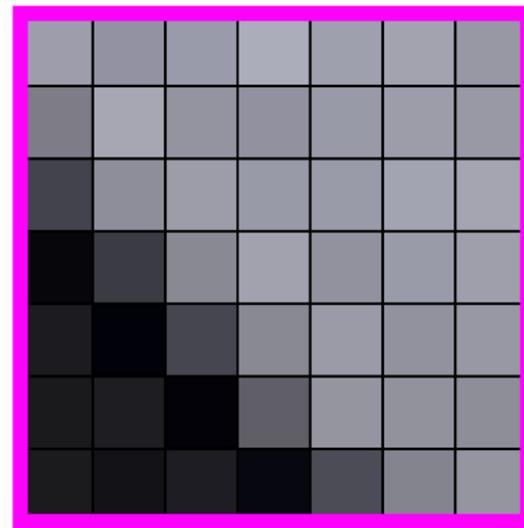
$$\text{sum} \left[ \frac{1}{49} \right]$$

# Call this operation “convolution”

*Filter or kernel*

$\frac{1}{49}$

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

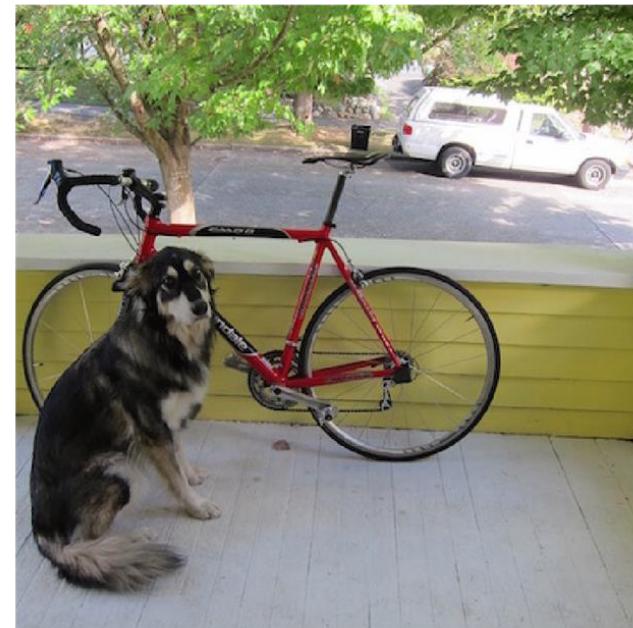
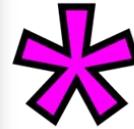


Note: multiplying an image section by a filter is actually called “correlation” and convolution involves inverting the filter first, but since our filters are generally symmetric, we call everything convolution. This is what all computer vision people do.

# Convolutions on larger images

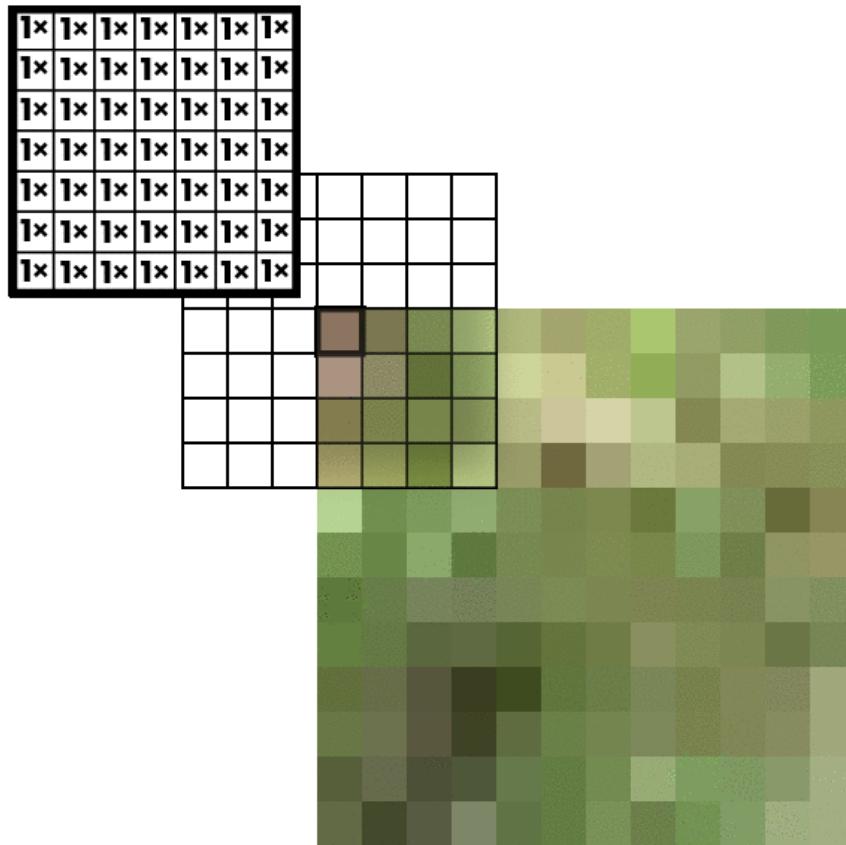
$\frac{1}{49}$

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



# Kernel slides across image

1  
49



---

# Kernel slides across image

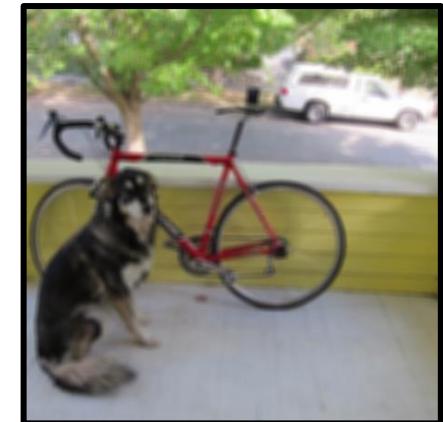
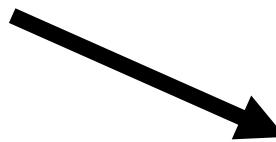
What happens at the edges of the images?

1. Zero padding: easiest but can give bad results
2. Duplicate the outermost row or column
3. Use wraparound

# Convolutions on larger images

$\frac{1}{49}$

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



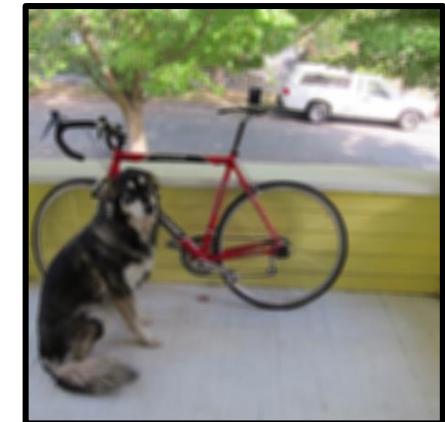
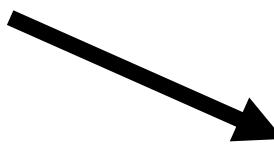
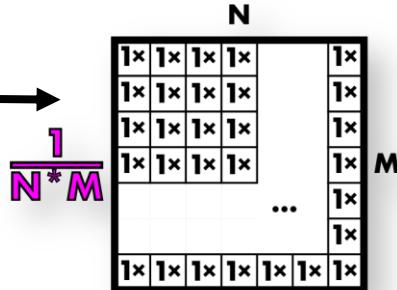
# This is called box filter

$\frac{1}{49}$

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



Box filters



# Box filters smooth image

$\frac{1}{49}$

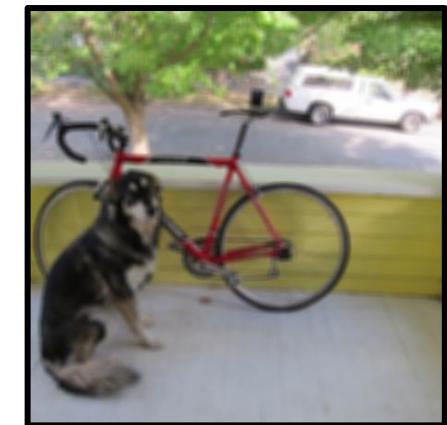
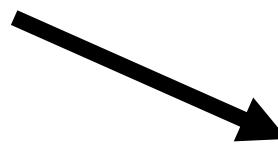
1x							
1x							
1x							
1x							
1x							
1x							
1x							



Box filters

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

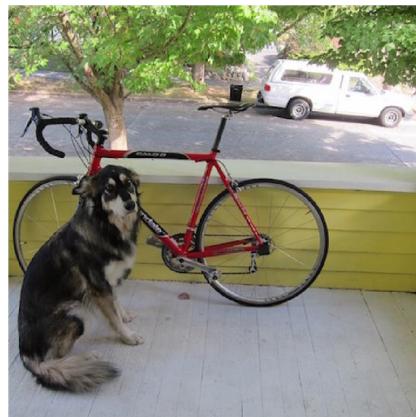
1x	1x	1x	1x	1x			
1x	1x	1x	1x	1x			
1x	1x	1x	1x	1x			
1x	1x	1x	1x	1x			
1x	1x	1x	1x	1x			
					...		
1x	1x	1x	1x	1x	1x	1x	1x
1x	1x	1x	1x	1x	1x	1x	1x



# Box filters smooth image

$\frac{1}{49}$

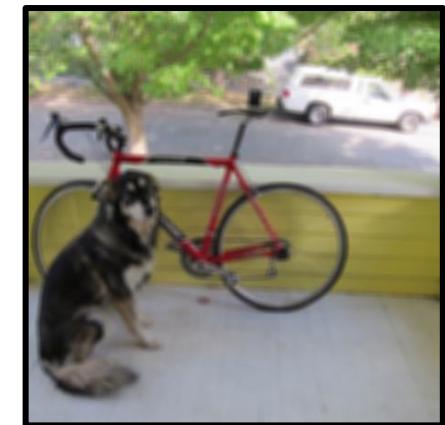
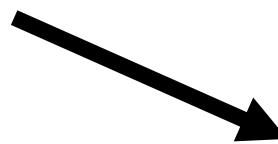
1x							
1x							
1x							
1x							
1x							
1x							
1x							



Box filters

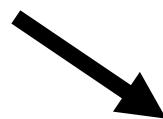
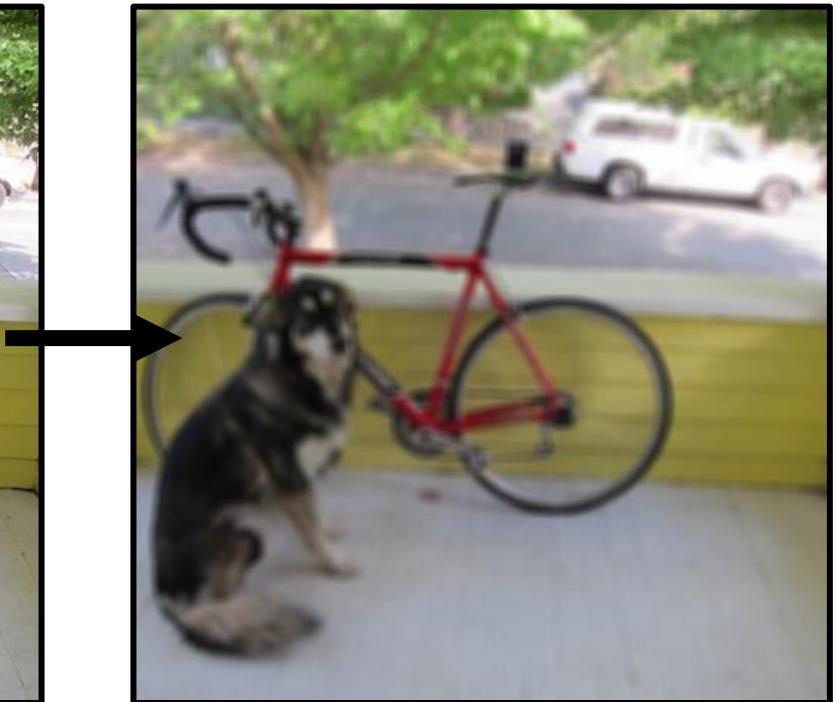
$$\frac{1}{N \times M}$$

1x	1x	1x	1x	1x			
1x	1x	1x	1x	1x			
1x	1x	1x	1x	1x			
1x	1x	1x	1x	1x			
1x	1x	1x	1x	1x			
					...		
1x	1x	1x	1x	1x	1x	1x	1x
1x	1x	1x	1x	1x	1x	1x	1x



---

# Now we resize our smoothed image



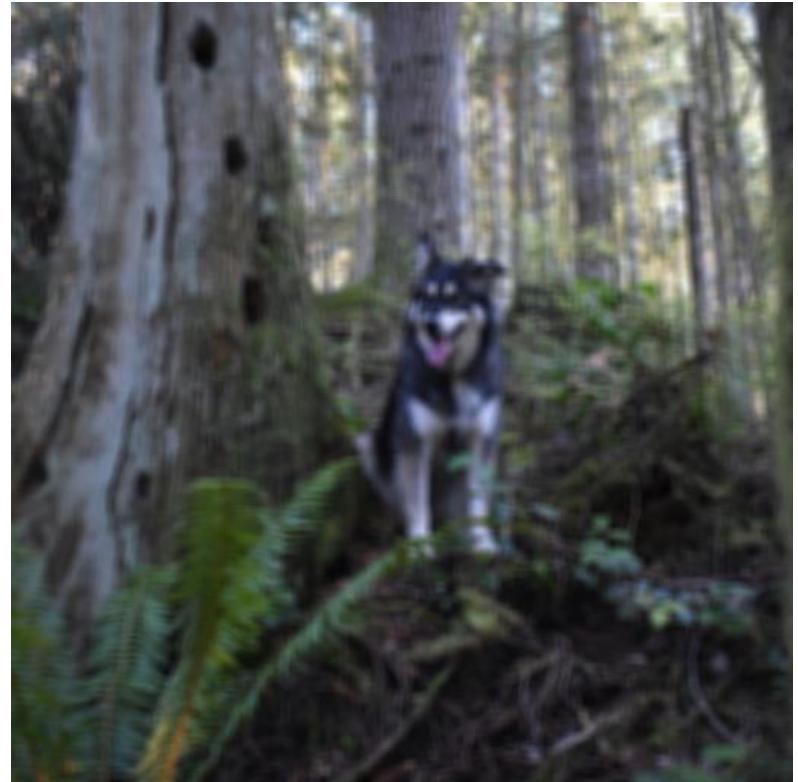
---

So much better!



---

## Box filters have artifacts

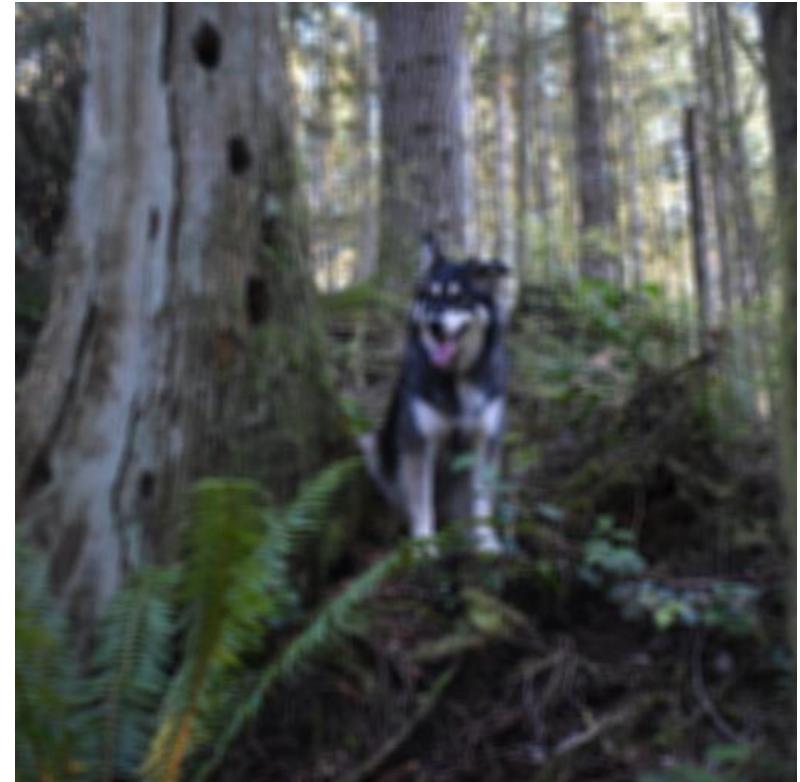


Box filters have artifacts



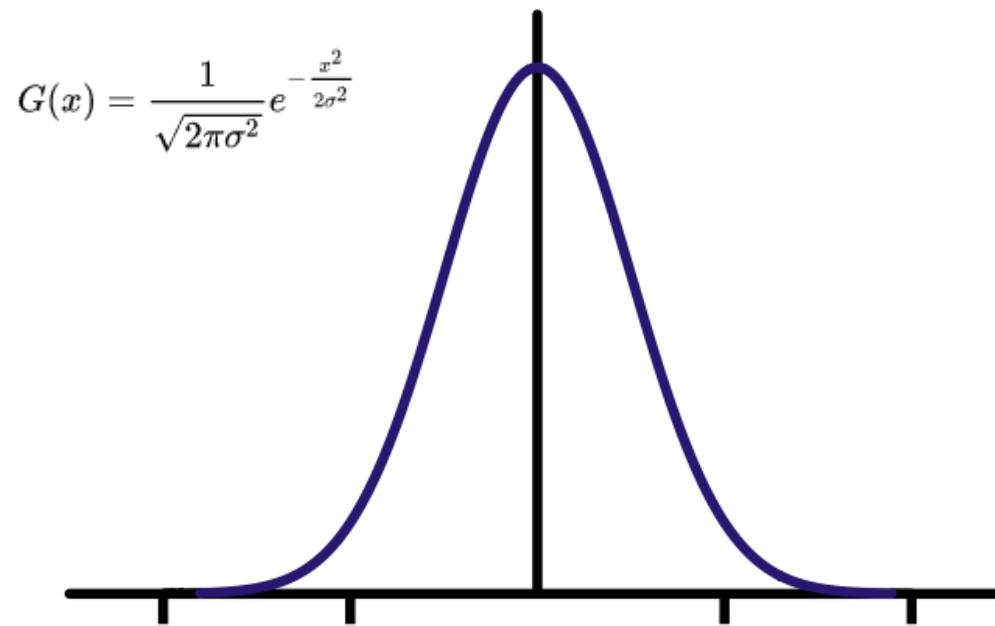
---

# We want a smoothly weighted kernel



---

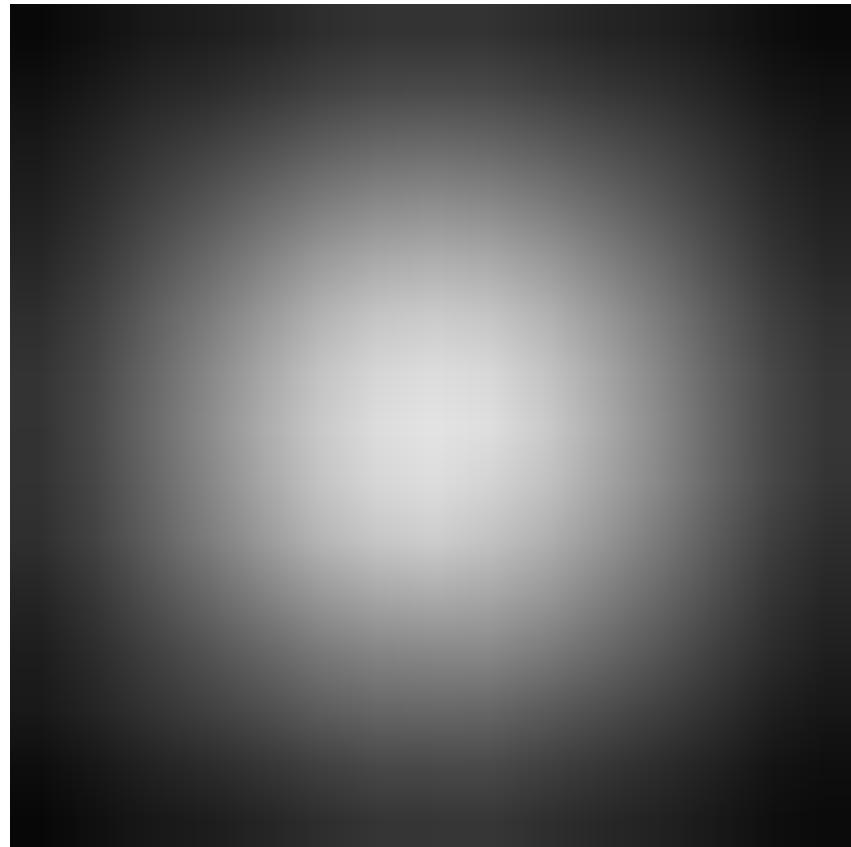
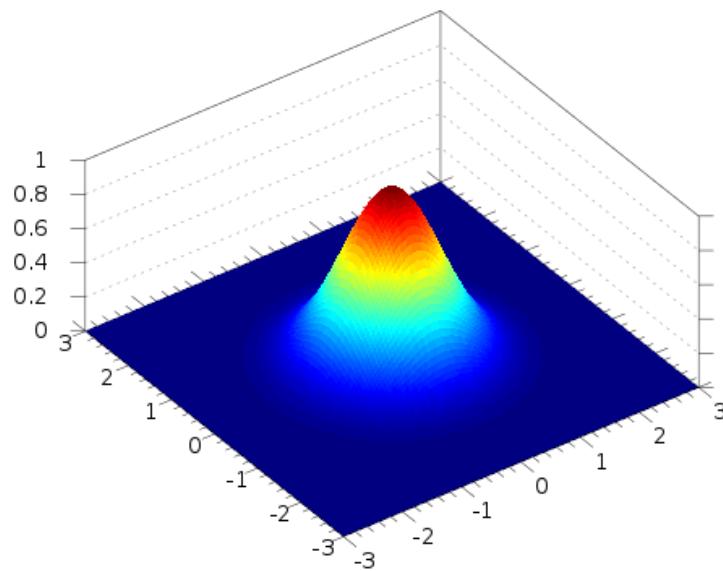
# Gaussians



---

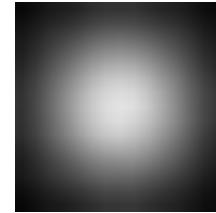
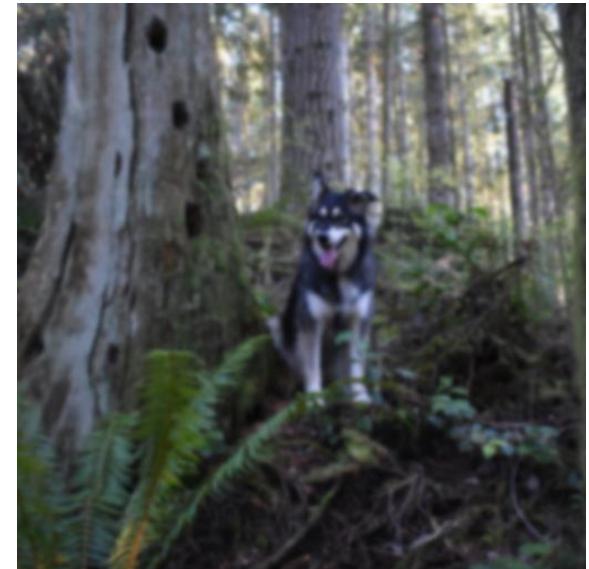
# 2d Gaussian

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



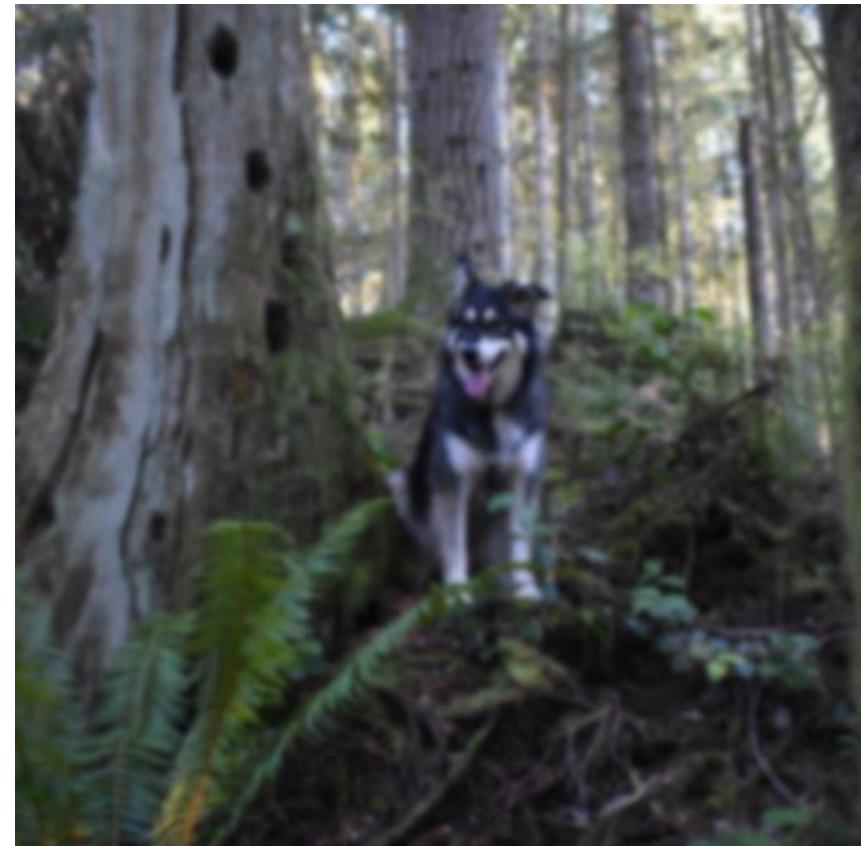
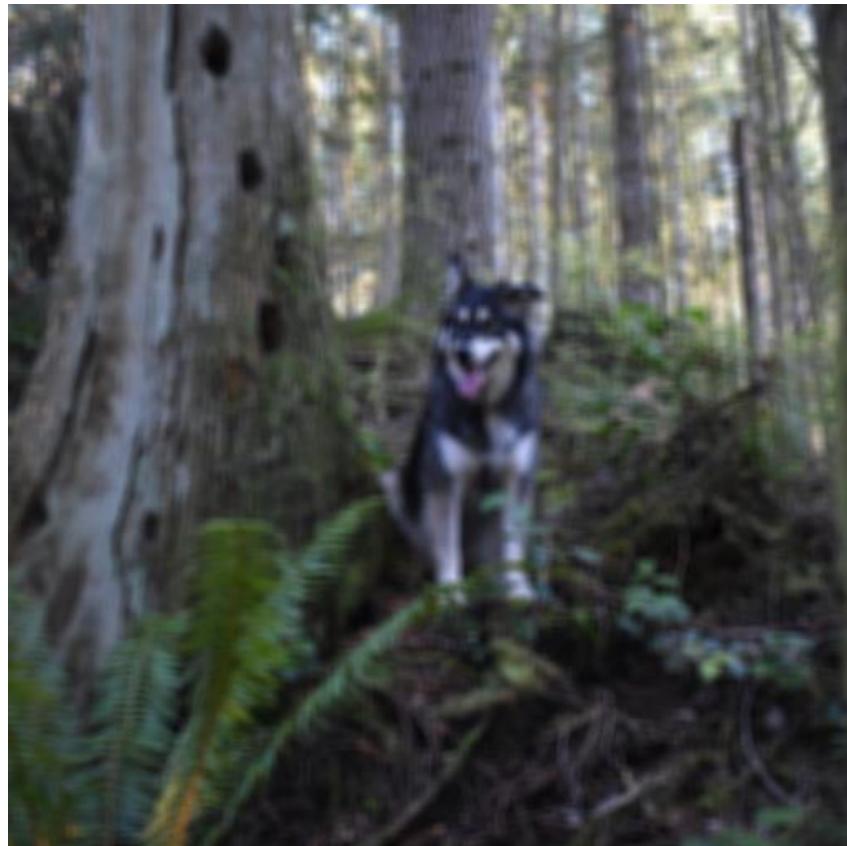
---

# Better smoothing with Gaussians

 $*$  $=$ 

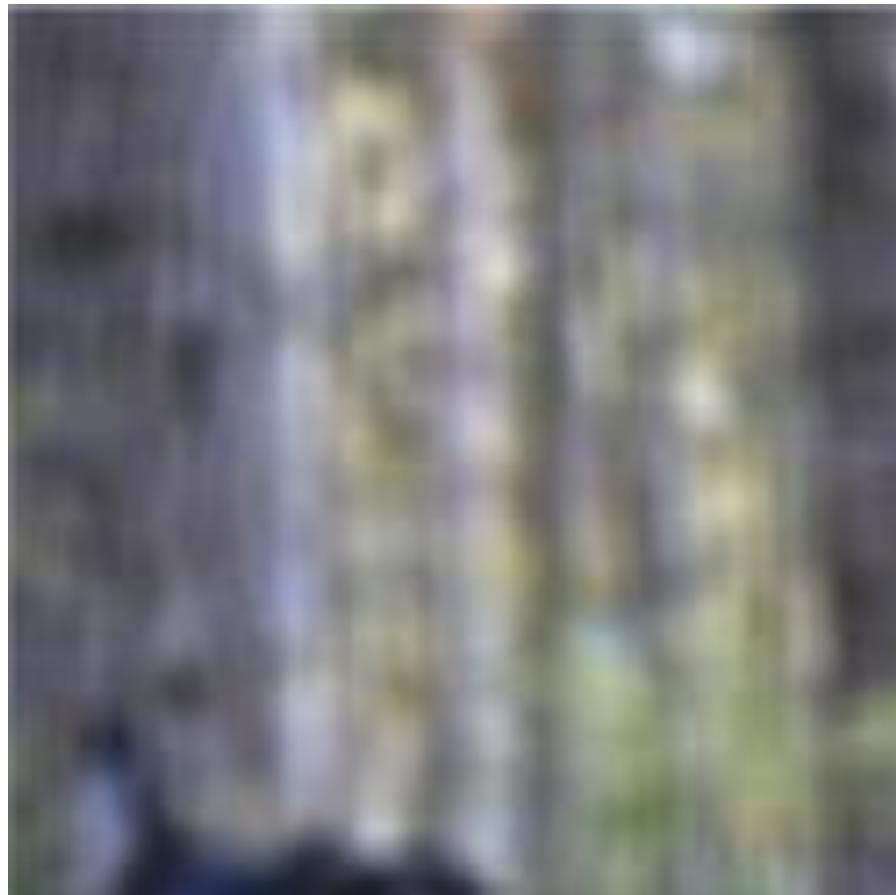
---

# Better smoothing with Gaussians

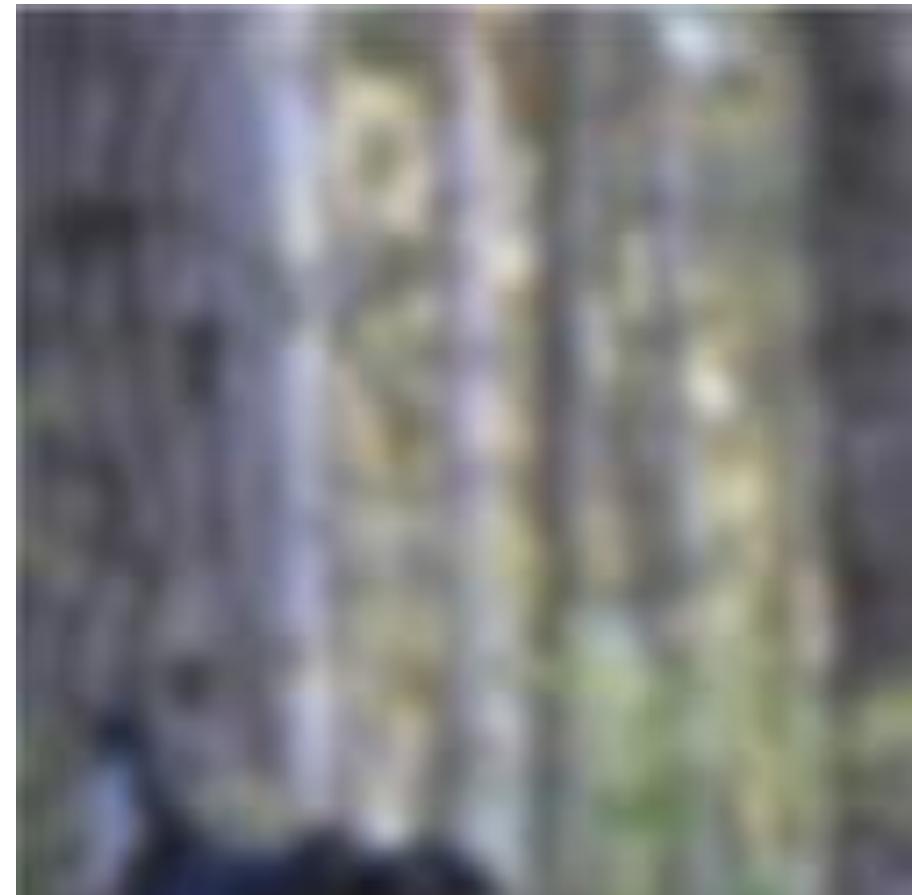


---

# Better smoothing with Gaussians



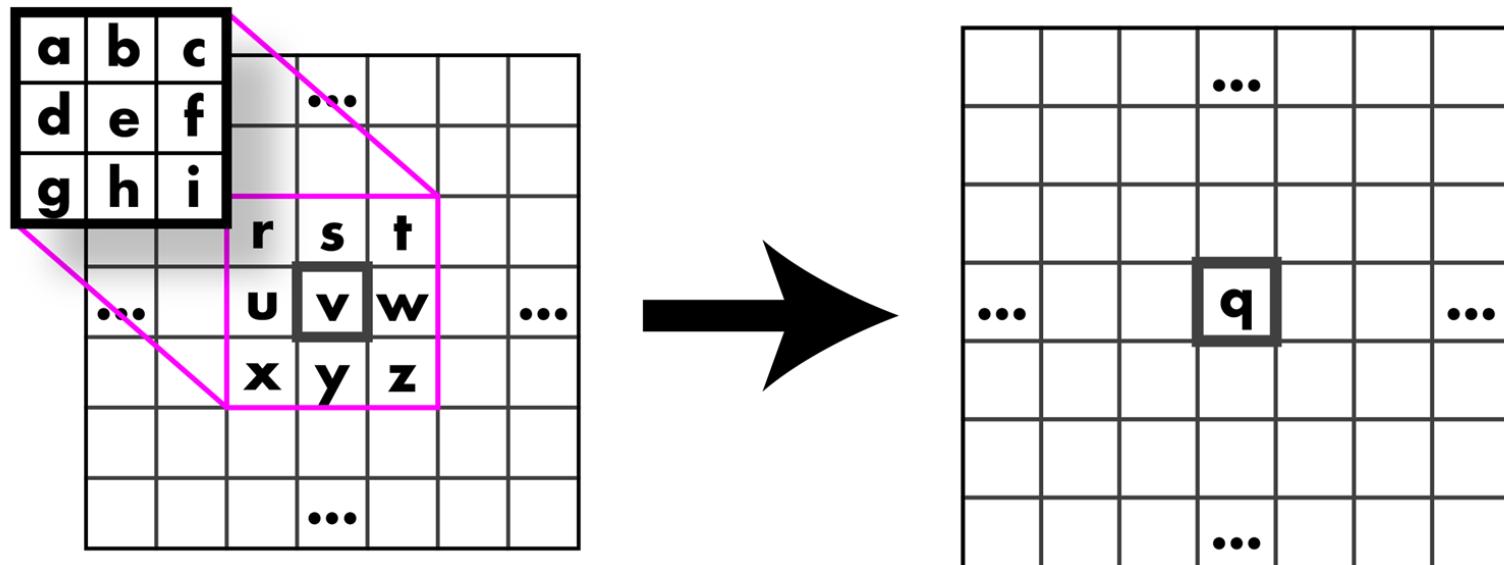
Box Filtered



Gaussian Filtered

---

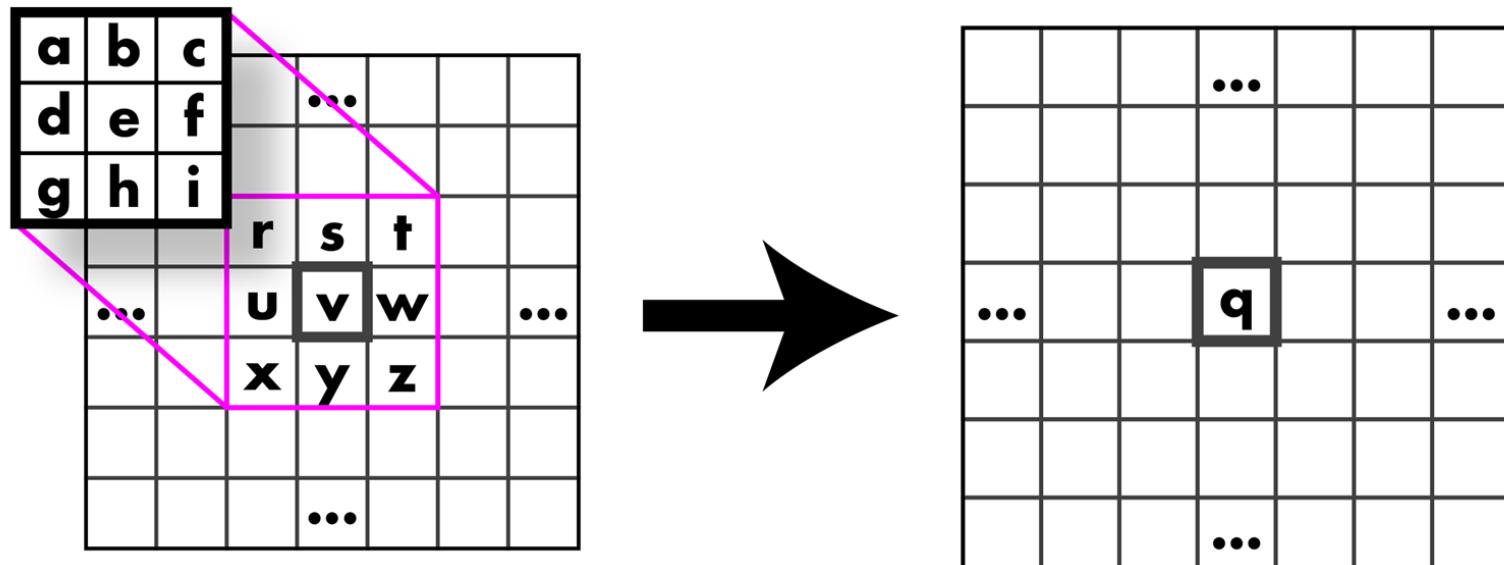
Wow, so what was that convolution thing??



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

---

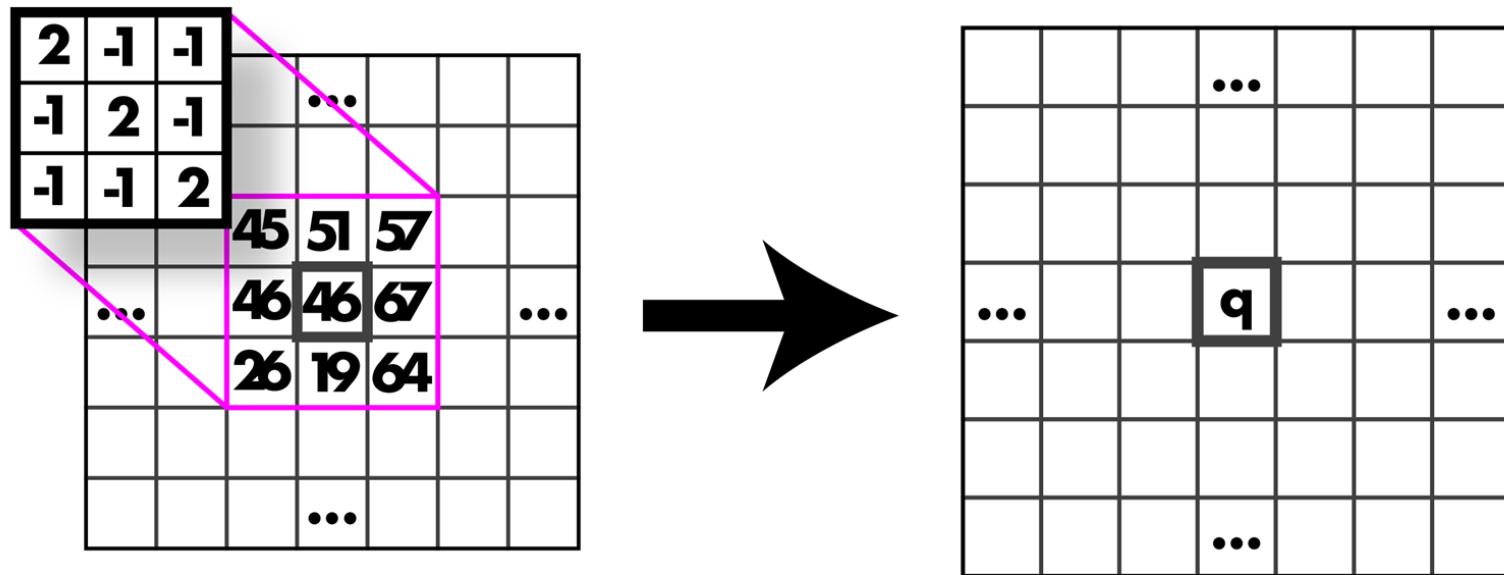
Wow, so what was that convolution thing??



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

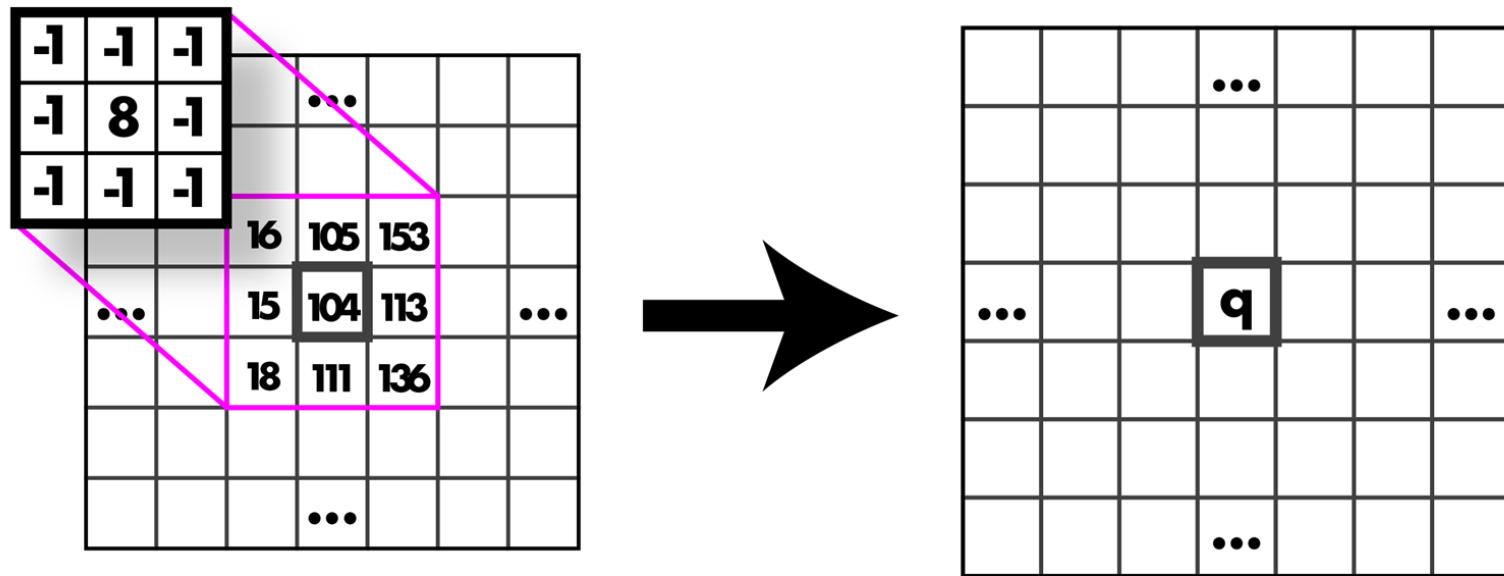
---

Calculate it, go!



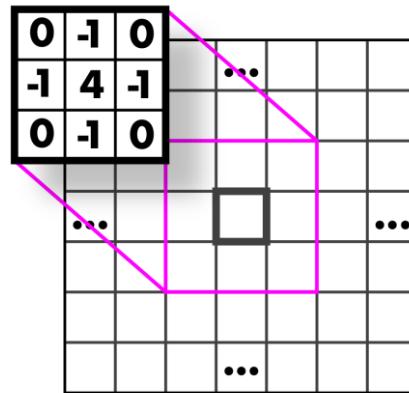
---

Calculate it, go!



---

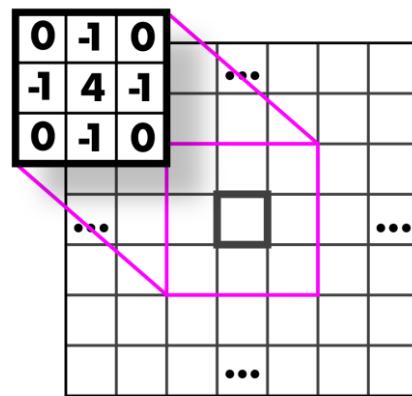
# Guess that kernel!



# Highpass Kernel: finds edges

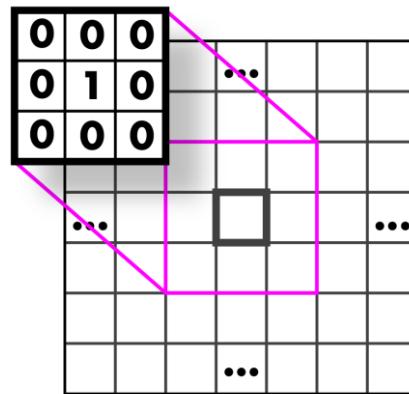
---

## (applied to the graytone image!)



---

# Guess that kernel!



# Identity Kernel: Does nothing!



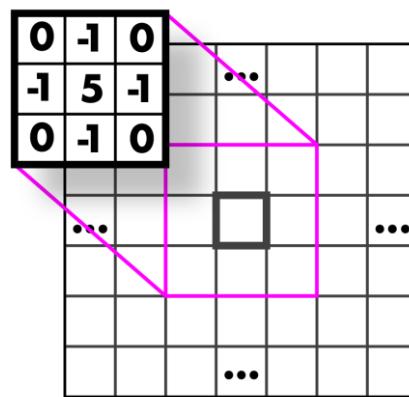
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{matrix} \cdots & & & \\ & \cdots & & \\ & & \cdots & \\ & & & \cdots \end{matrix}$$

A 3x3 matrix with the top-left element as 1 and all other elements as 0. To its right is a 5x5 grid with a 3x3 identity matrix in the top-left corner. A pink line connects the two, illustrating the convolution operation with a 3x3 kernel.



---

# Guess that kernel!



# Sharpen Kernel: sharpens!

---

## (applied to all three bands)



$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

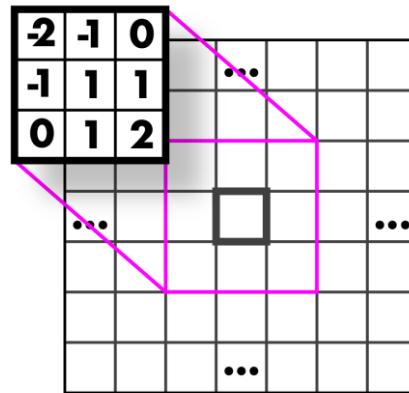
A diagram illustrating the application of a sharpening kernel. On the left is a 3x3 kernel matrix. A 5x5 input grid is shown to its right, with a central 3x3 subgrid highlighted in black. A pink line connects the top-left cell of the kernel to the top-left cell of the input grid, indicating the receptive field of that kernel cell. Ellipses in the kernel matrix and input grid indicate that the pattern repeats across the entire image.



Note: sharpen = highpass + identity!

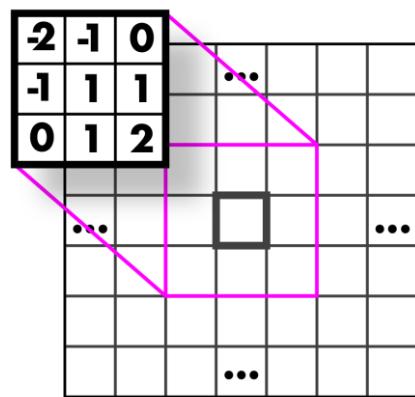
---

# Guess that kernel!

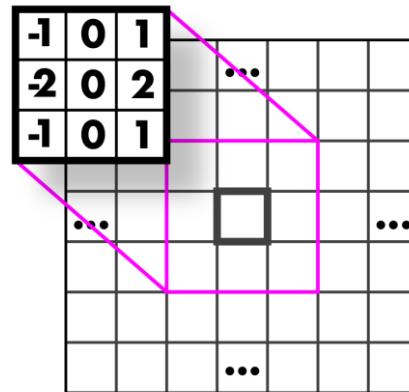
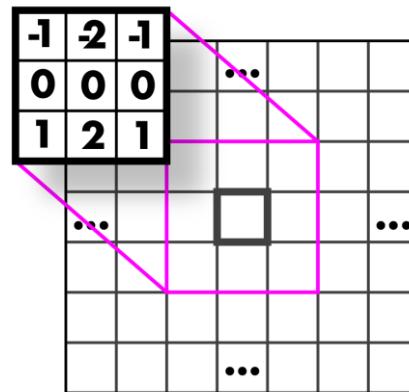


# Emboss Kernel: stylin' (applied to all three bands)

---

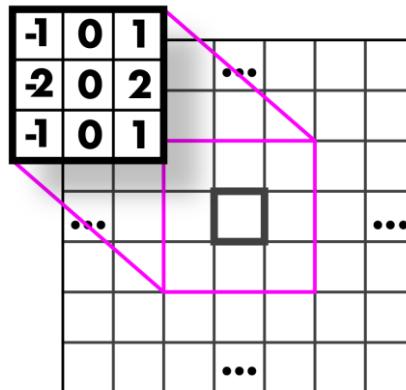
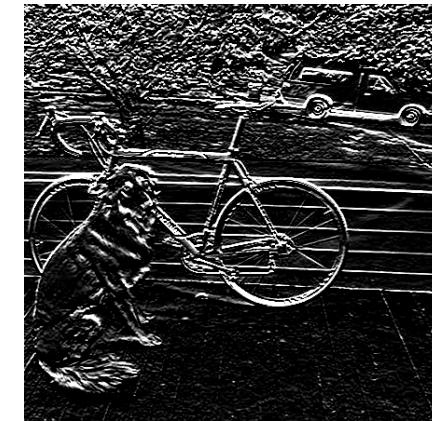
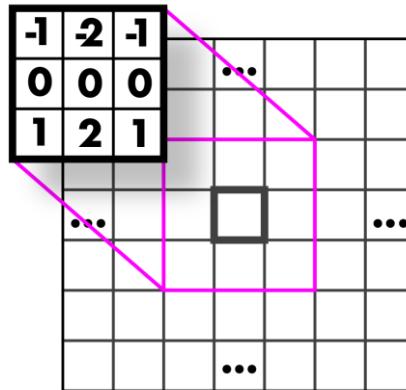


# Guess those kernels!



# Sobel Kernels: edges (applied to a graytone image and thresholded)

---



# Sobel Kernels: edges and gradient!



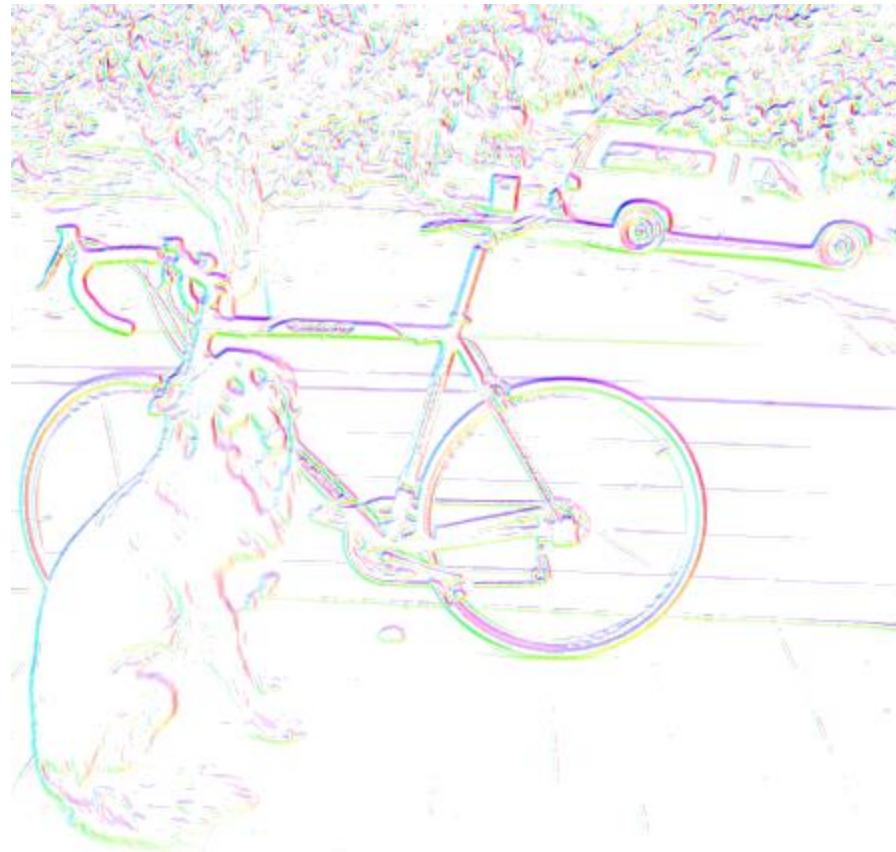
$$\begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



---

# Sobel Kernels: edges and gradient!



This visualization is showing the magnitude and direction of the gradient. We will talk further about this when we discuss edges.

---

And so much more!!

# Assignment 2

Image resizing and filtering

# First things first!

- First, you need to run `git pull` from inside your homeworks folder to get the latest changes from GitHub.
- Remember that you might need some of your code from the previous hw (e.g. `set_pixel`) for this hw as well. Have your code from hw1 in your src folder.
- Then run:
  - `make clean`
  - `make`

# Assignment 2

## Nearest Neighbor Interpolation and Resizing

- `float nn_interpolate(image im, float x, float y, int c);` in `src/modify_image.c`
- `image nn_resize(image im, int w, int h);`

## Bilinear Interpolation and Resizing

- `float bilinear_interpolate(image im, float x, float y, int c);`
- `image bilinear_resize(image im, int w, int h);`

# Assignment 2

## Box Filter

- void l1\_normalize(image im)
- image make\_box\_filter(int w)

## Convolution

- image convolve\_image(image im, image filter, int preserve)
- image make\_highpass\_filter()
- image make\_sharpen\_filter()
- image make\_emboss\_filter()

# Assignment 2

## Gaussian

- image make\_gaussian\_filter(float sigma)

## Hybrid Images

- image add\_image(image a, image b)
- image sub\_image(image a, image b)

# Assignment 2

## Sobel Operator (next lecture)

- `image make_gx_filter()`
- `image make_gy_filter()`
- `void feature_normalize(image im)`
- `image *sobel_image(image im)`
- `image colorize_sobel(image im)`