Computer Vision

CSE 455
Filters

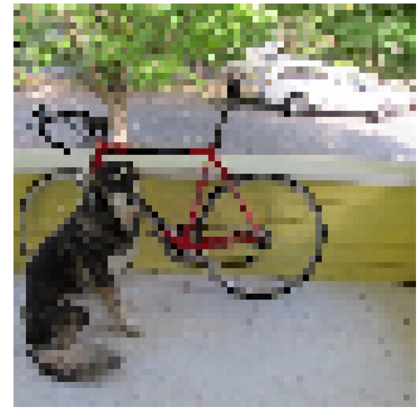
Linda Shapiro
Professor of Computer Science & Engineering
Professor of Electrical Engineering

Let’s do something interesting already!!

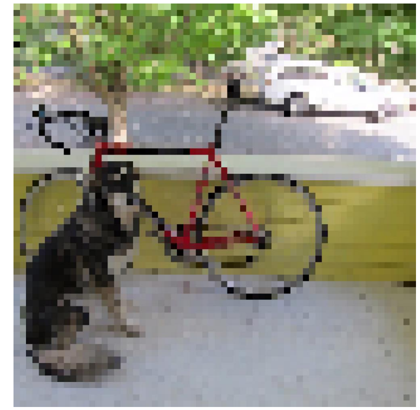
Want to make image smaller

448x448 -> 64x64

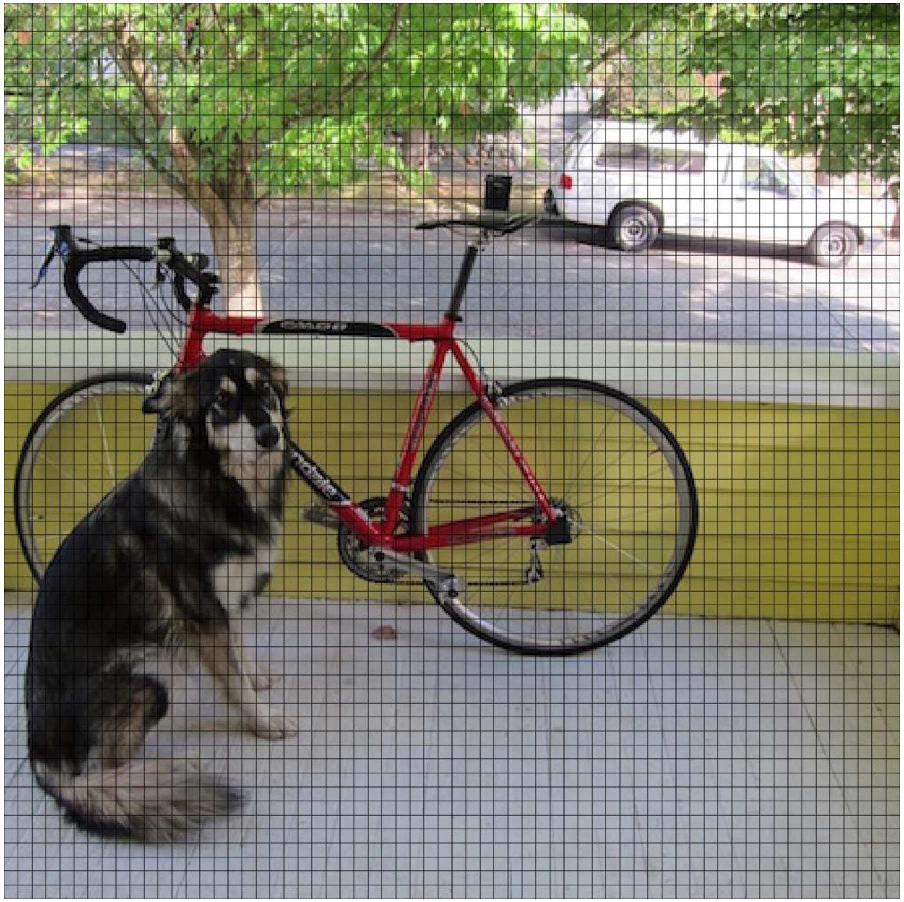
448x448 -> 64x64



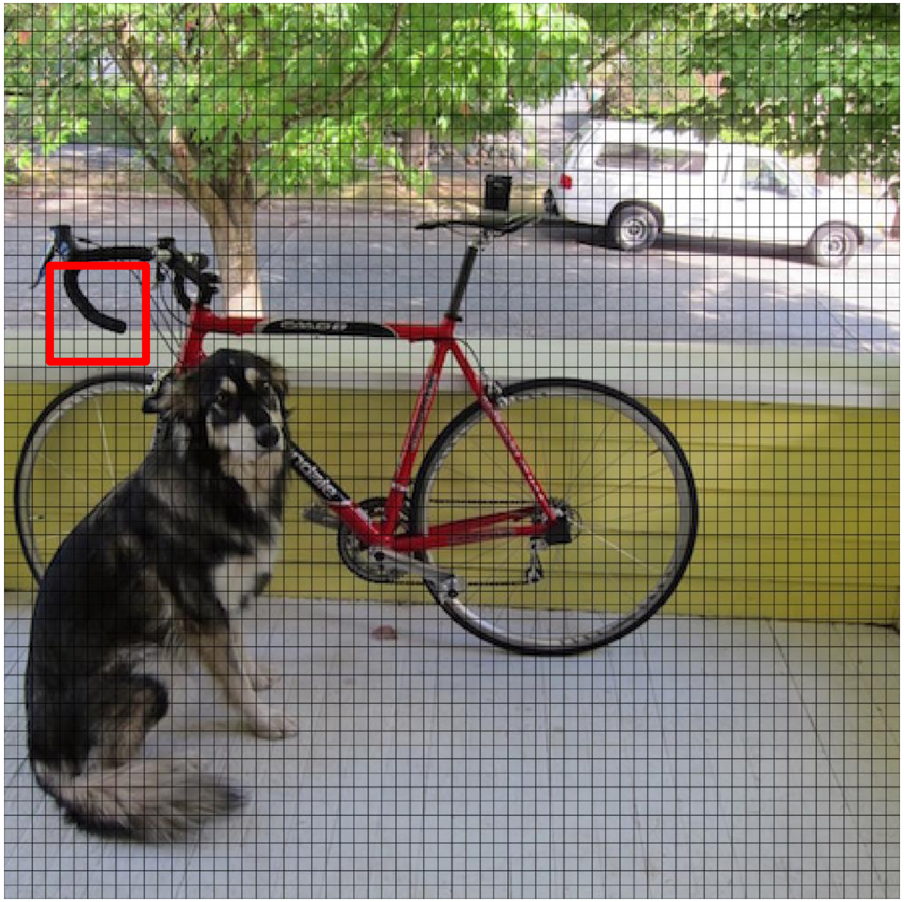
448x448 -> 64x64

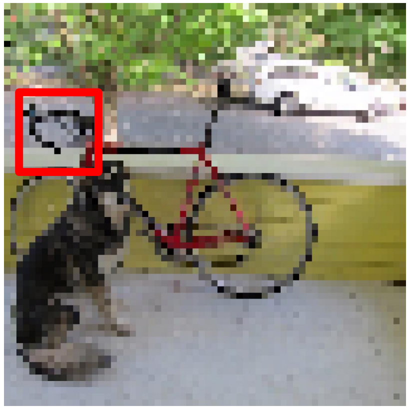


448x448 -> 64x64

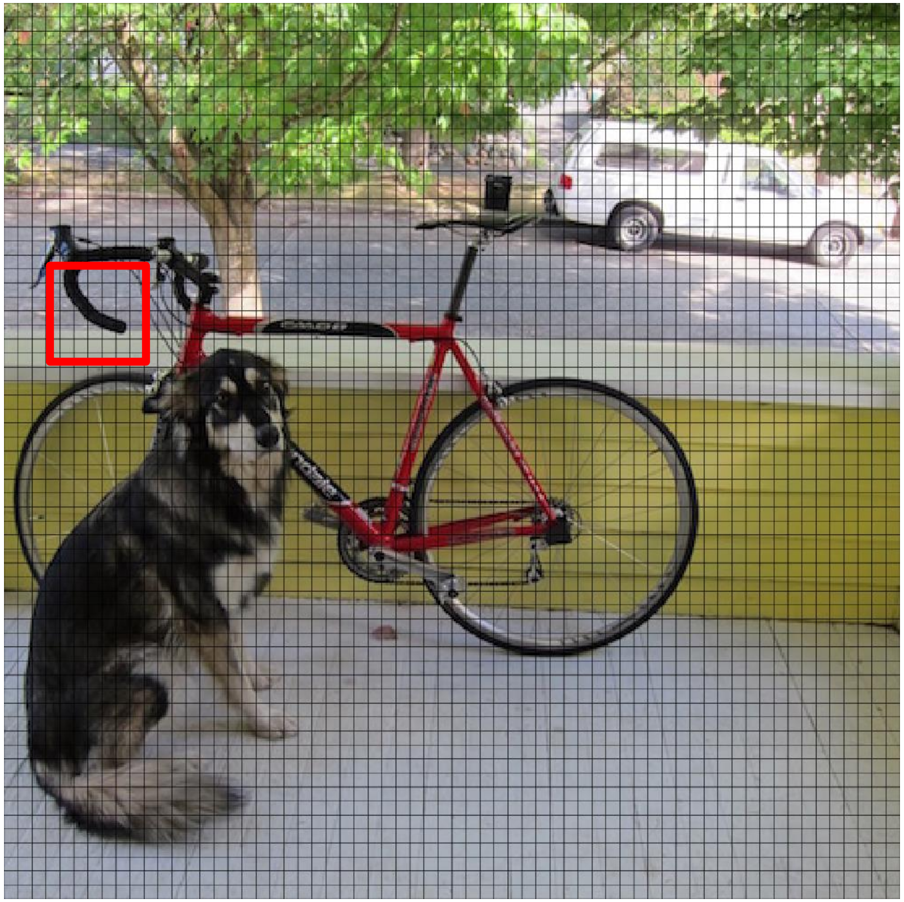


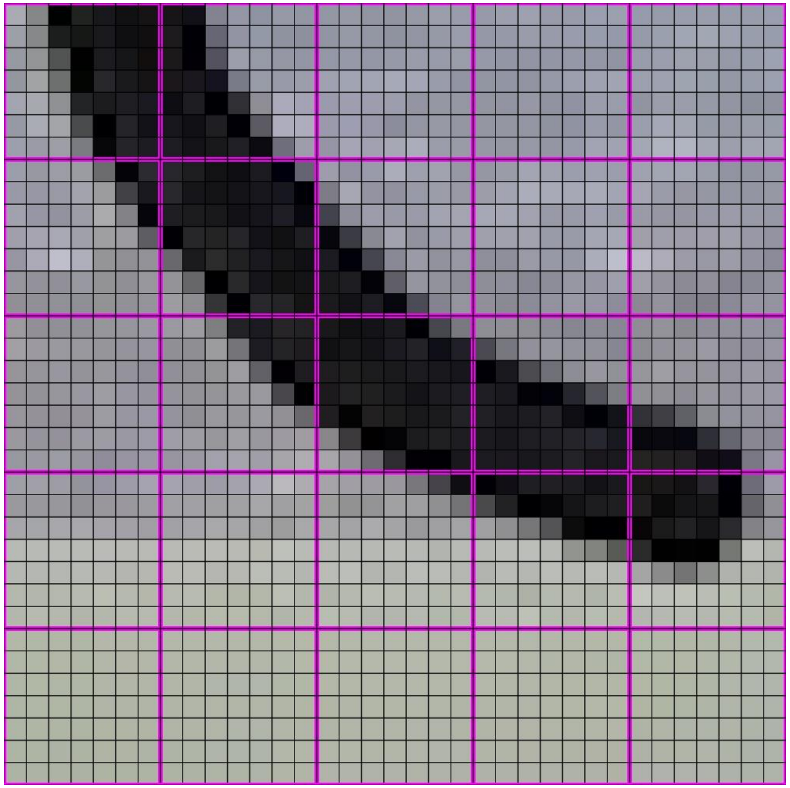
448x448 -> 64x64



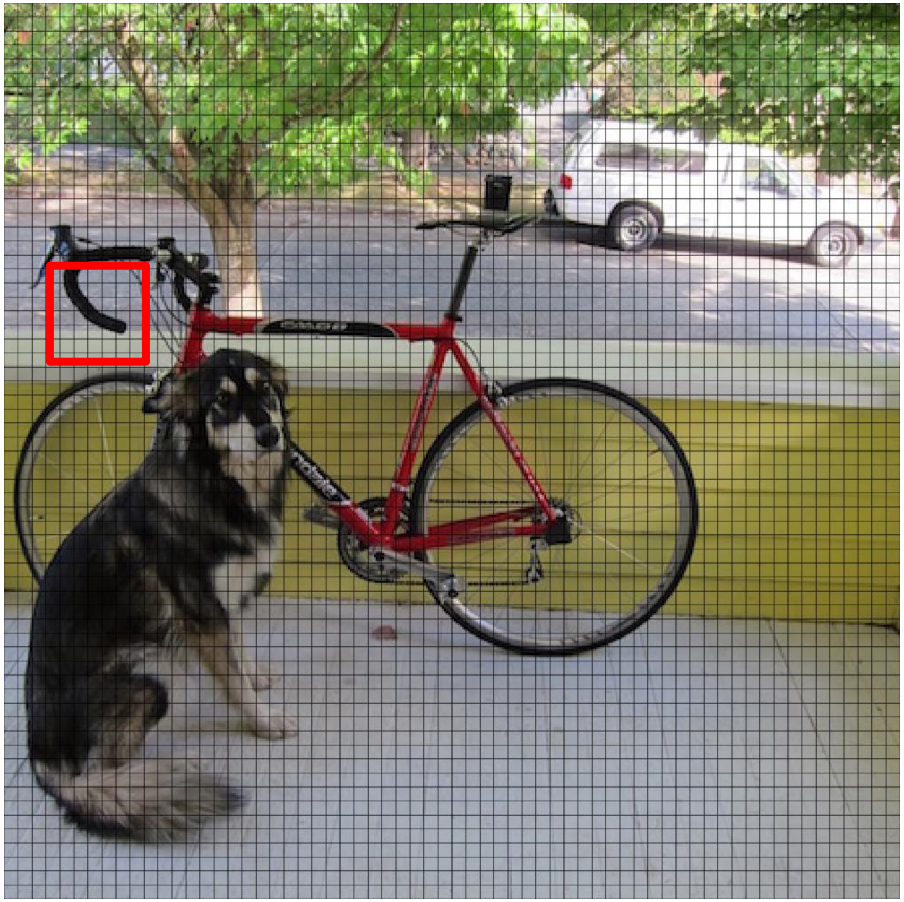


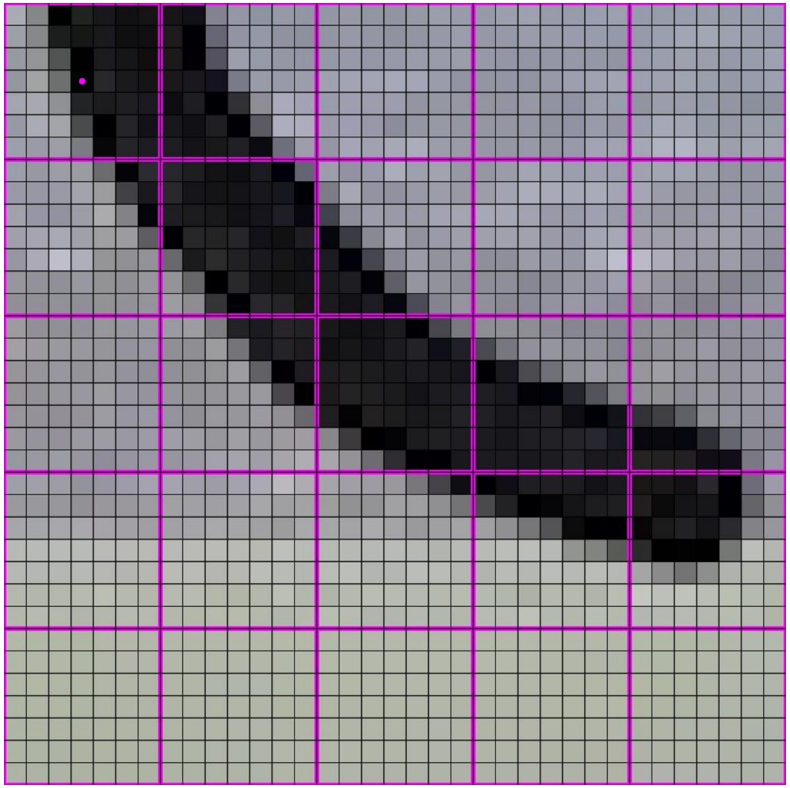
448x448 -> 64x64



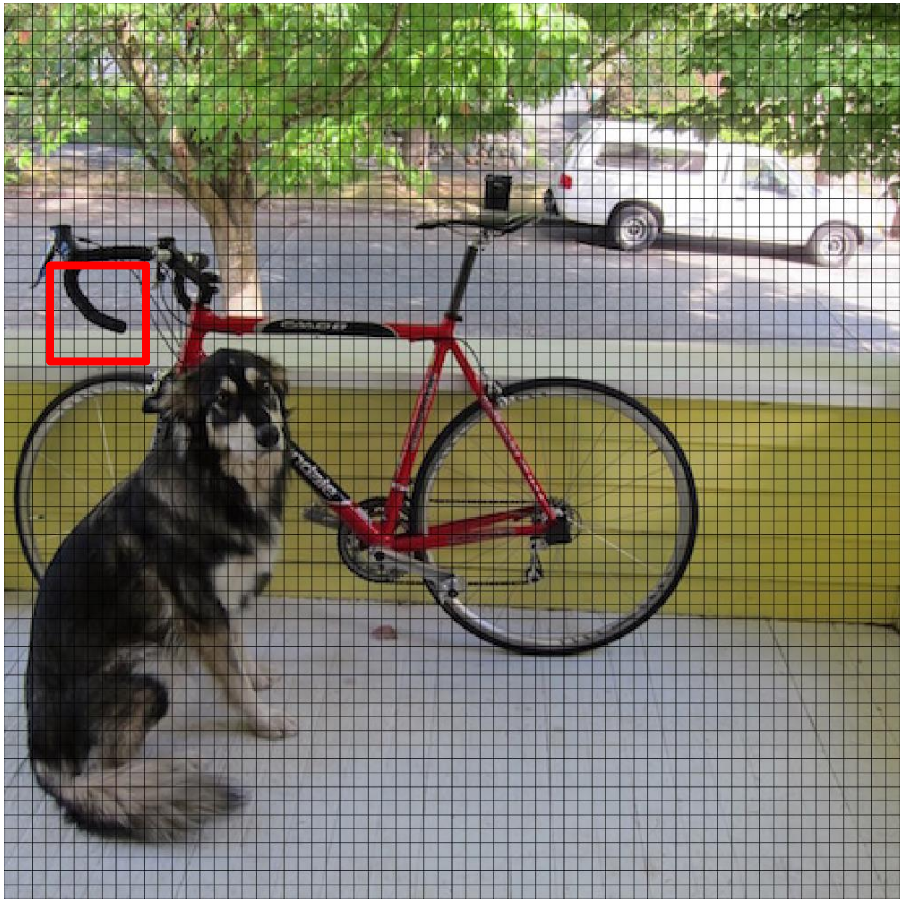


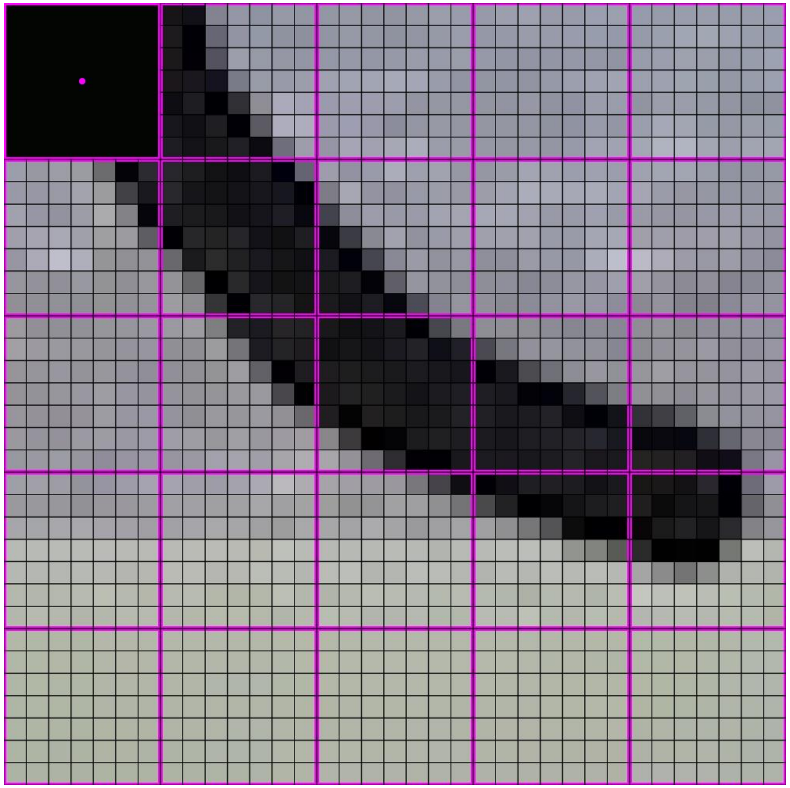
448x448 -> 64x64



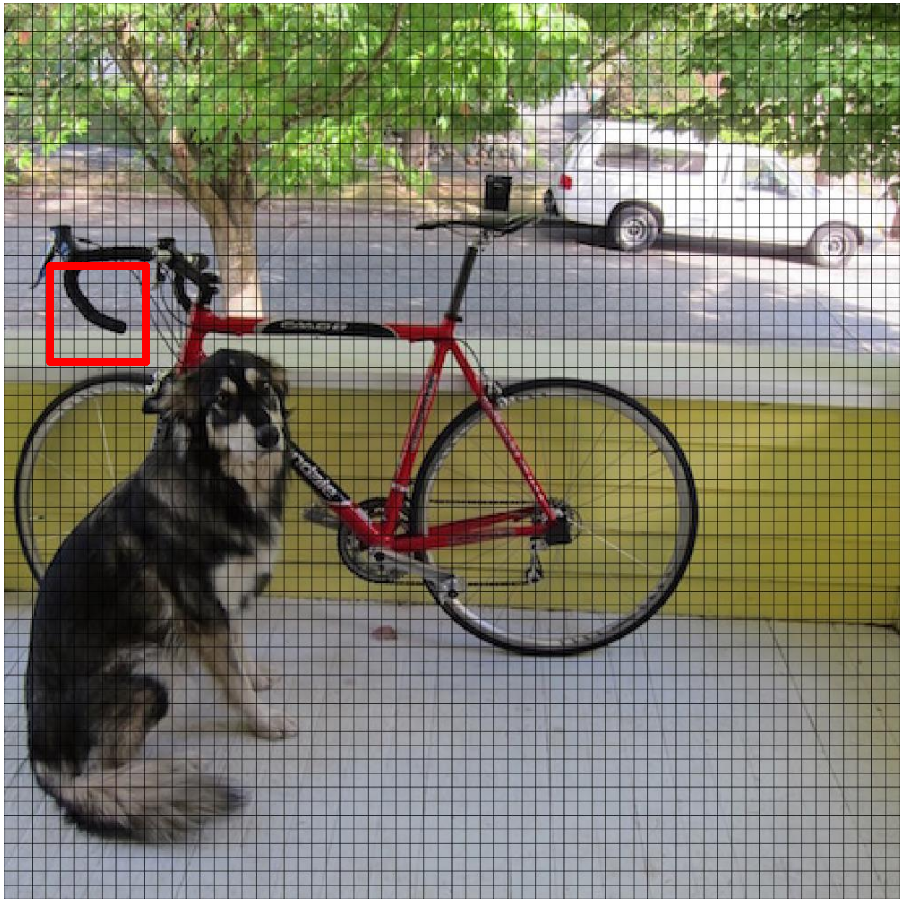


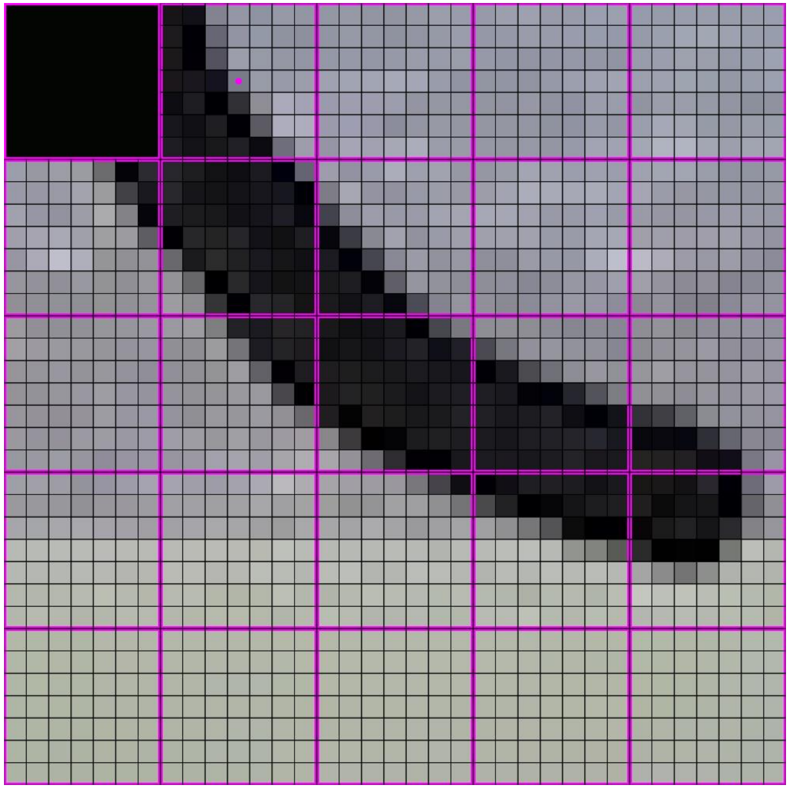
448x448 -> 64x64



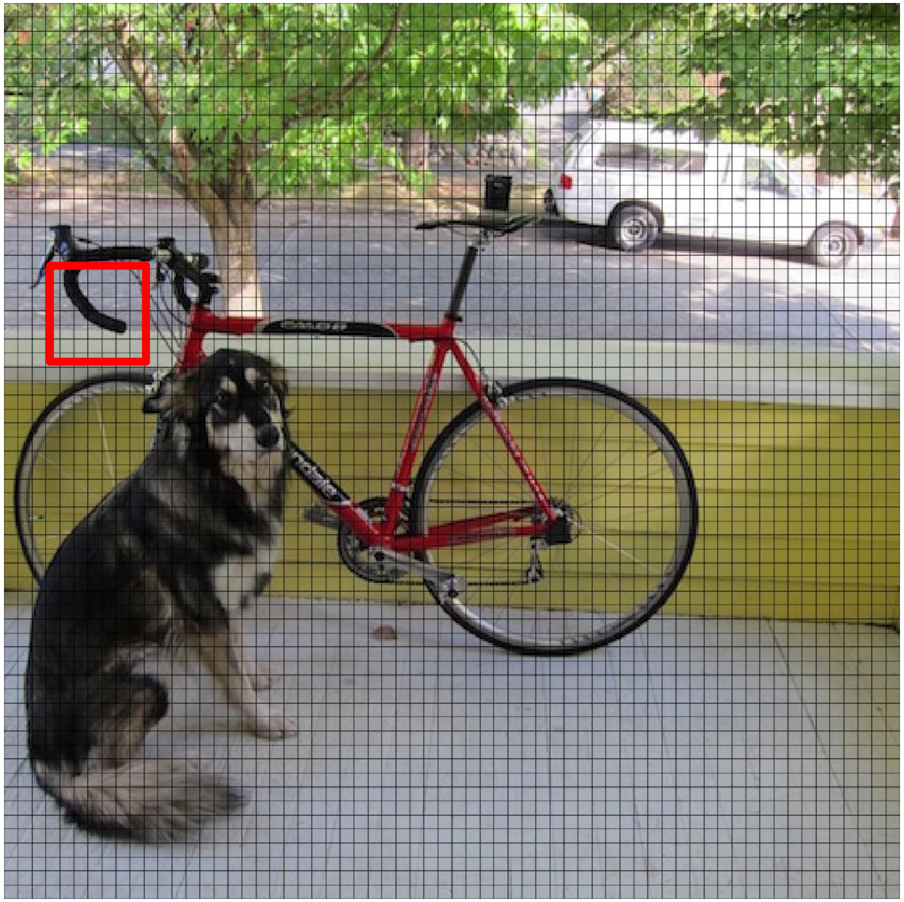


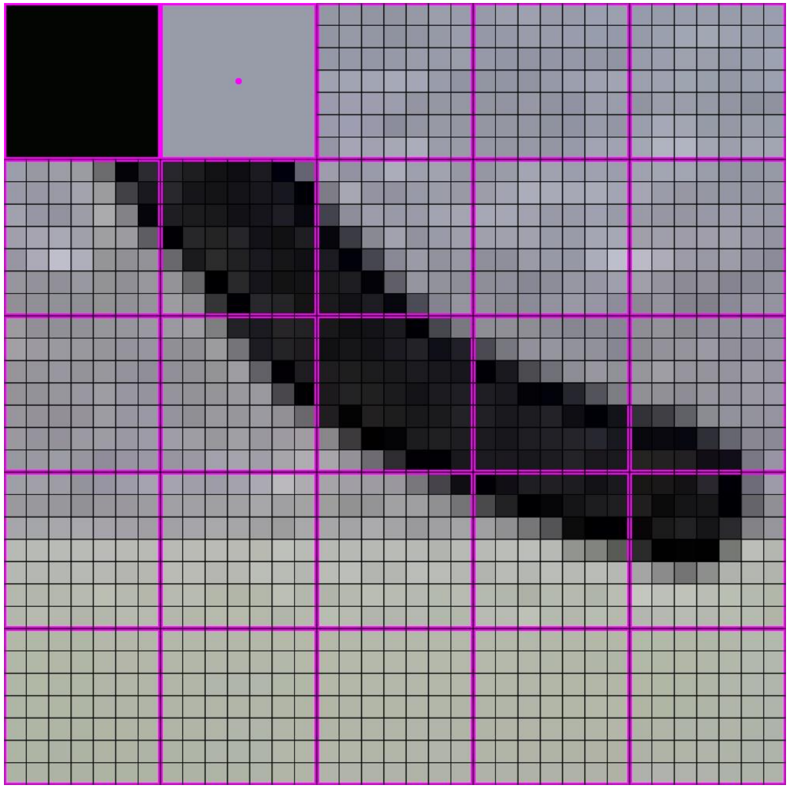
448x448 -> 64x64



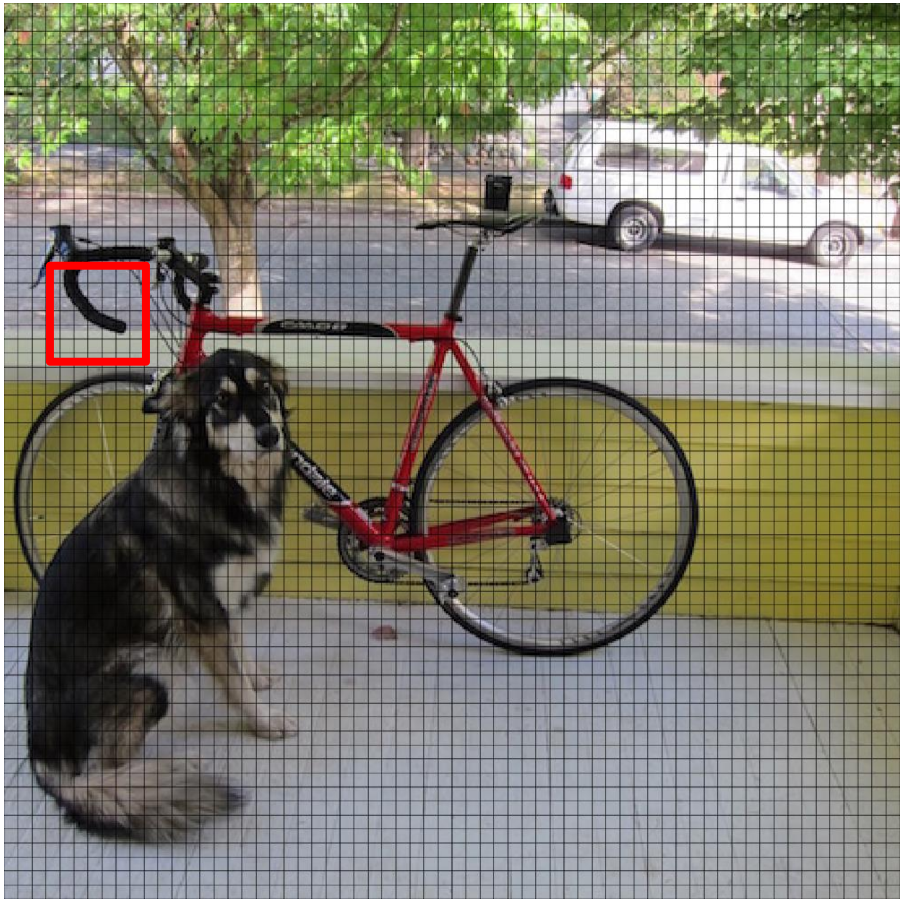


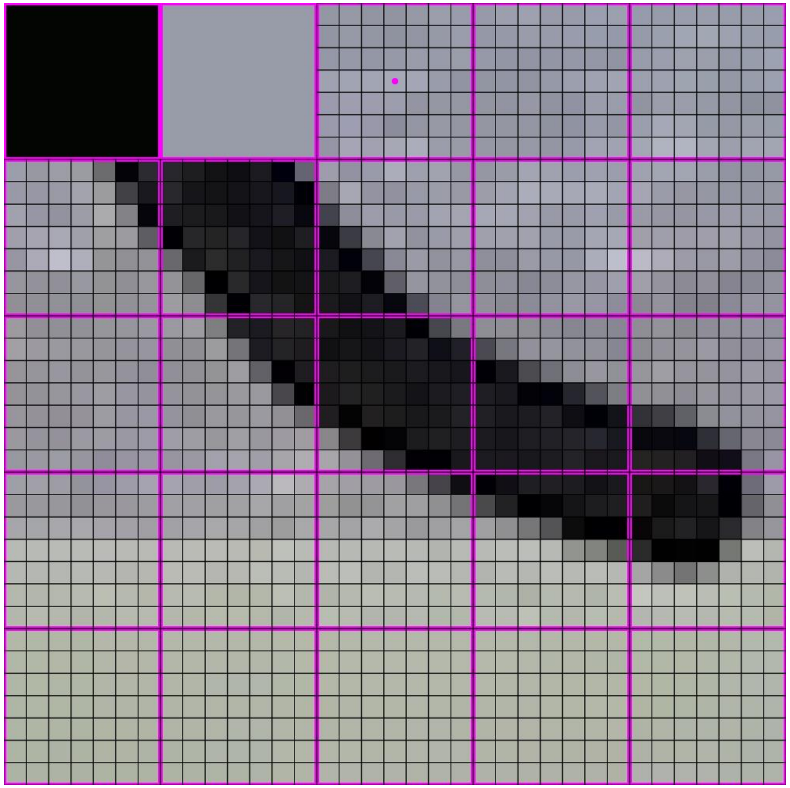
448x448 -> 64x64



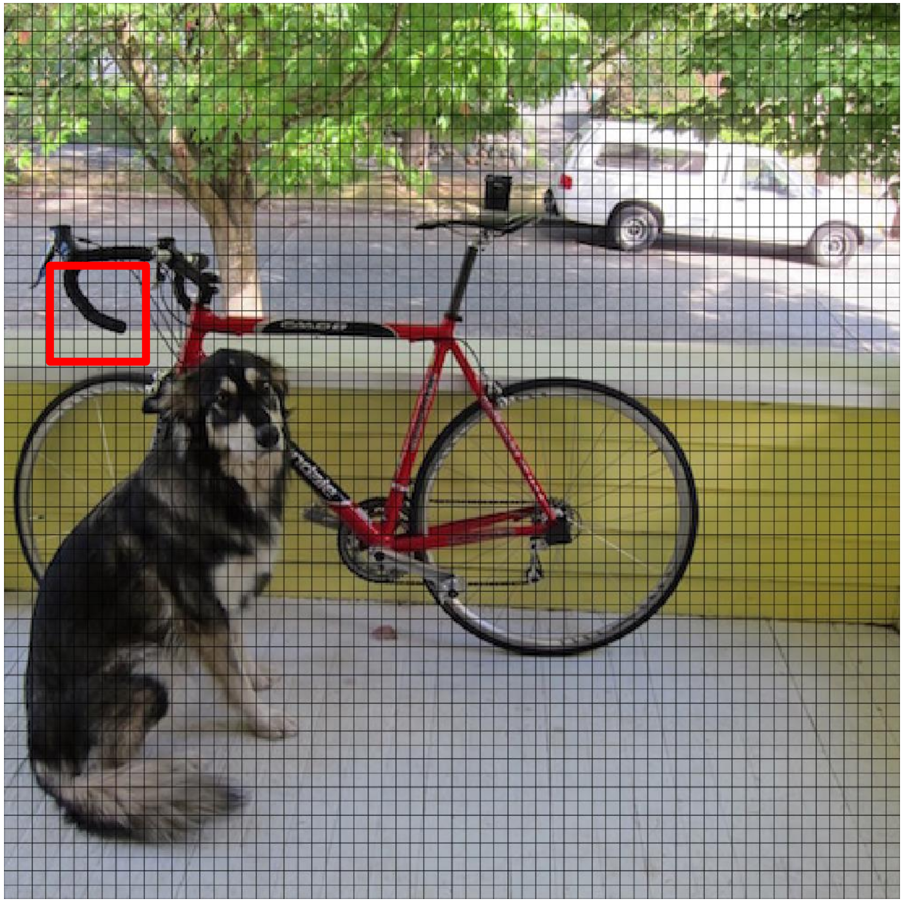


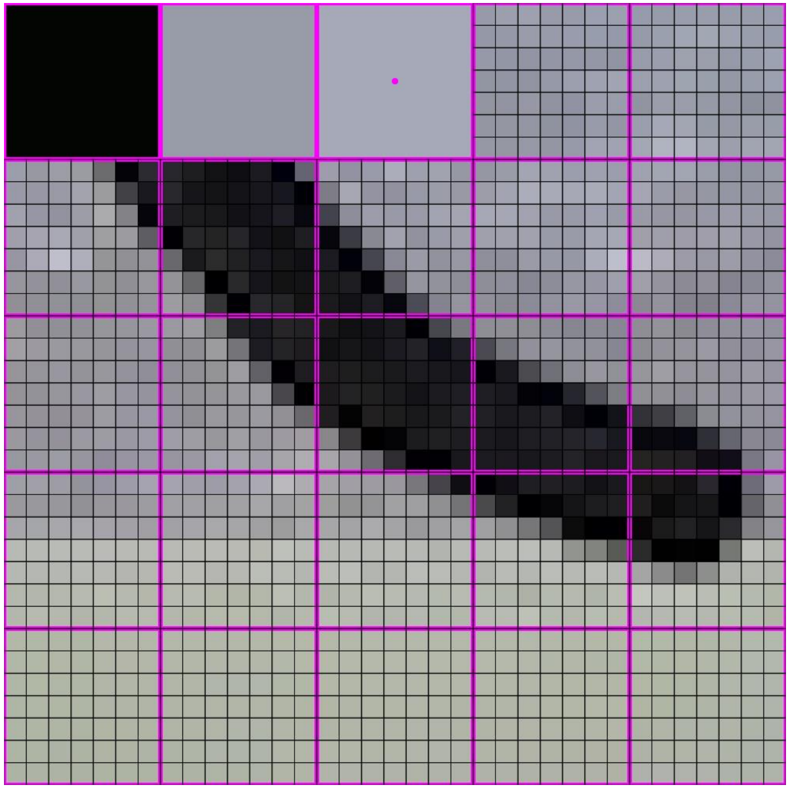
448x448 -> 64x64



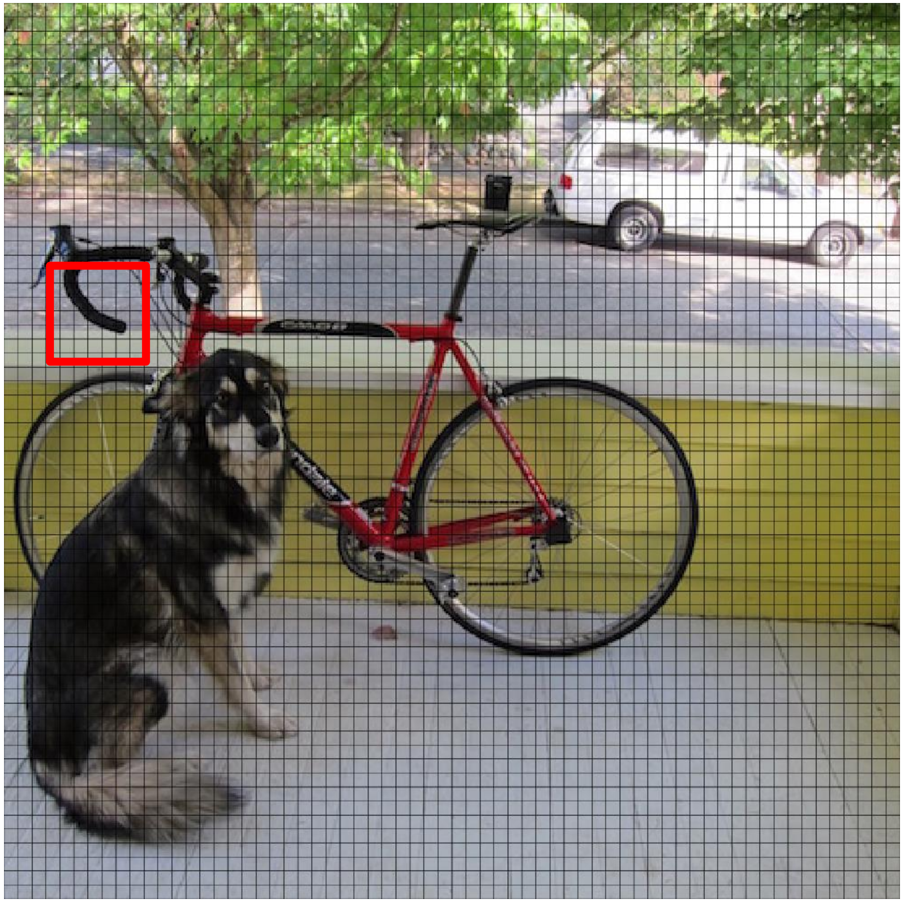


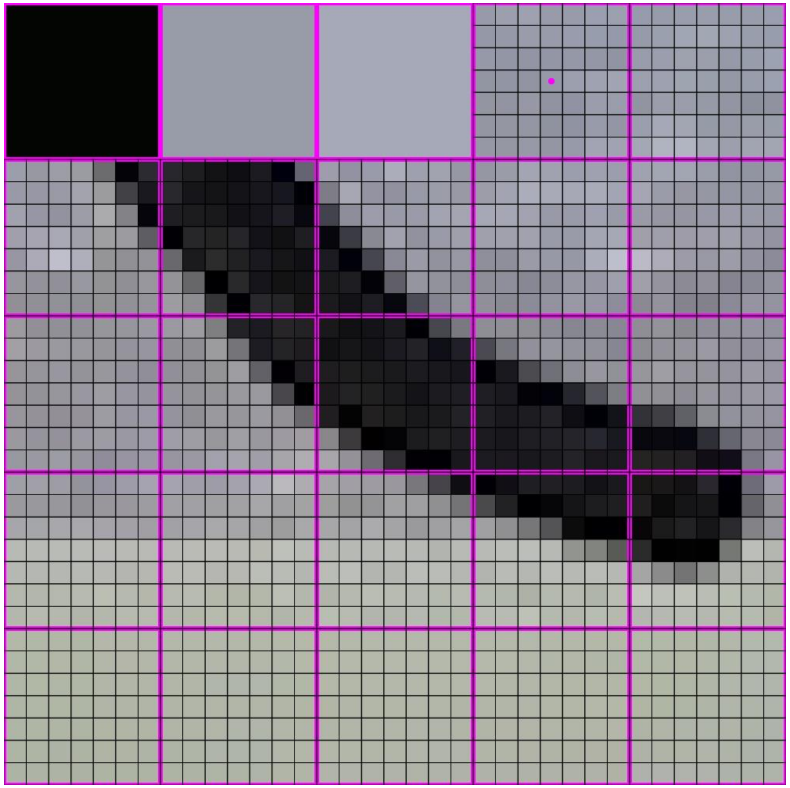
448x448 -> 64x64



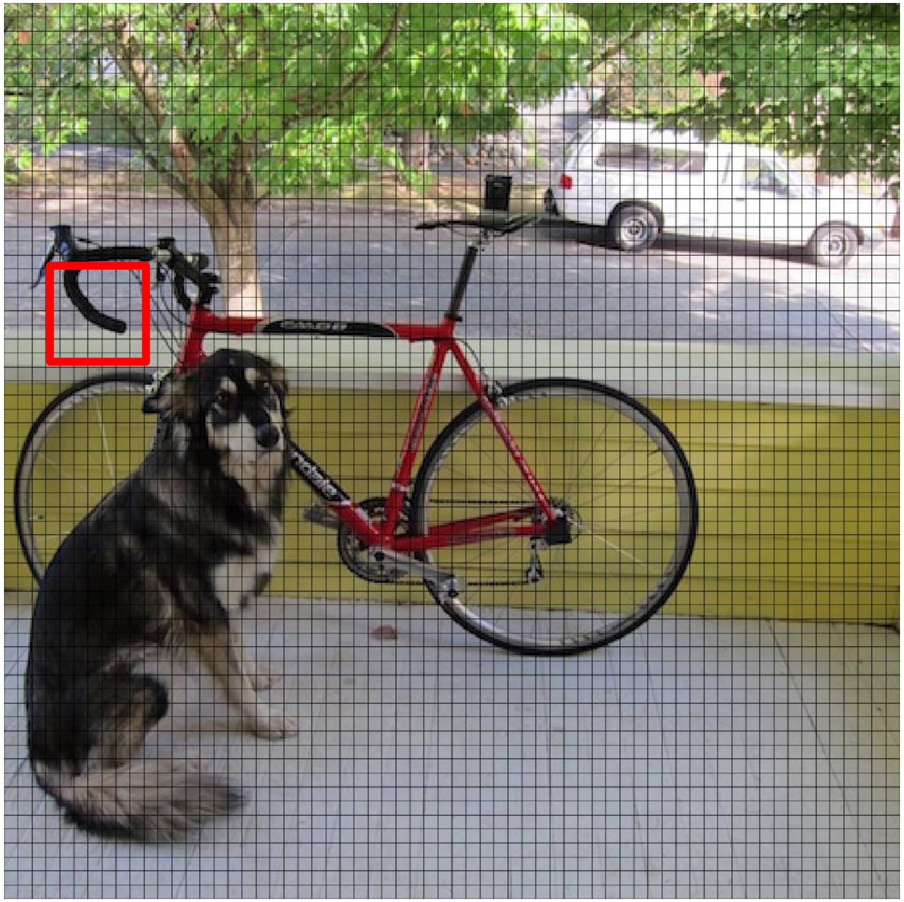


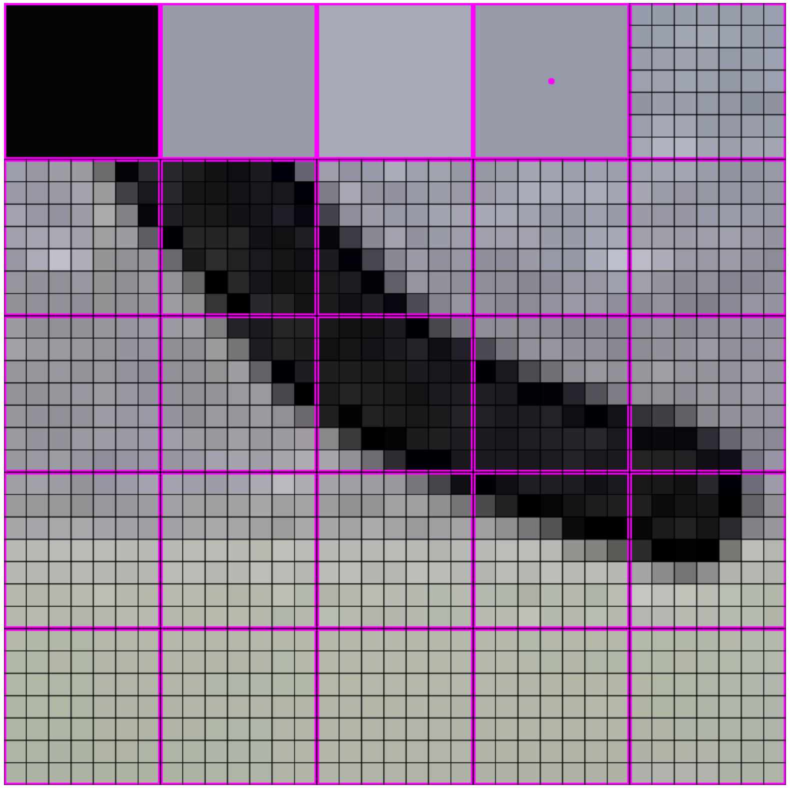
448x448 -> 64x64



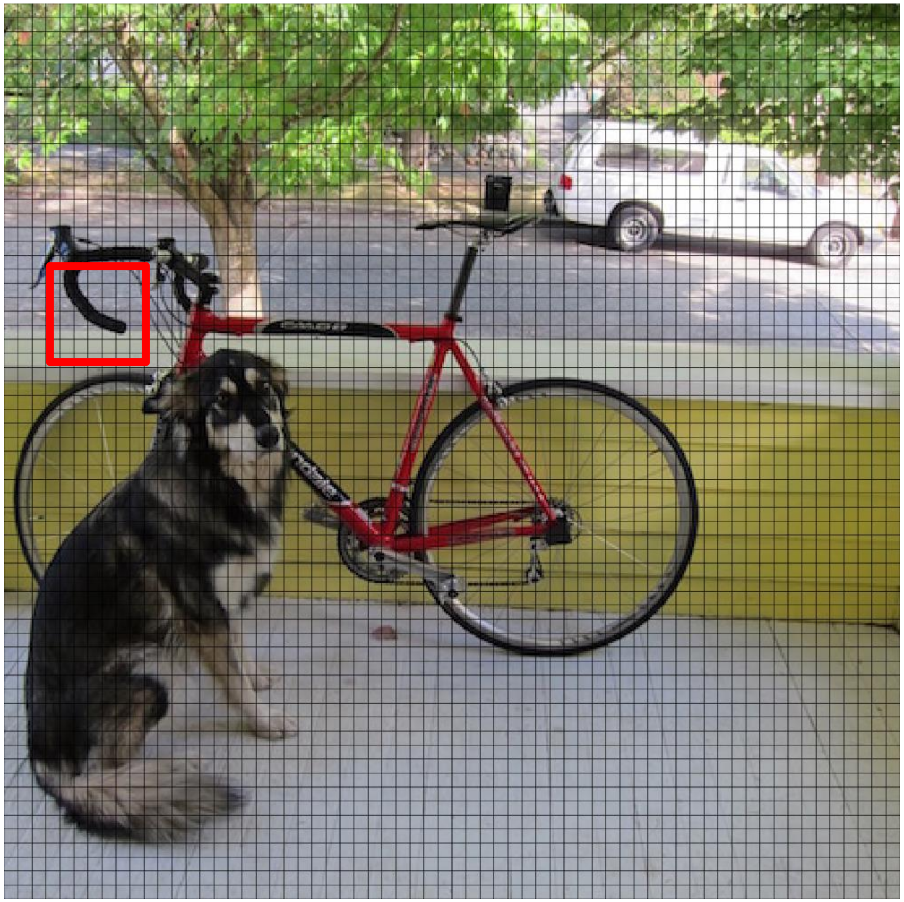


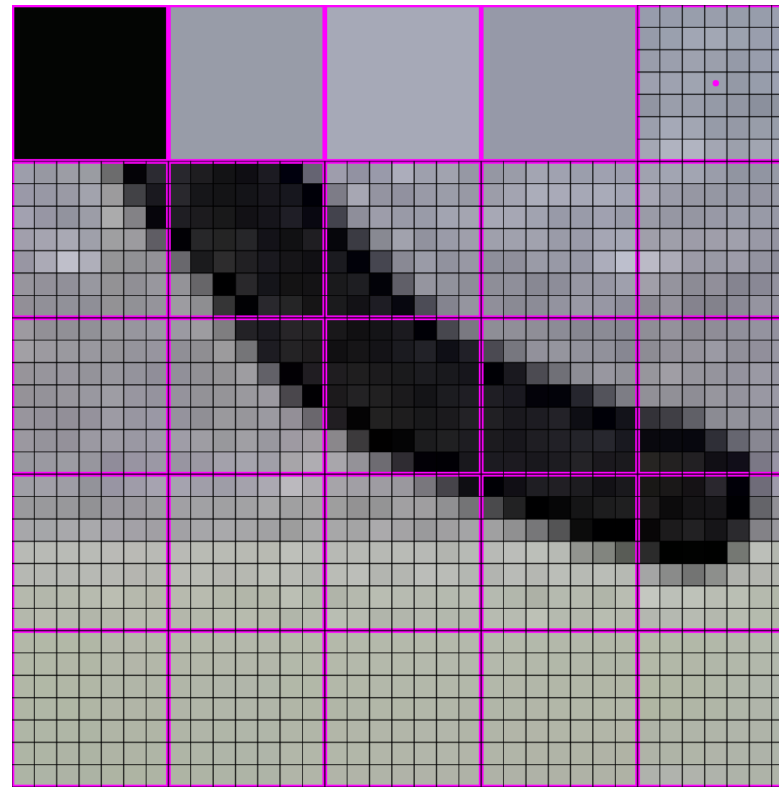
448x448 -> 64x64



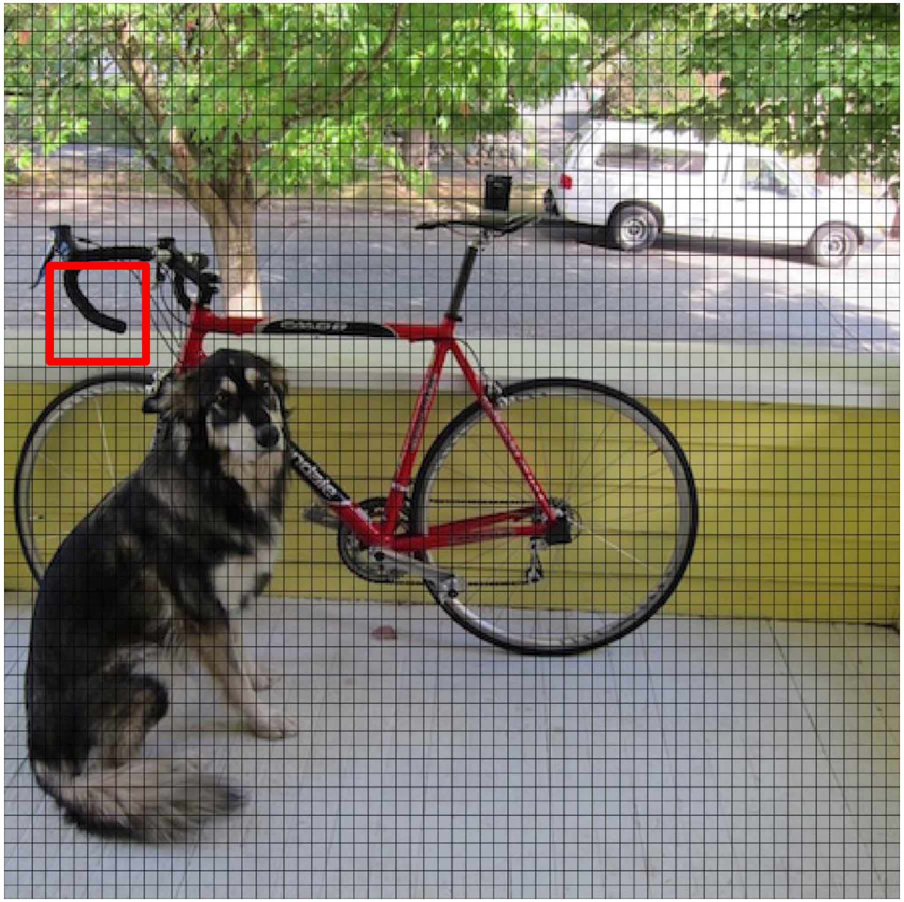


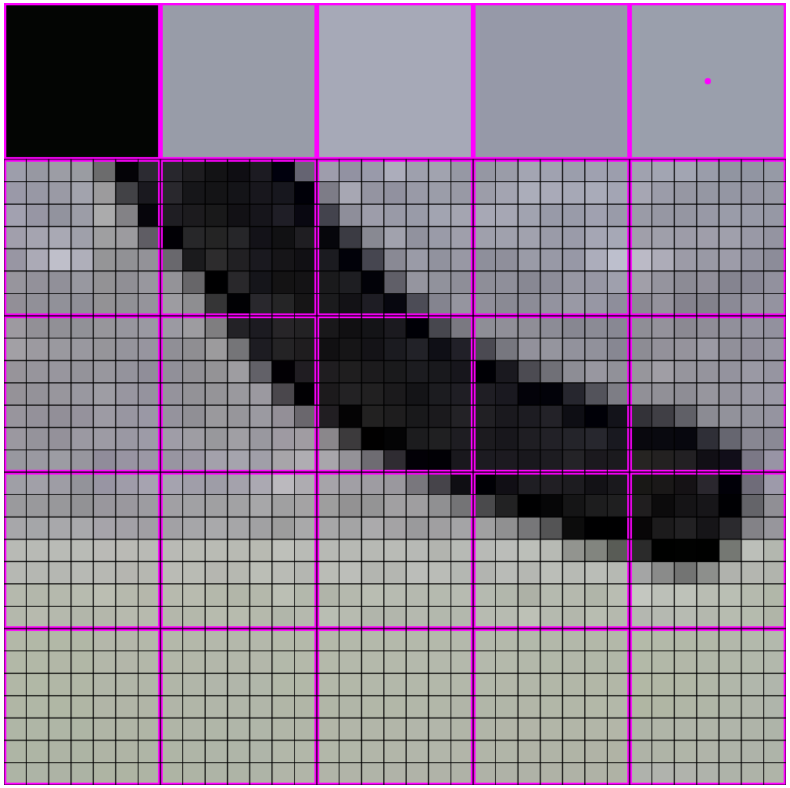
448x448 -> 64x64



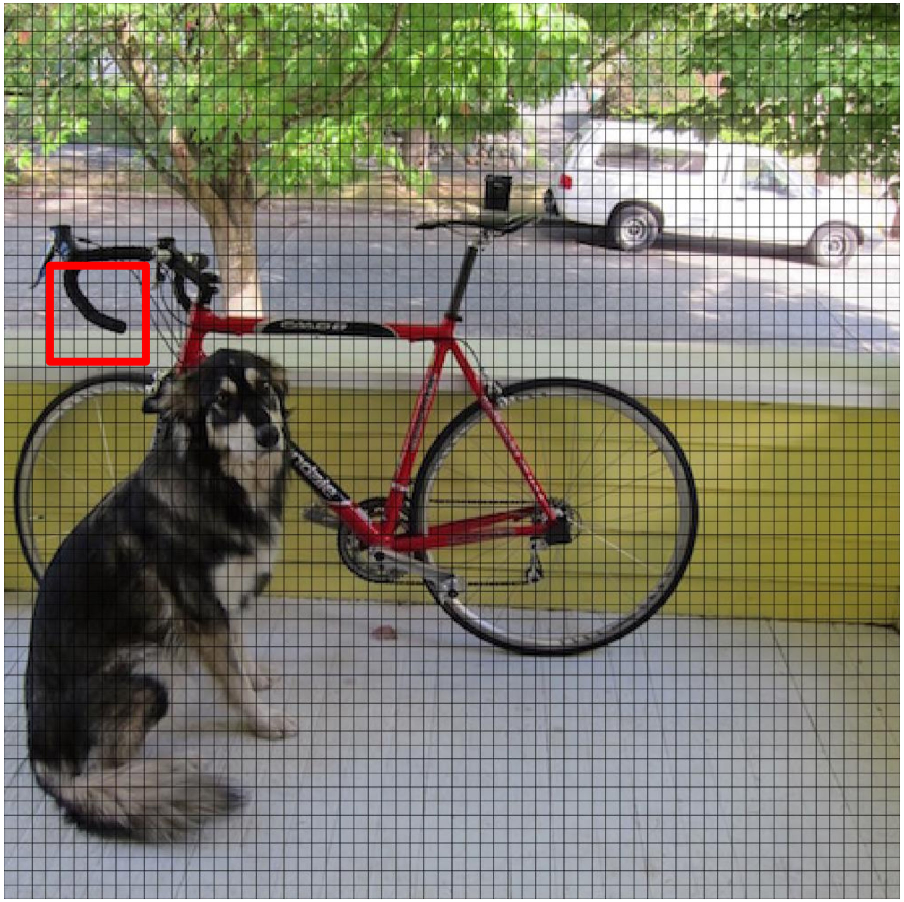


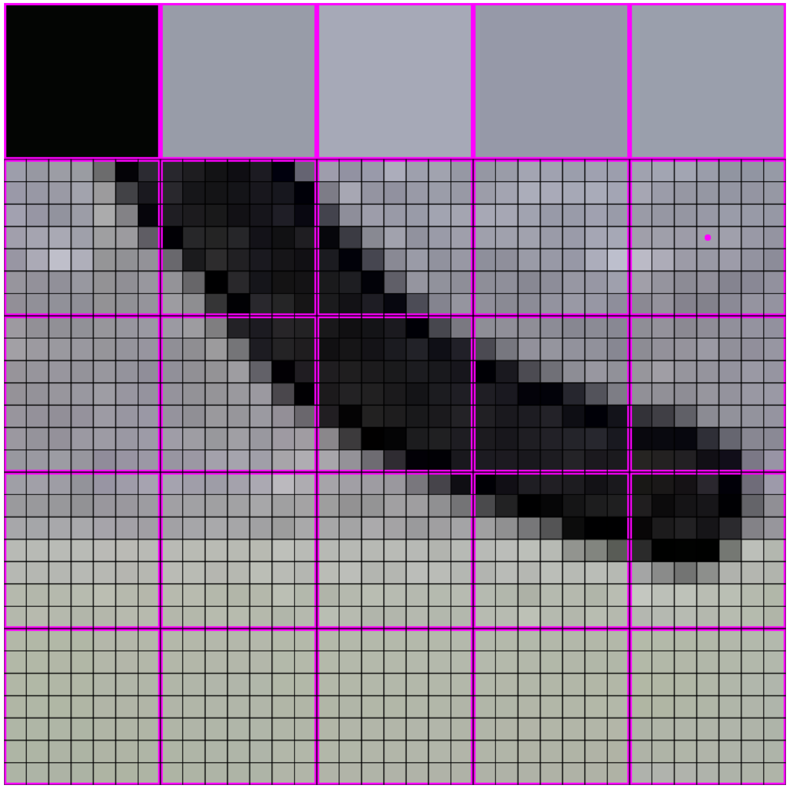
448x448 -> 64x64



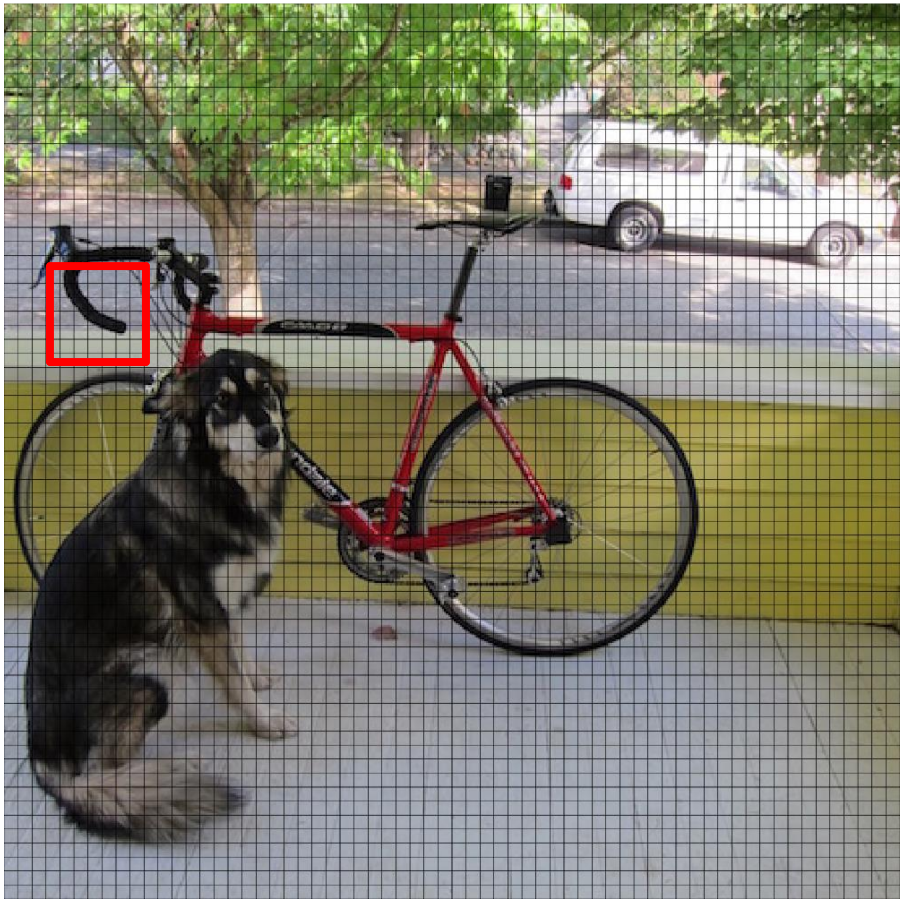


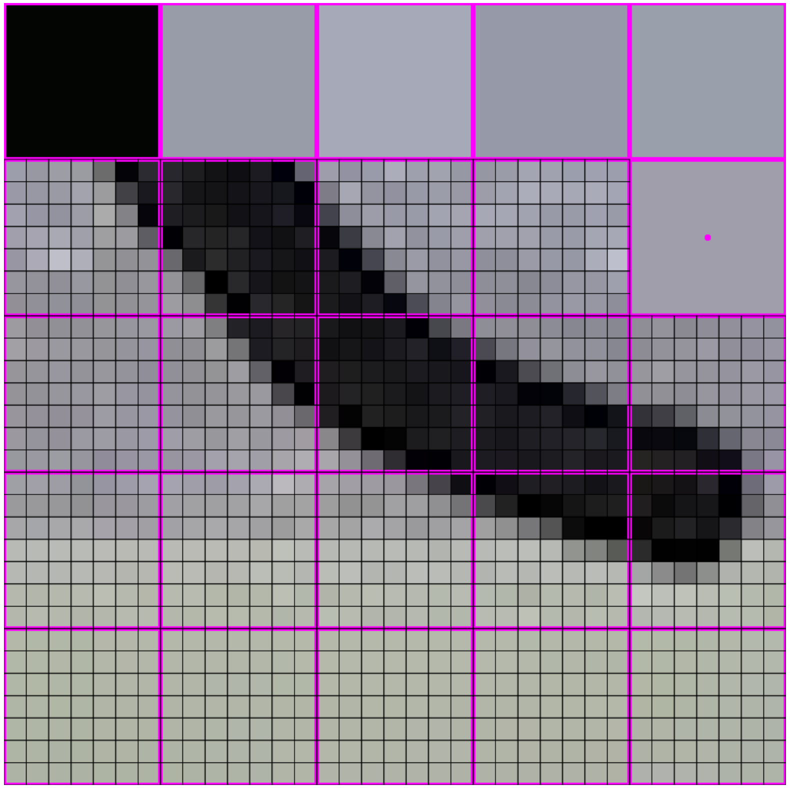
448x448 -> 64x64



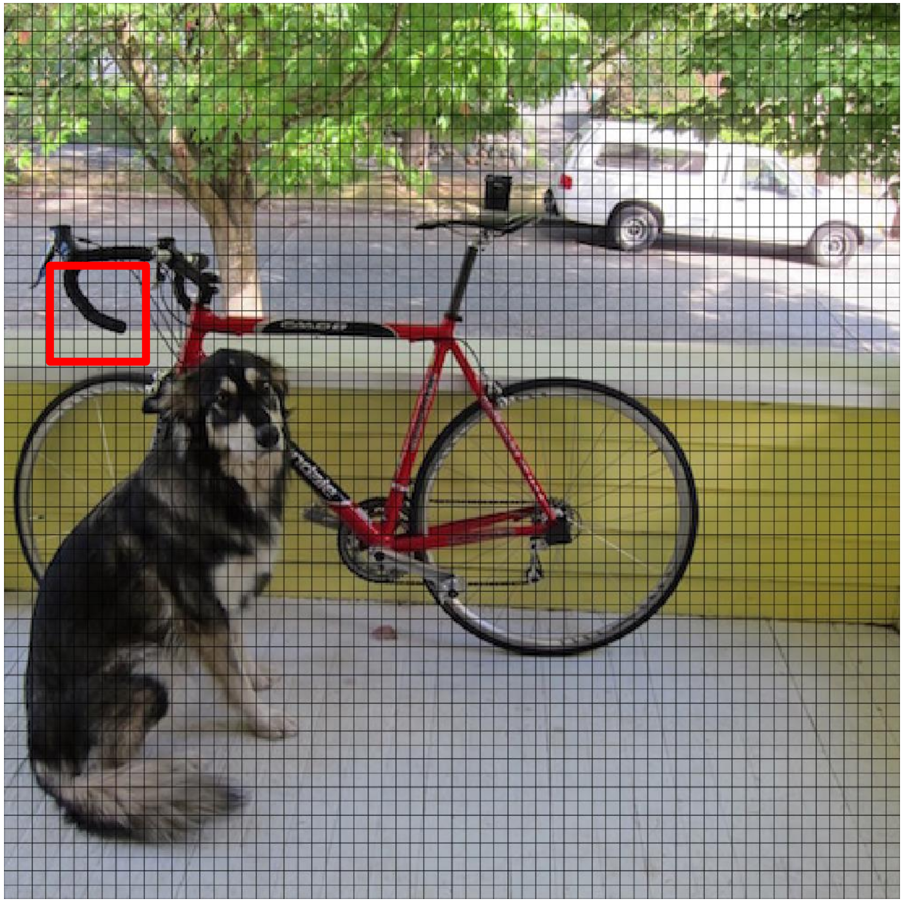


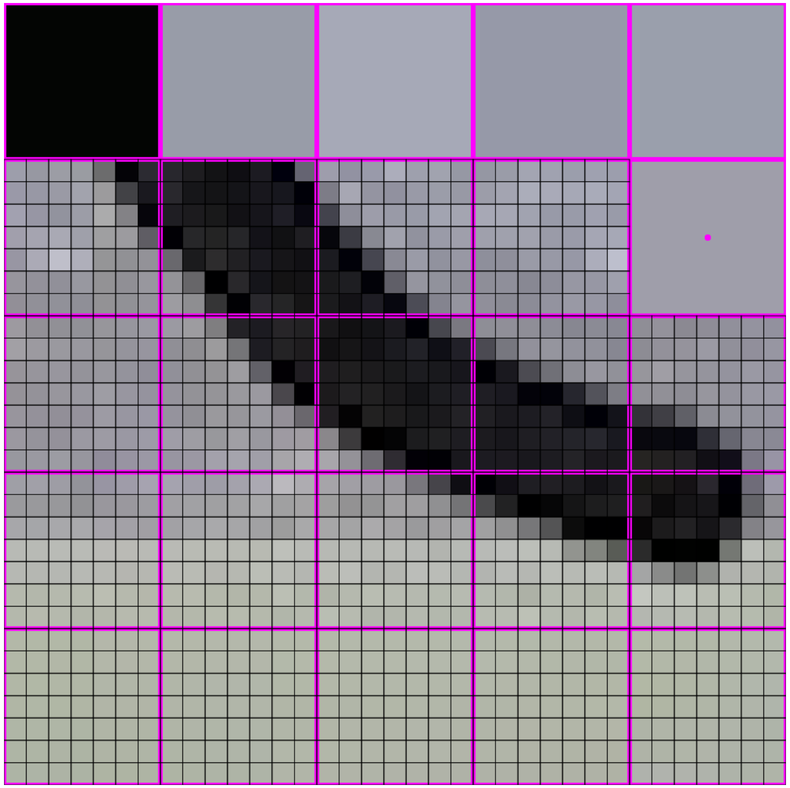
448x448 -> 64x64



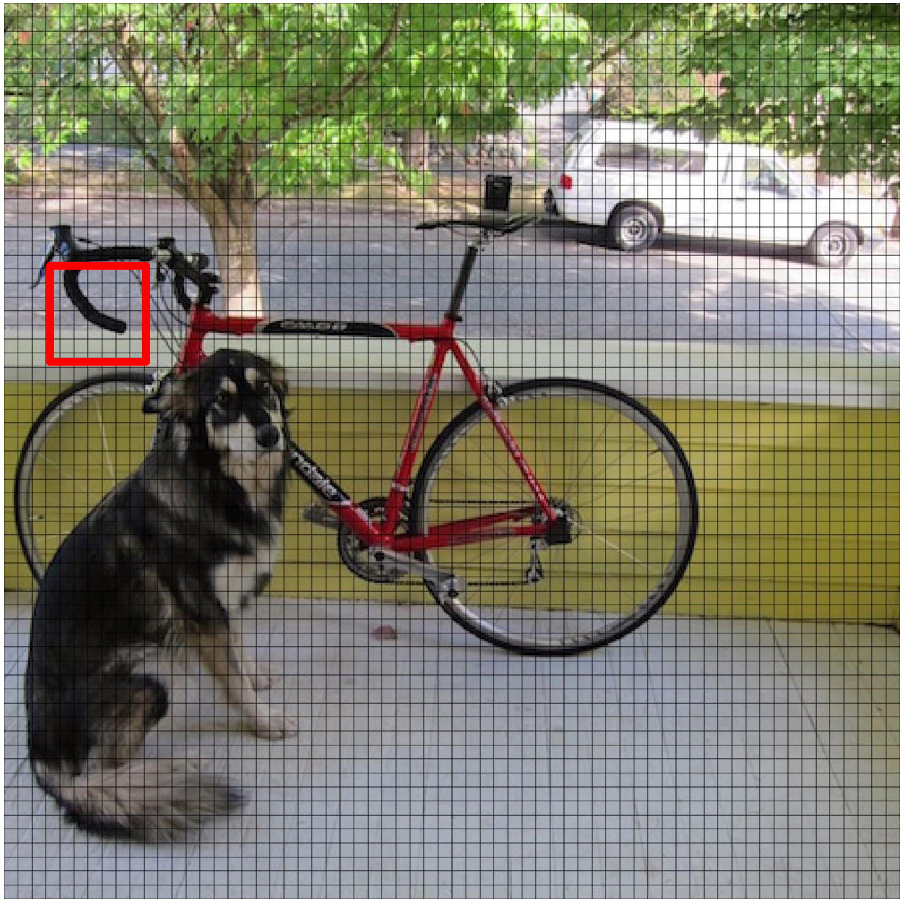


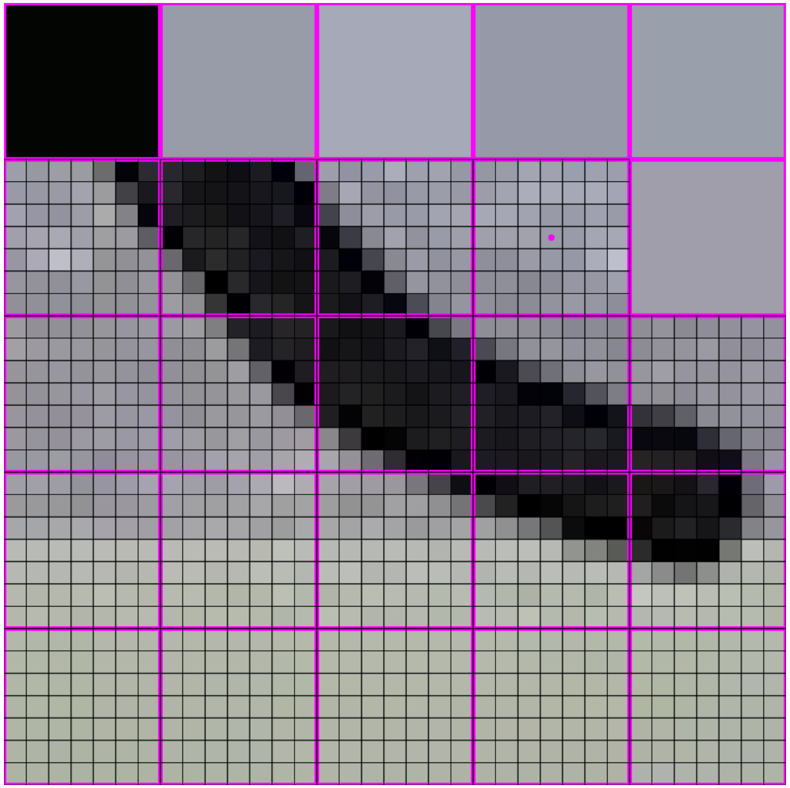
448x448 -> 64x64



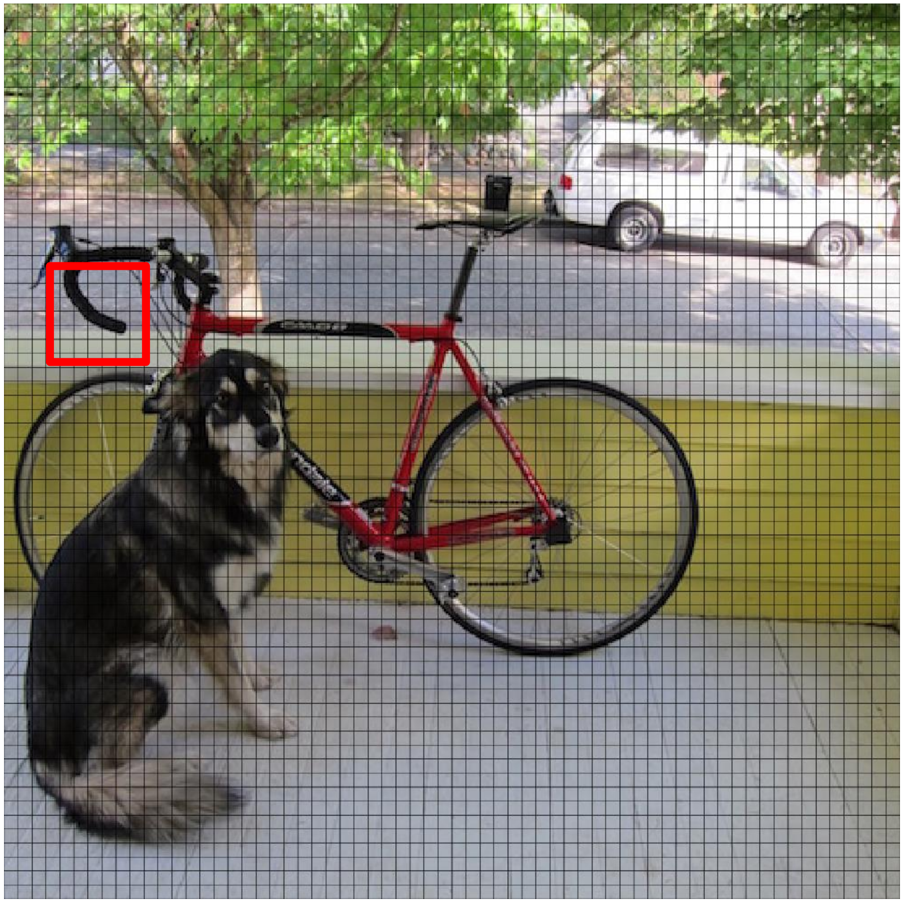


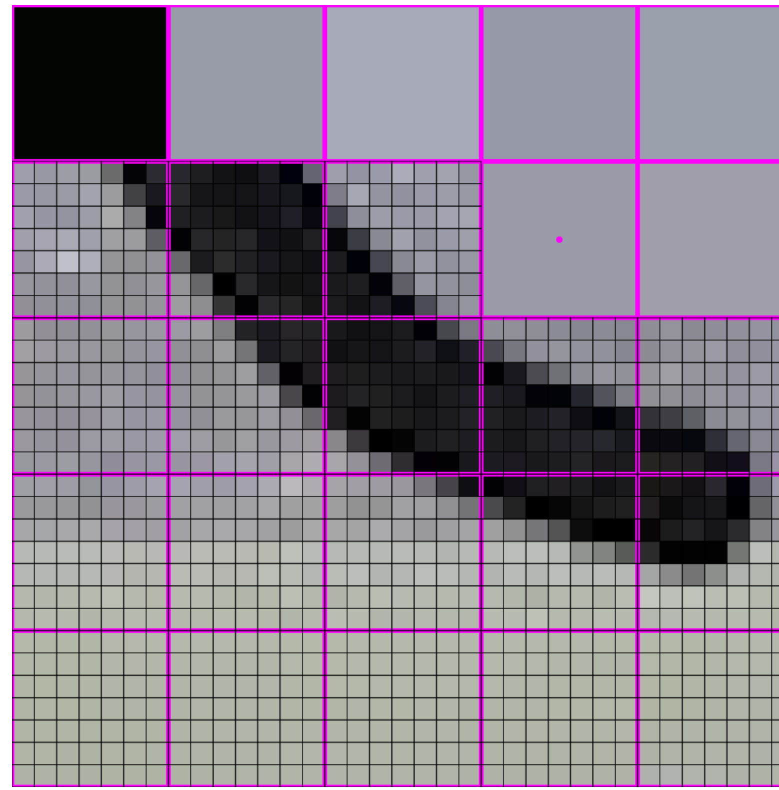
448x448 -> 64x64



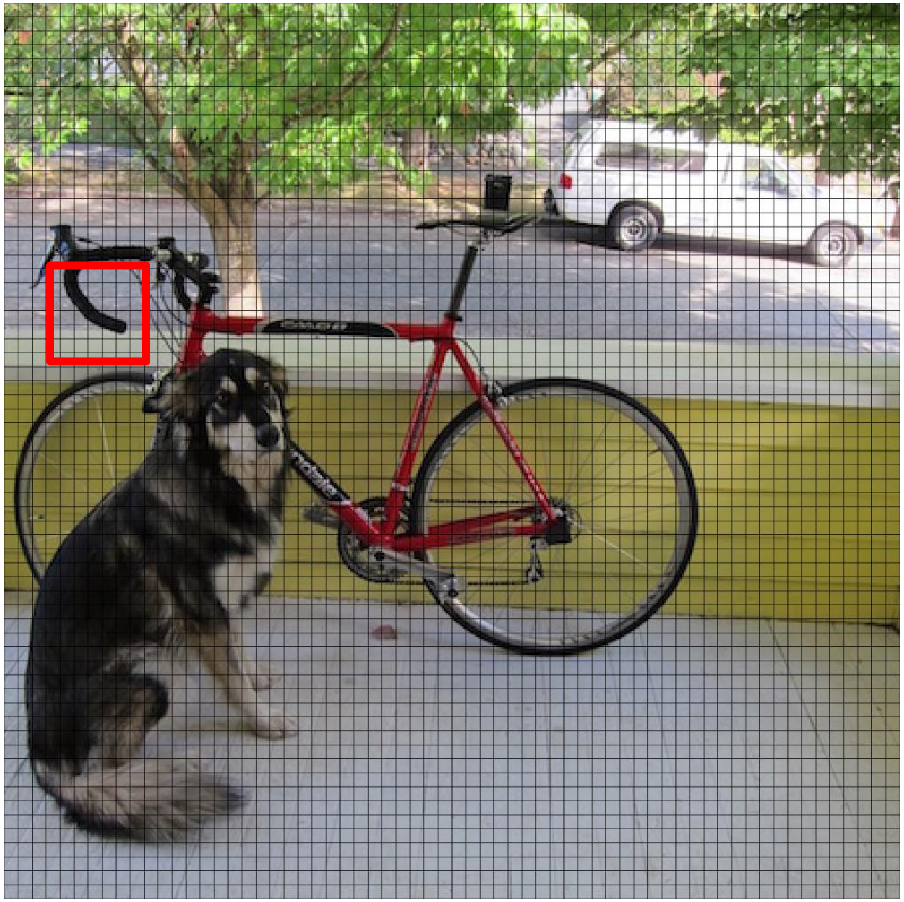


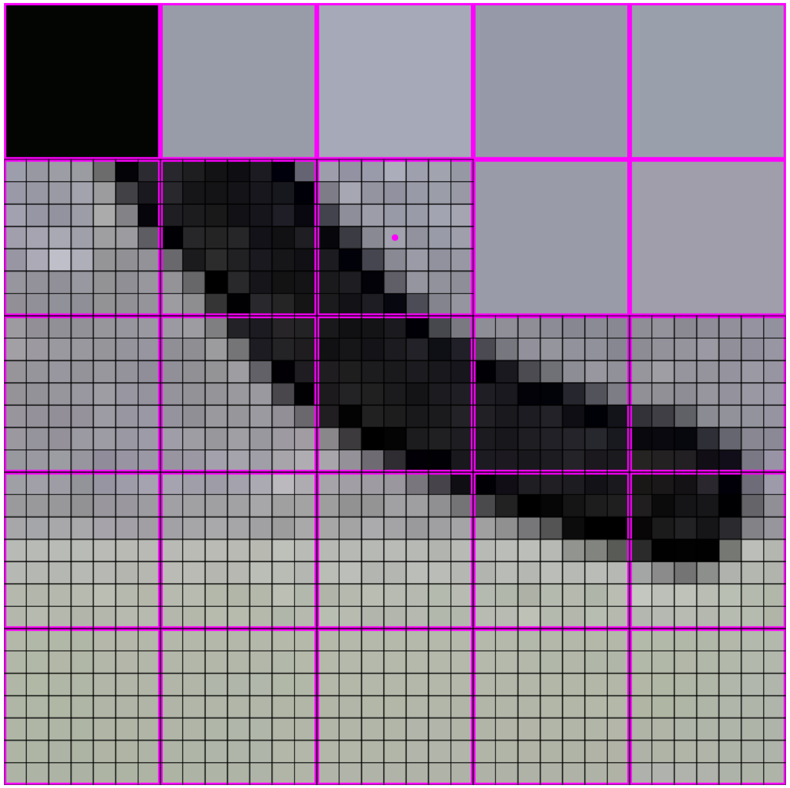
448x448 -> 64x64



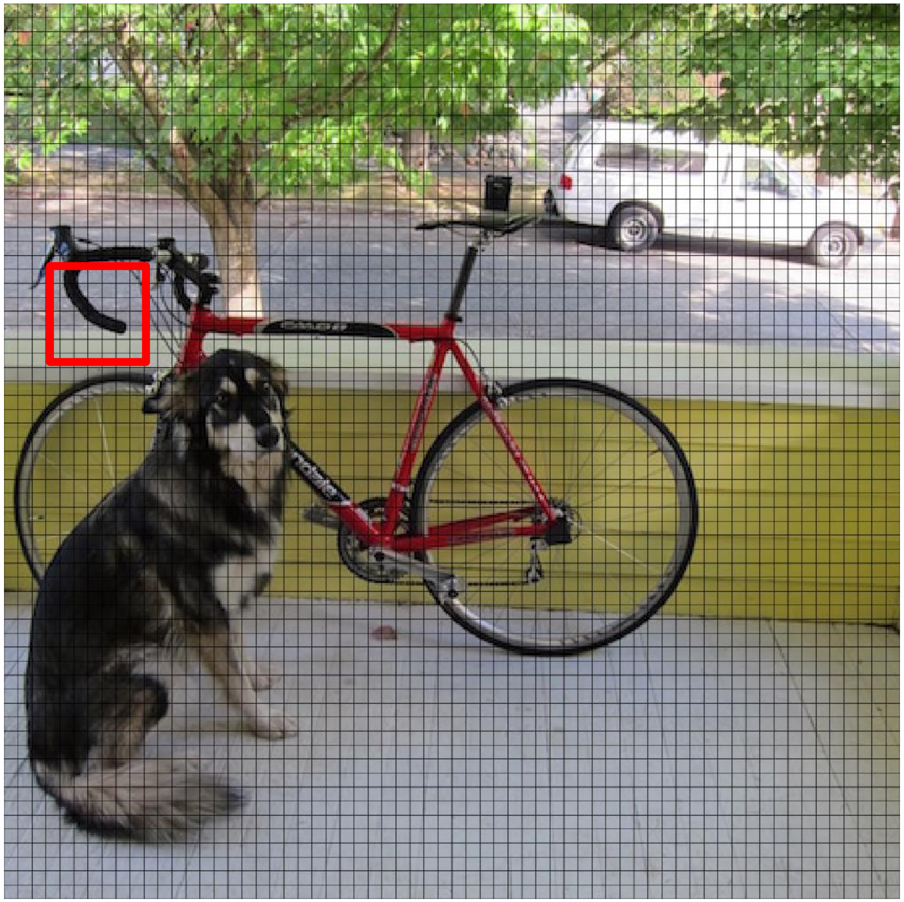


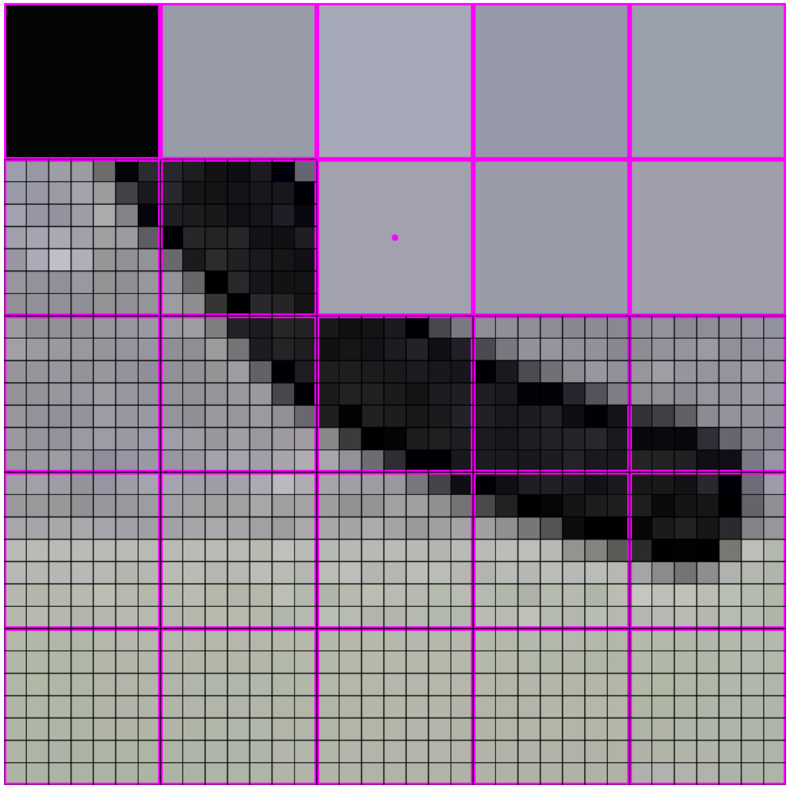
448x448 -> 64x64



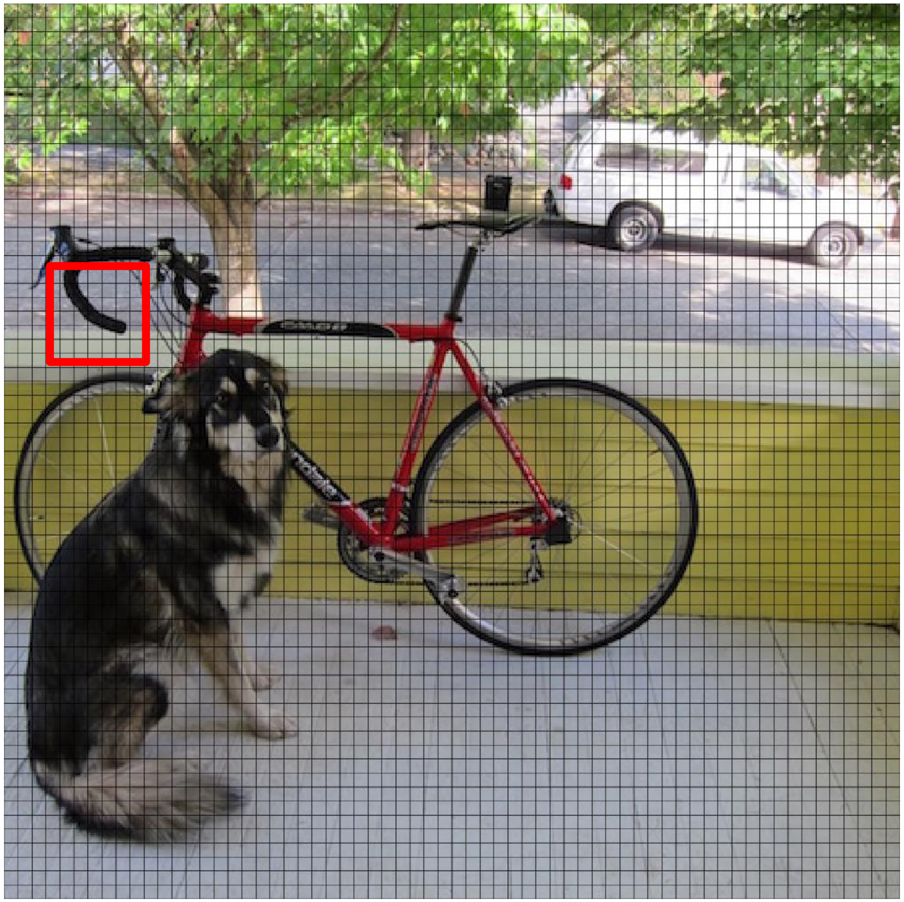


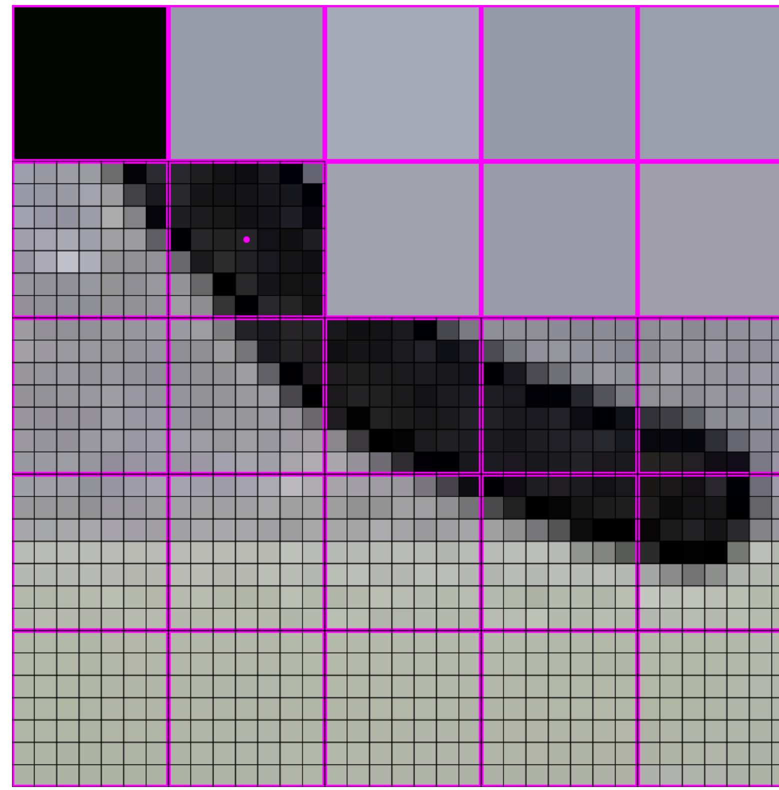
448x448 -> 64x64



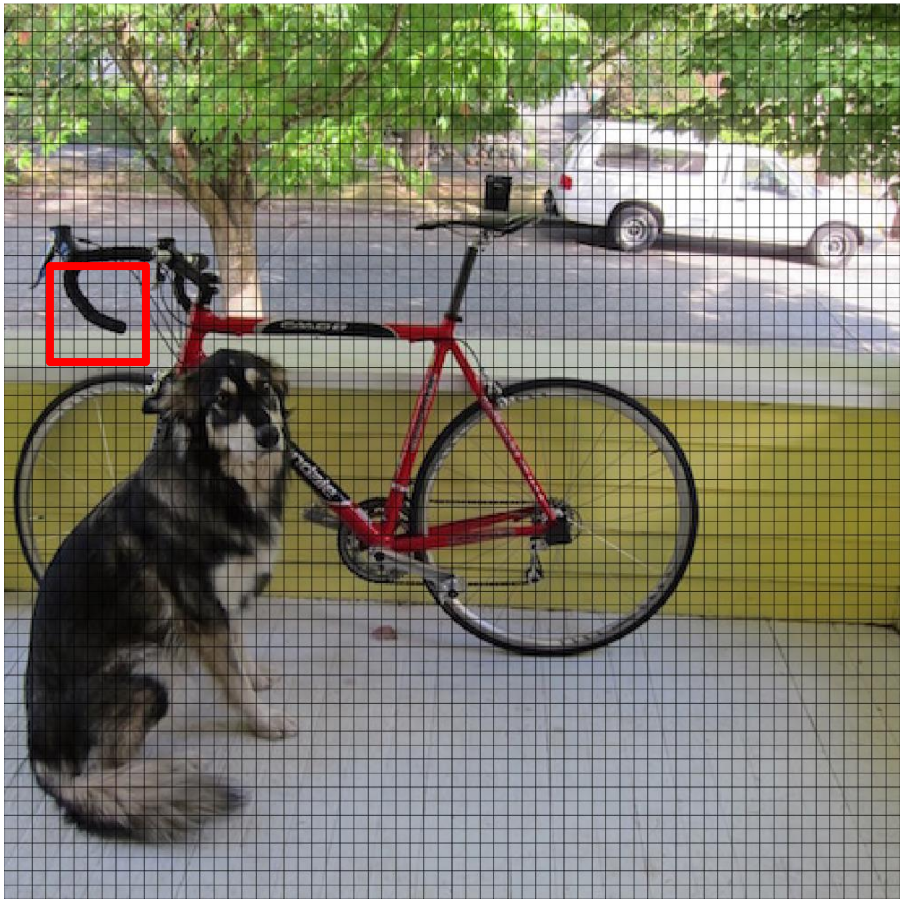


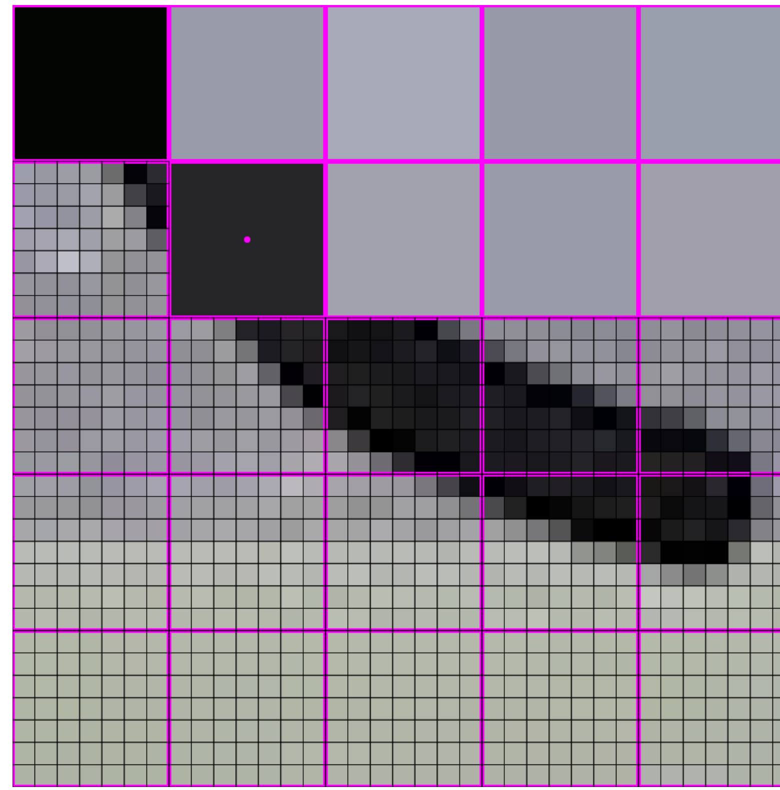
448x448 -> 64x64



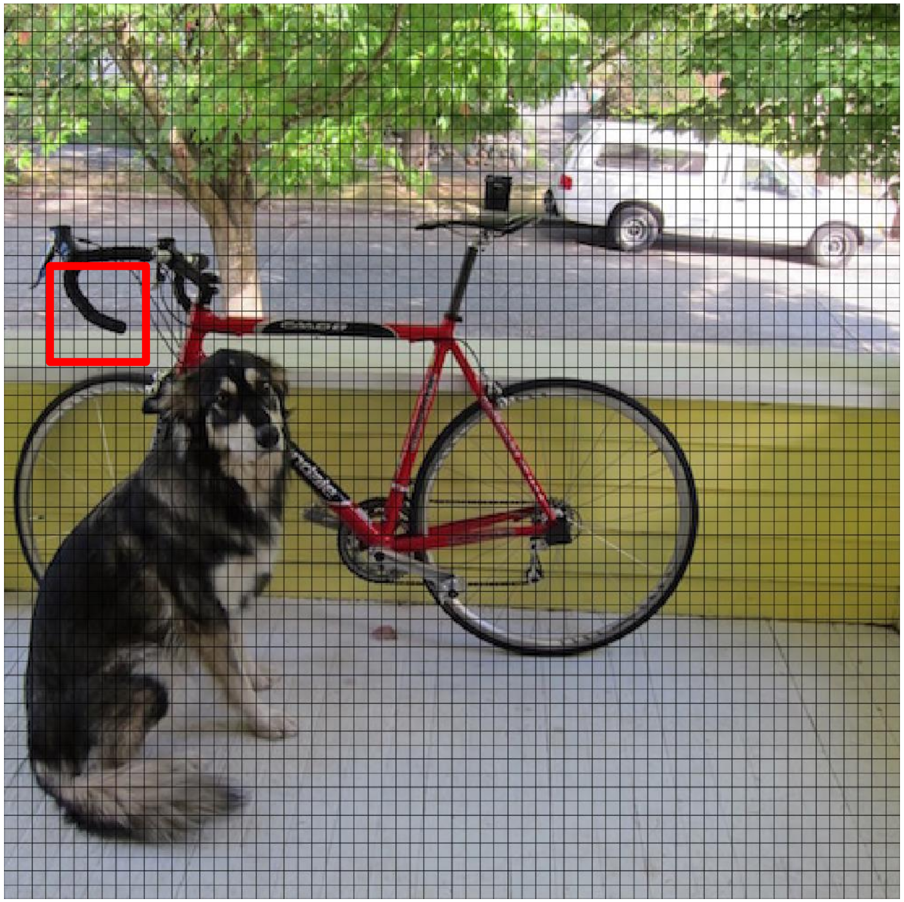


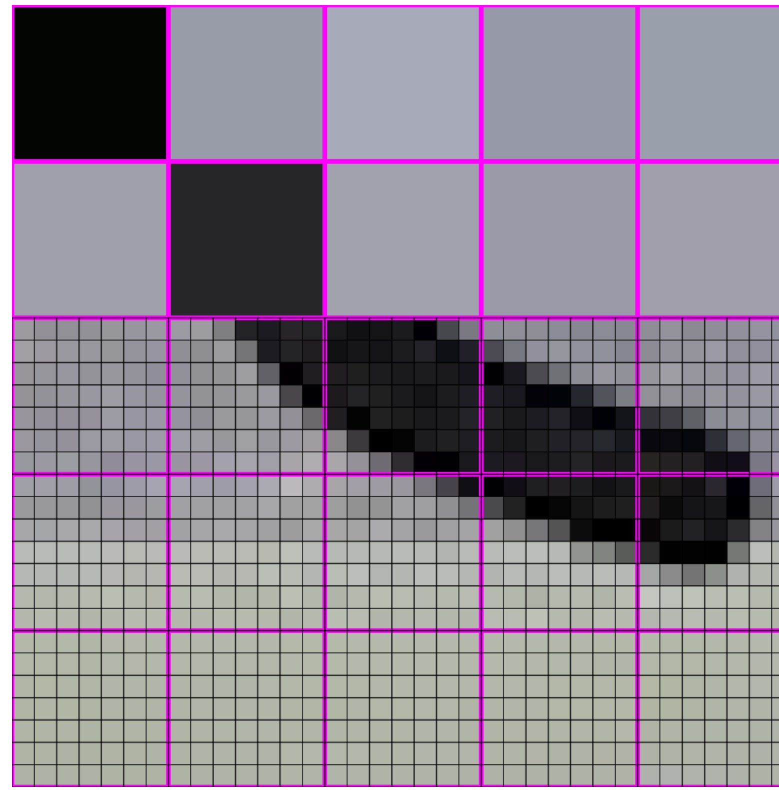
448x448 -> 64x64



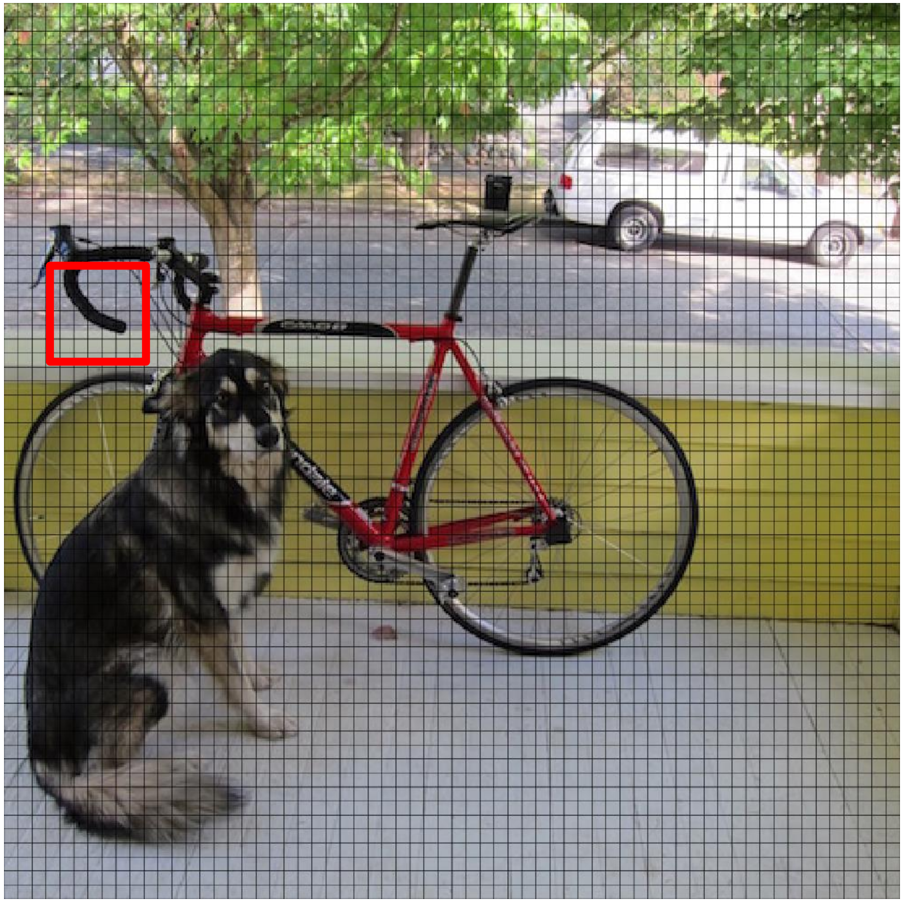


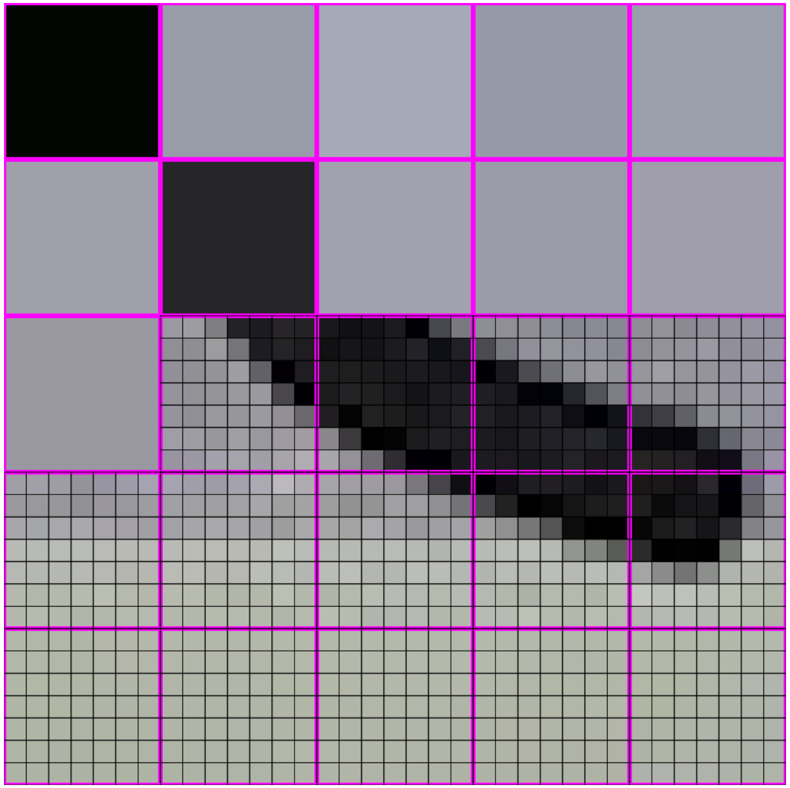
448x448 -> 64x64



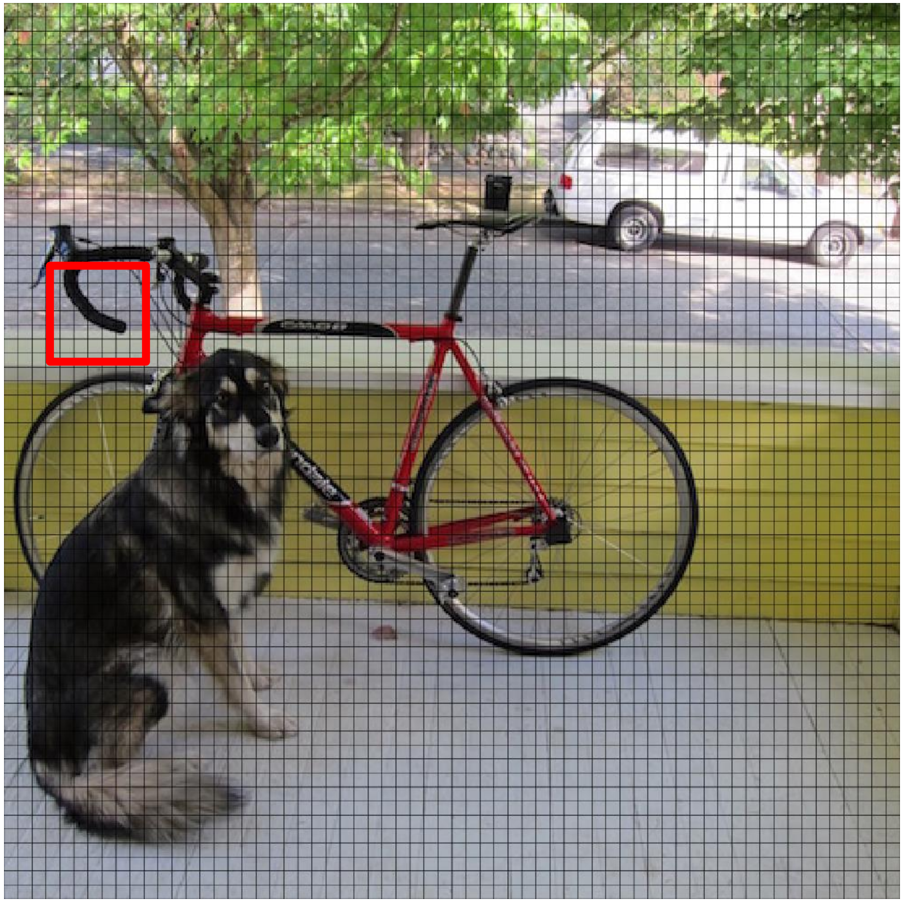


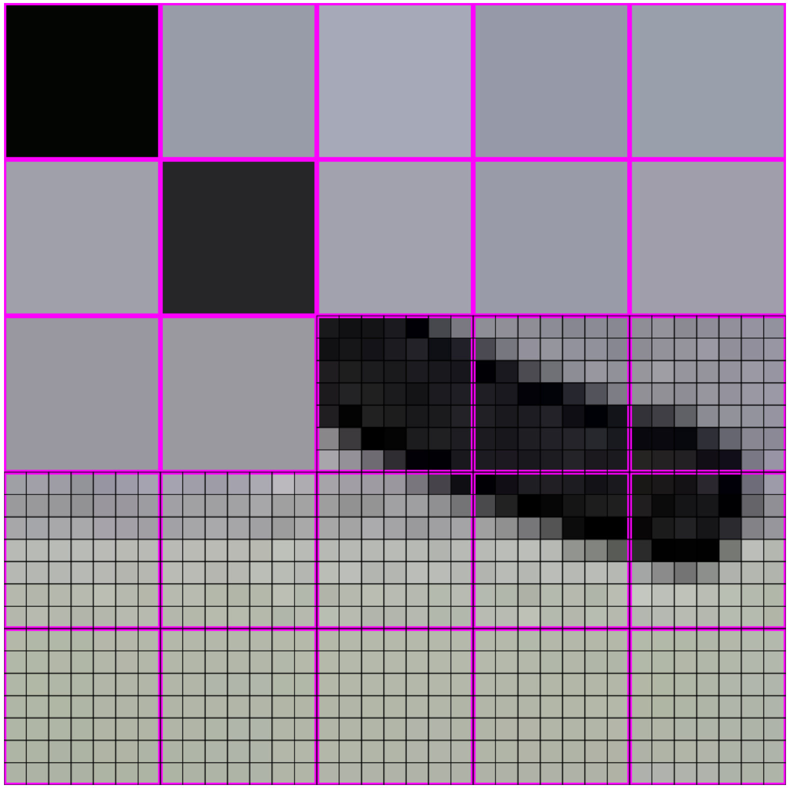
448x448 -> 64x64



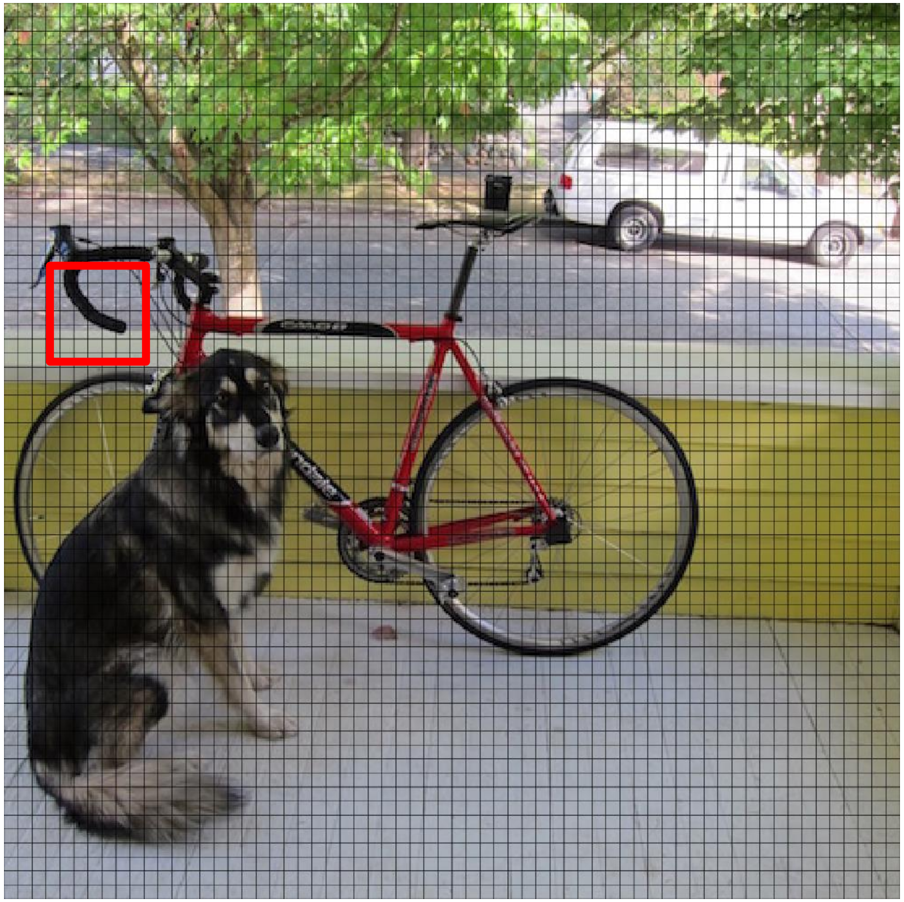


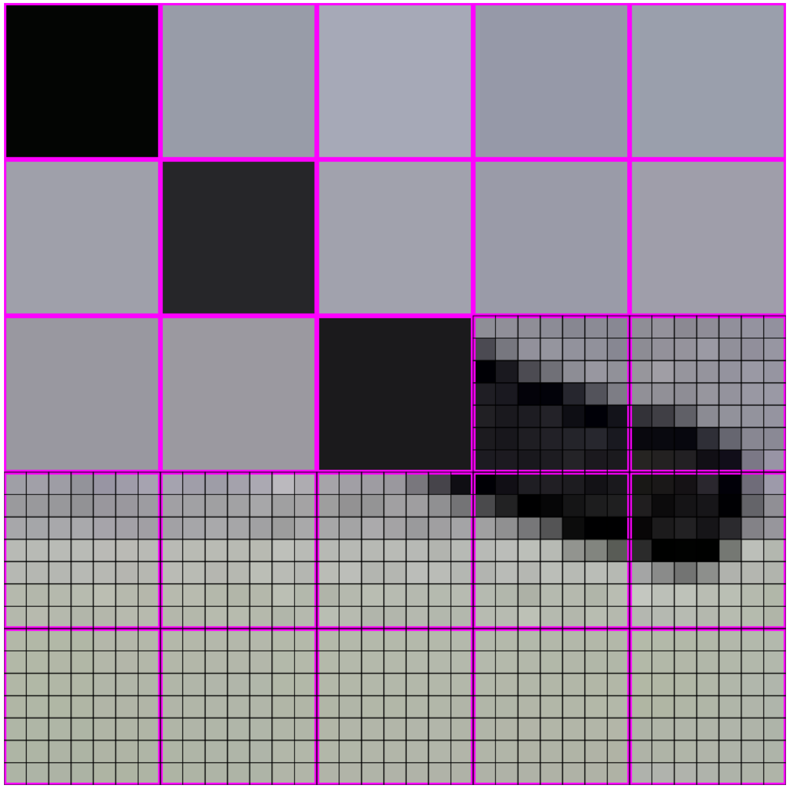
448x448 -> 64x64



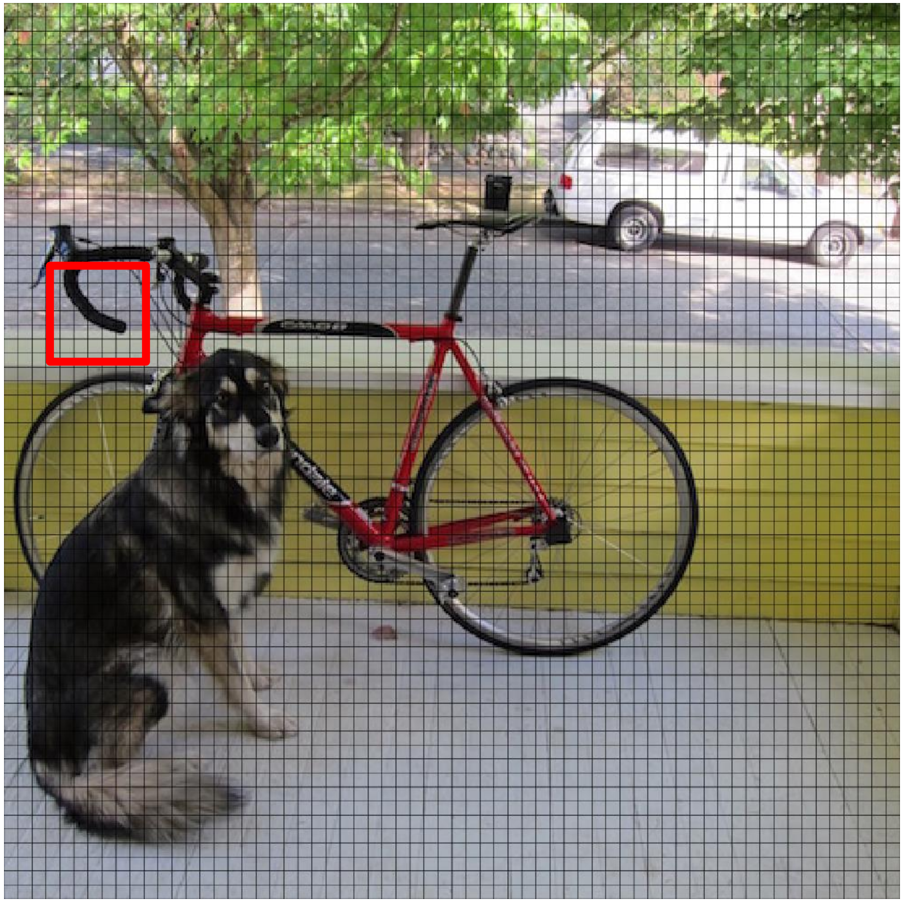


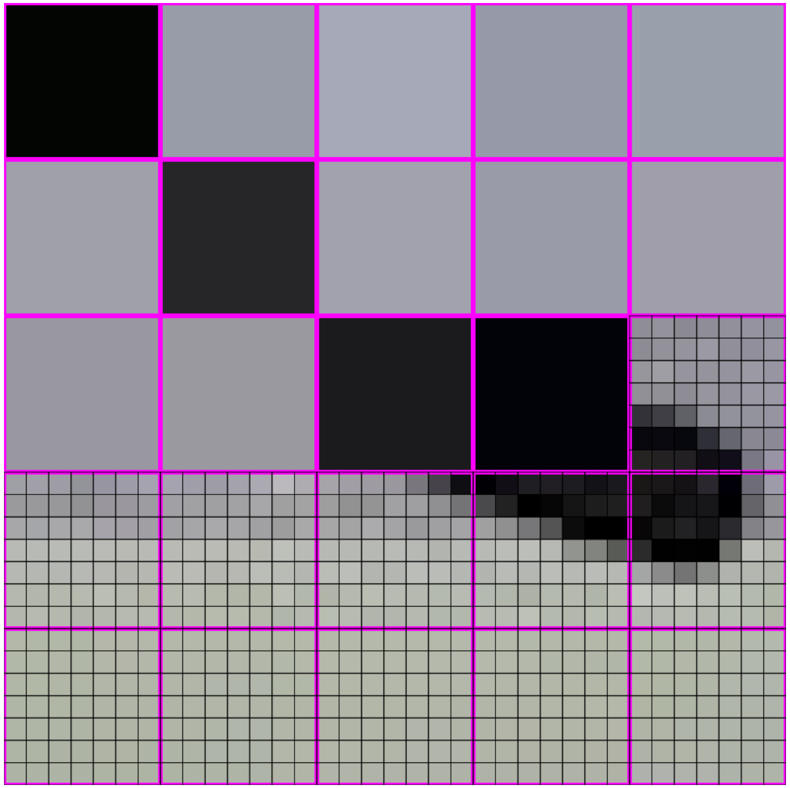
448x448 -> 64x64



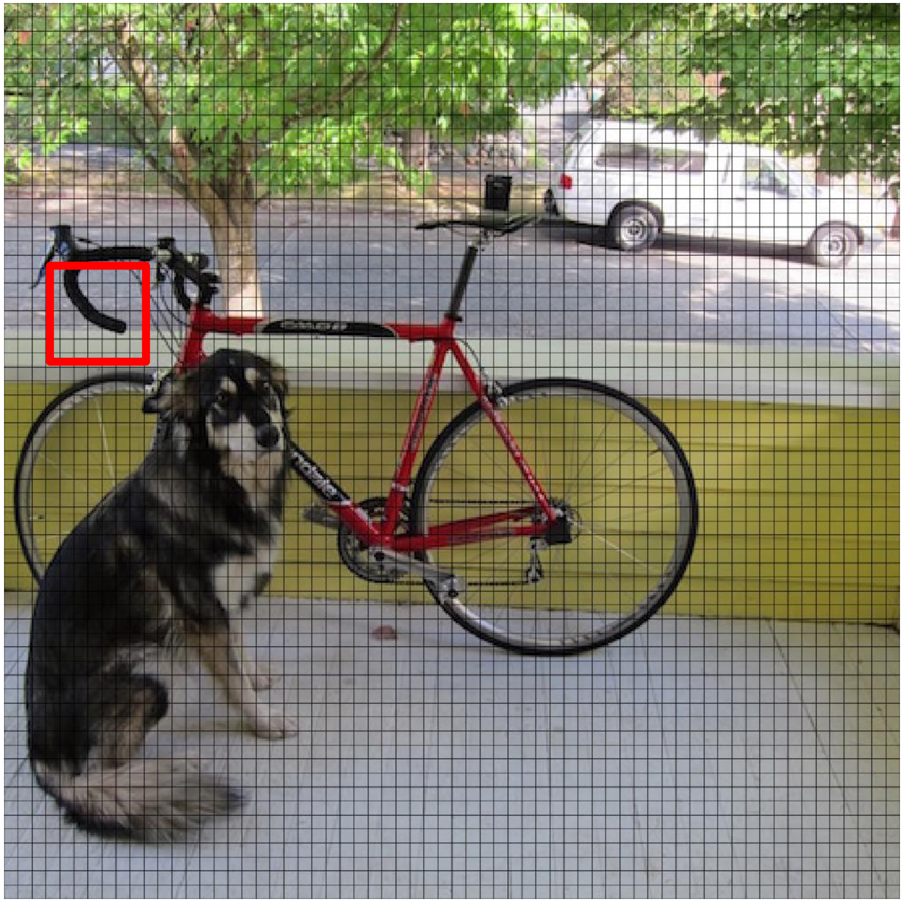


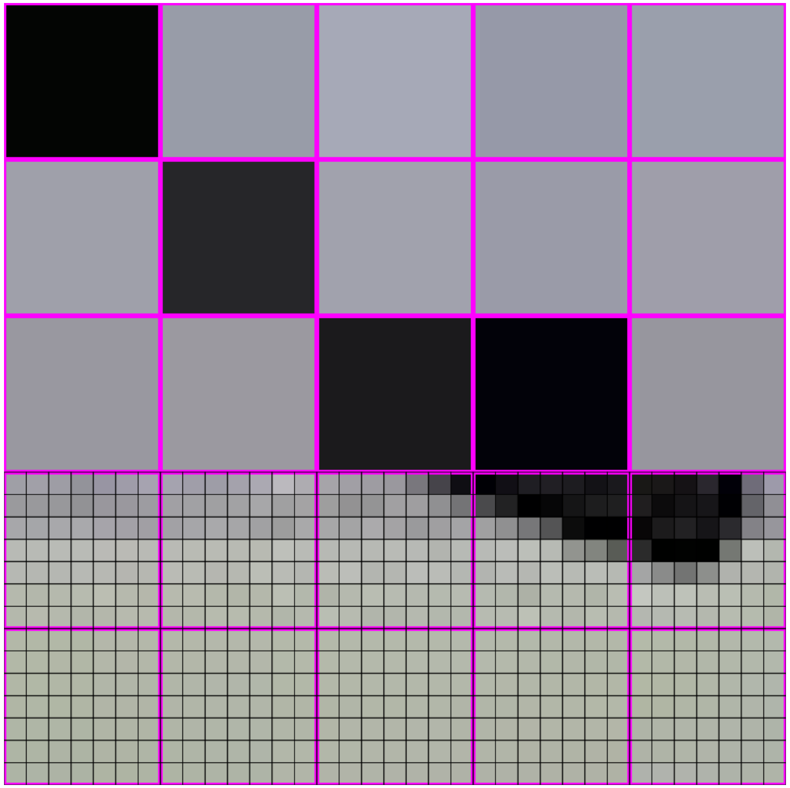
448x448 -> 64x64



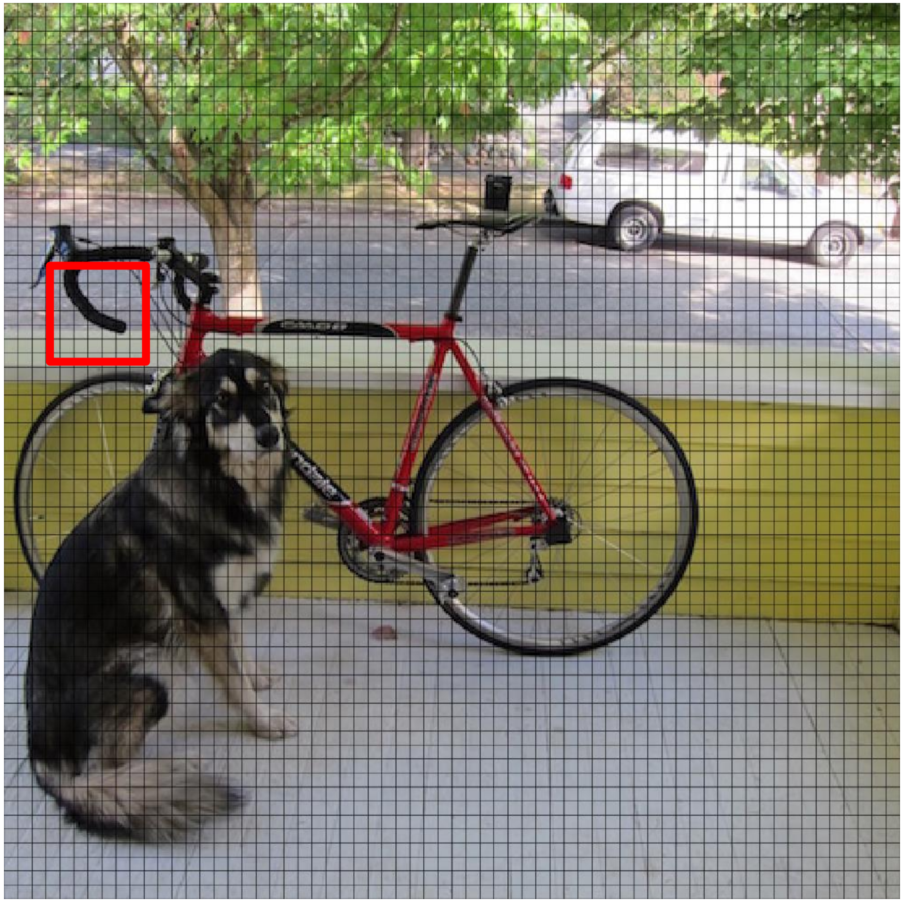


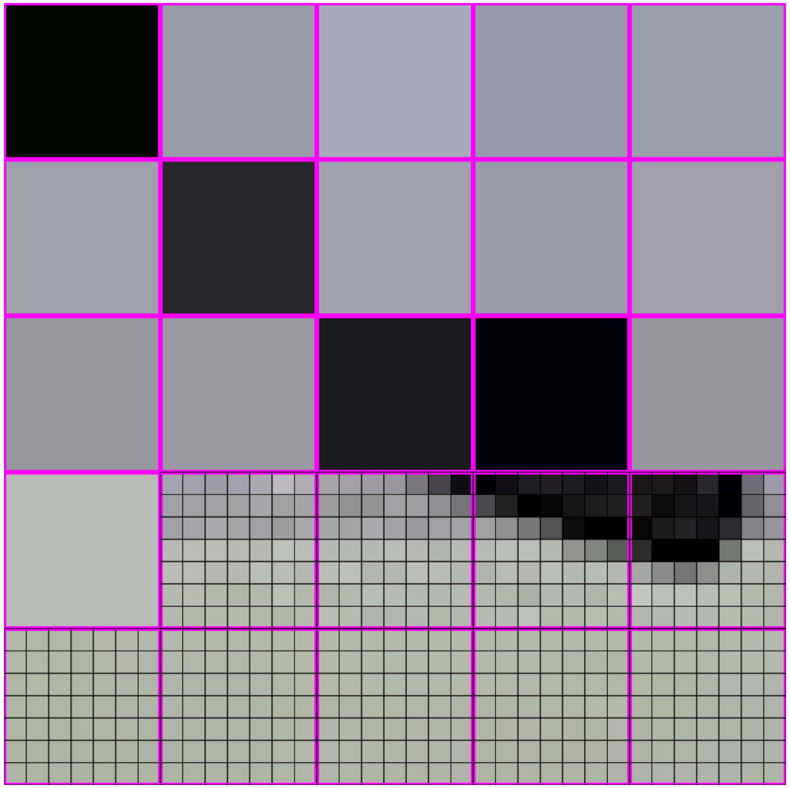
448x448 -> 64x64



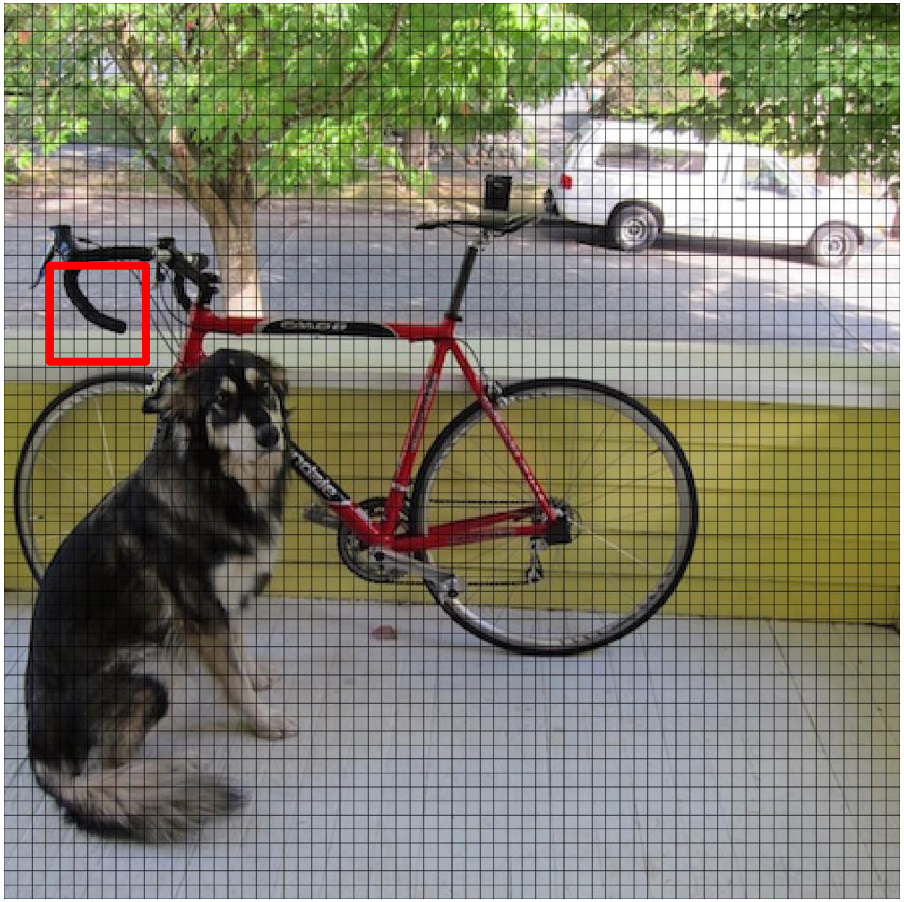


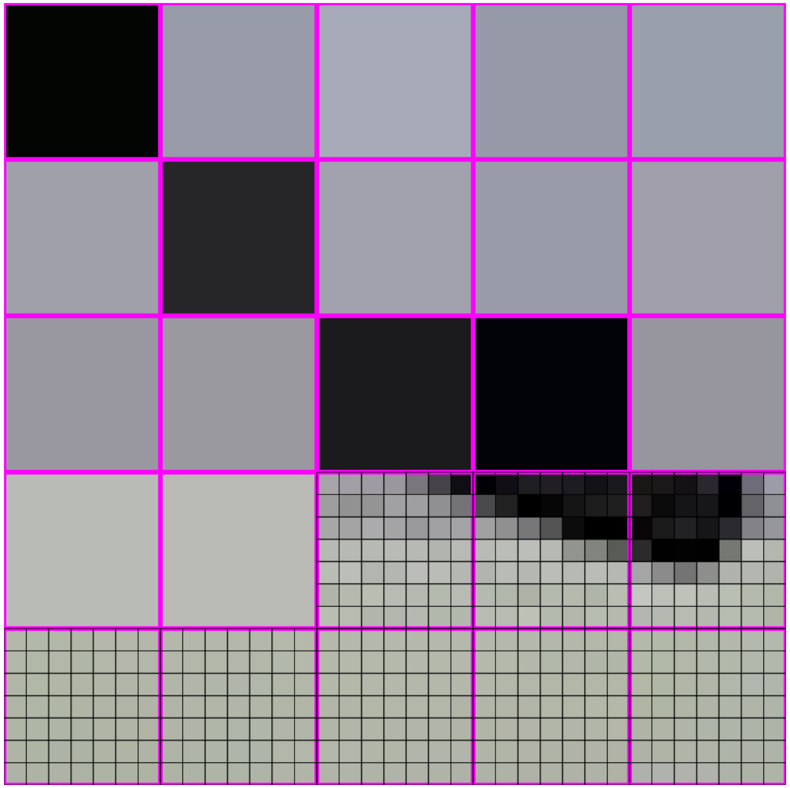
448x448 -> 64x64



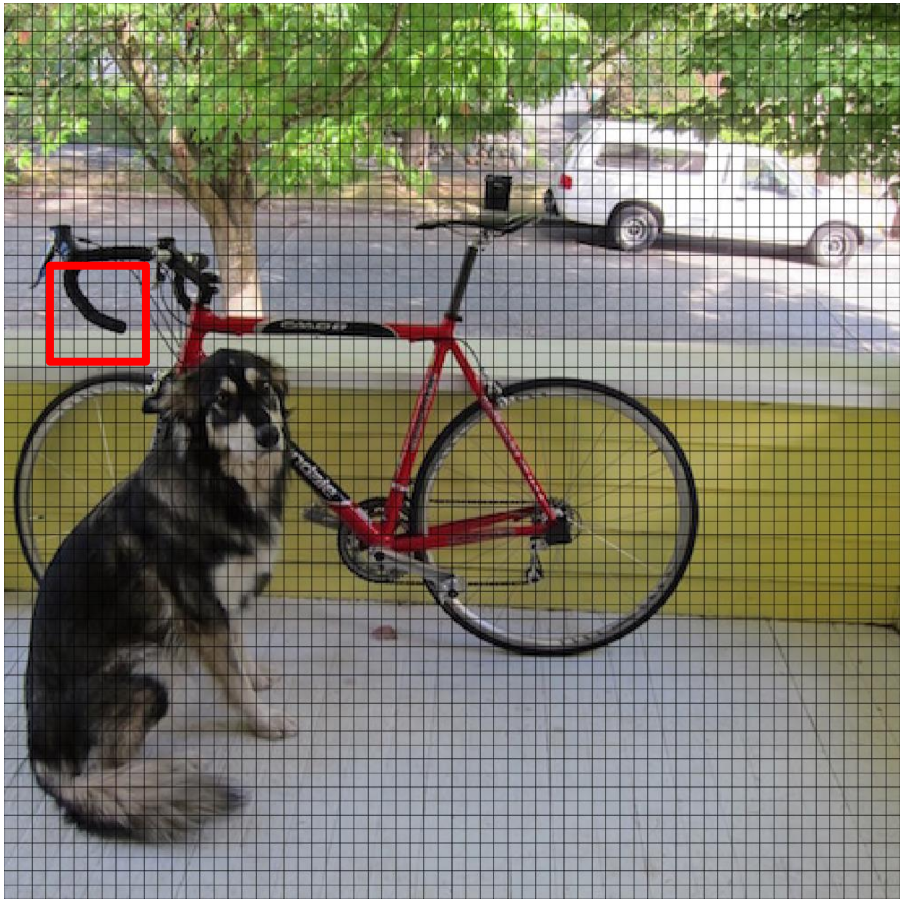


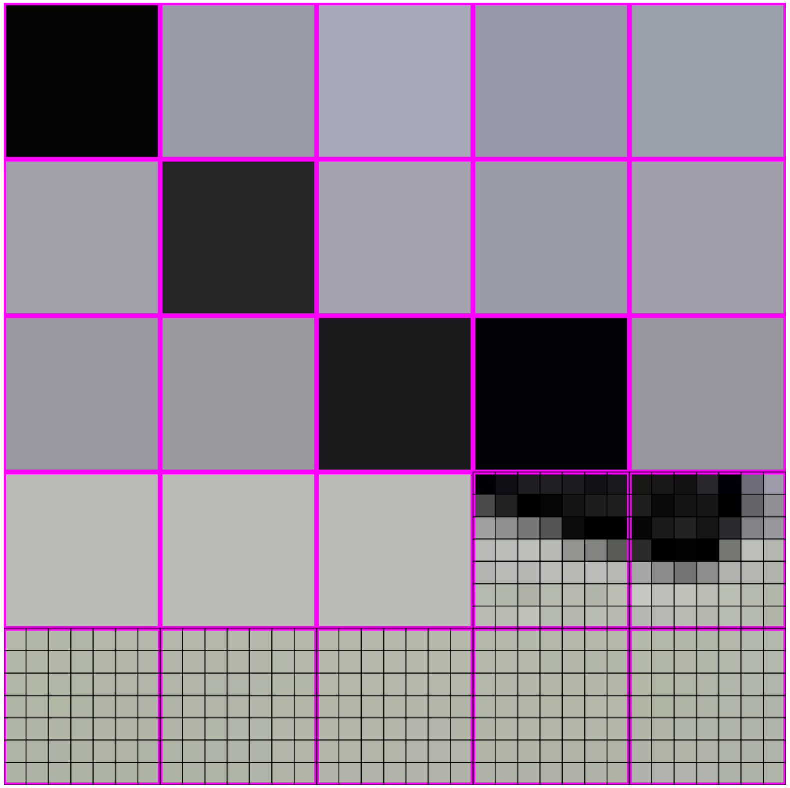
448x448 -> 64x64



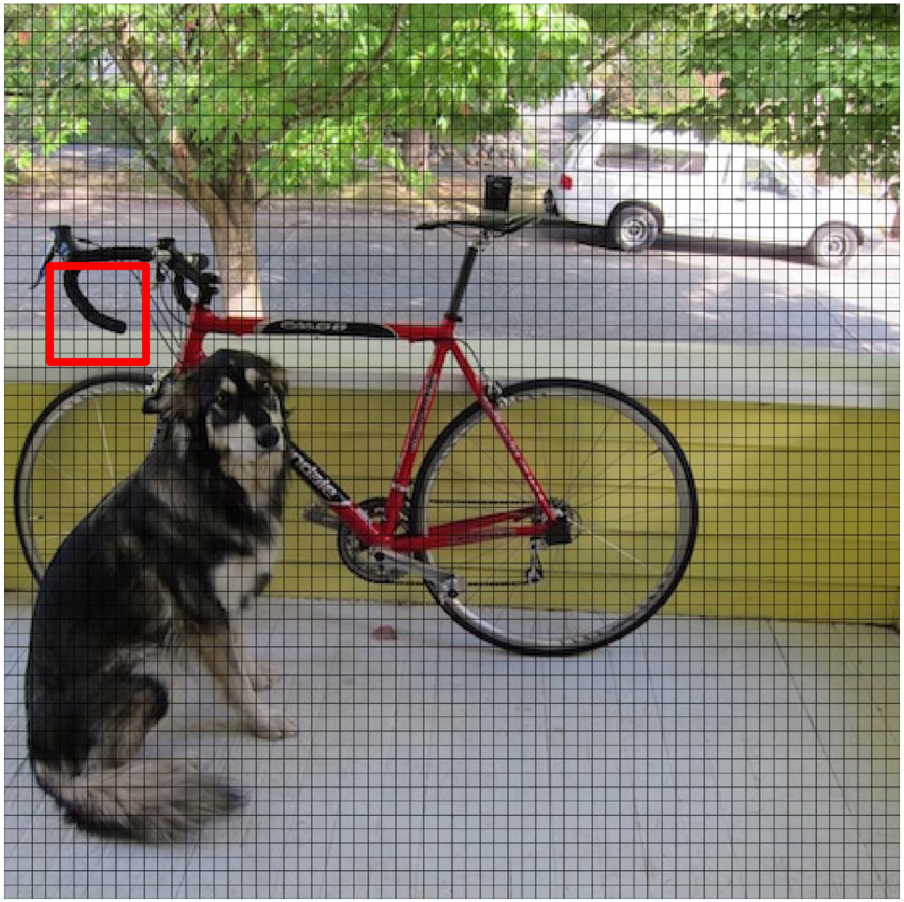


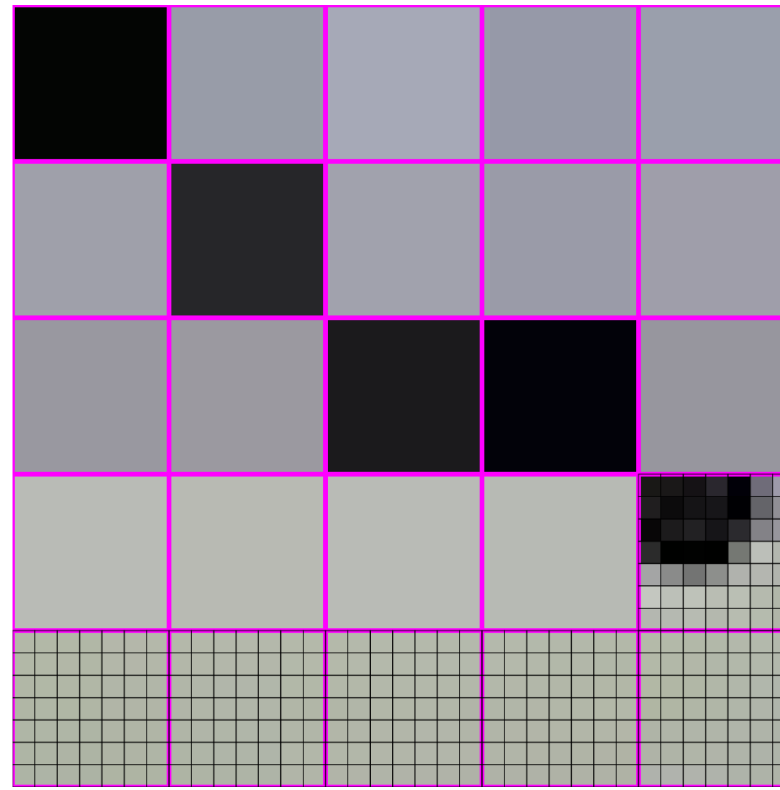
448x448 -> 64x64



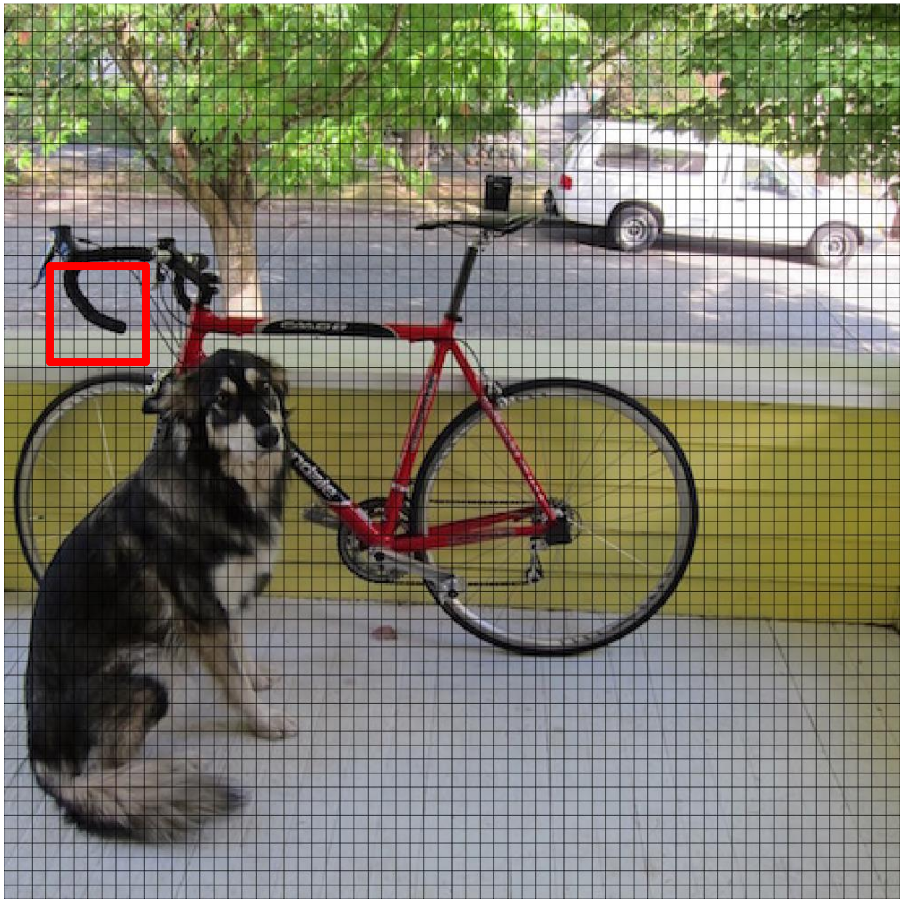


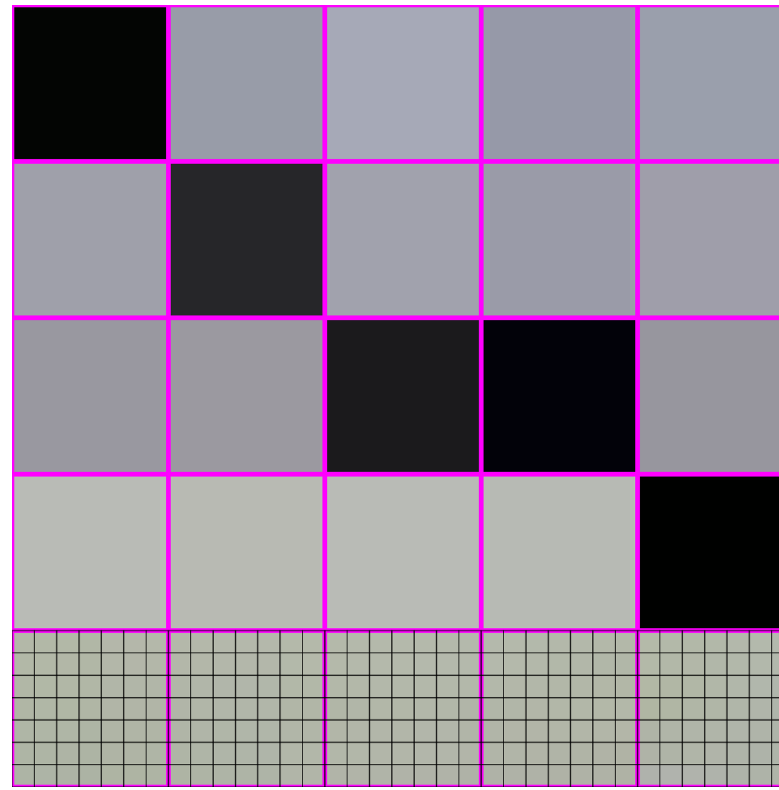
448x448 -> 64x64



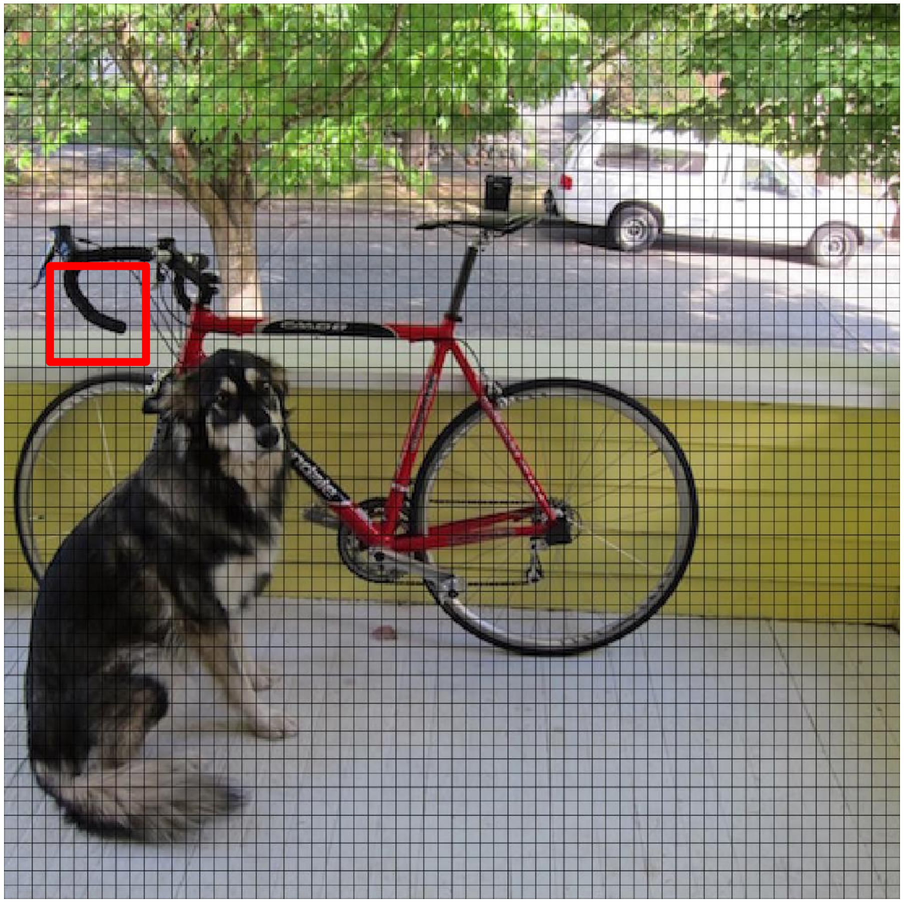


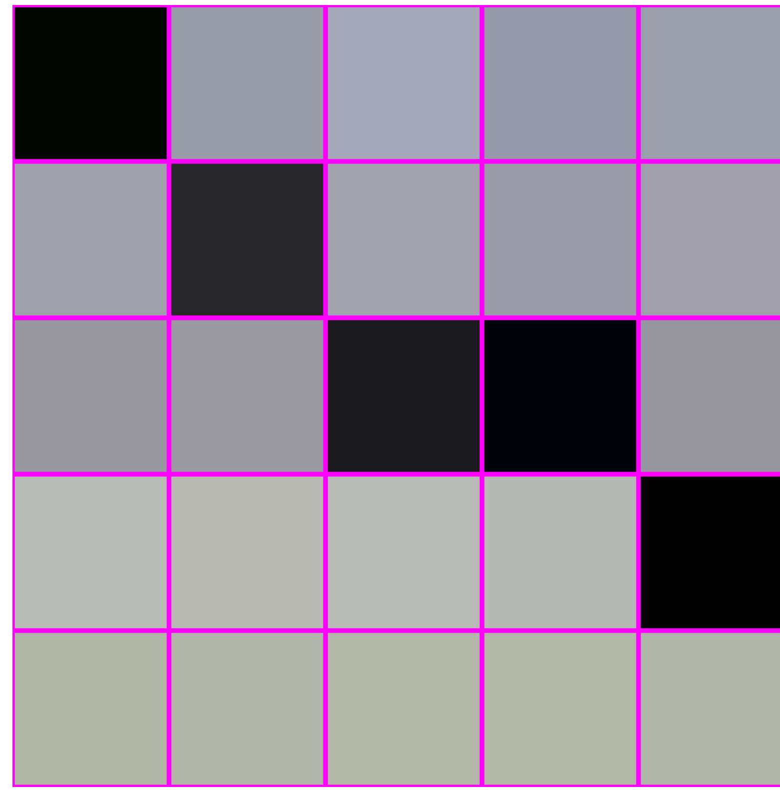
448x448 -> 64x64



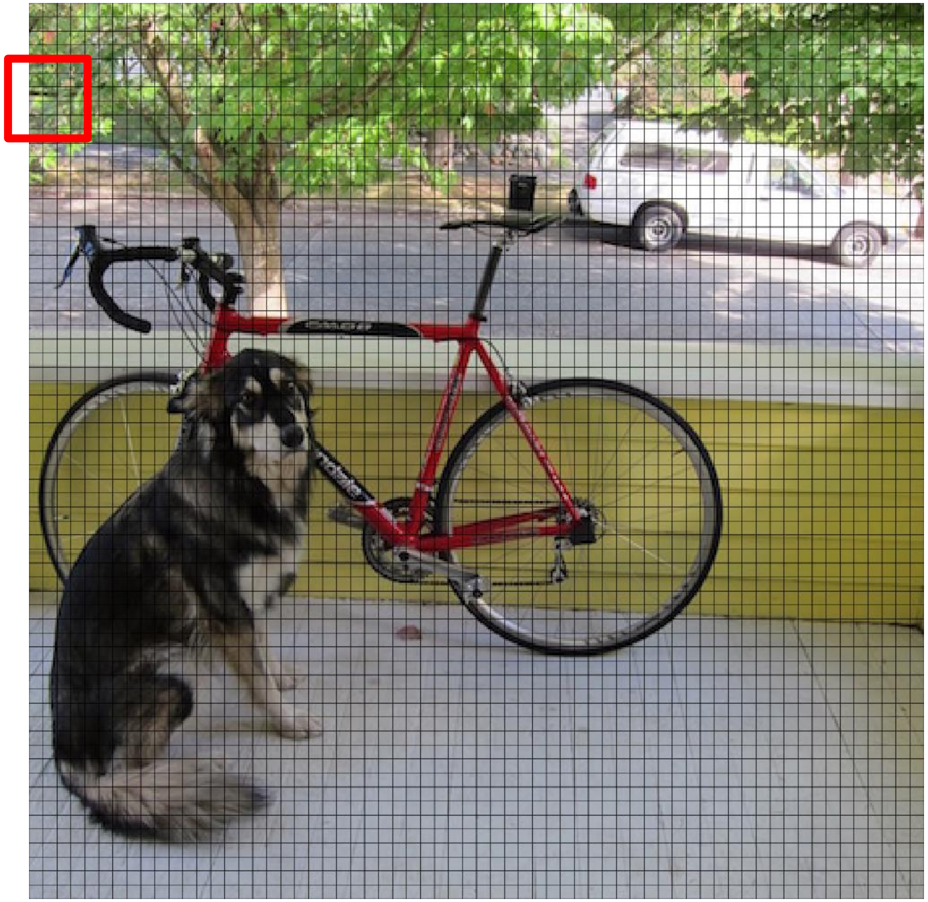


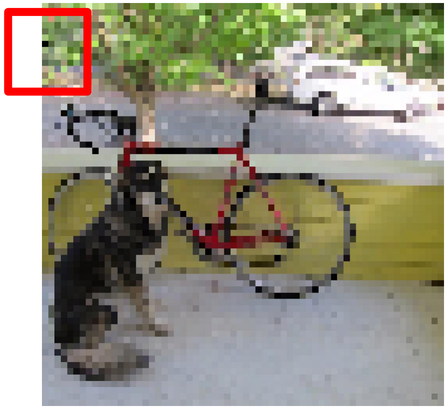
448x448 -> 64x64



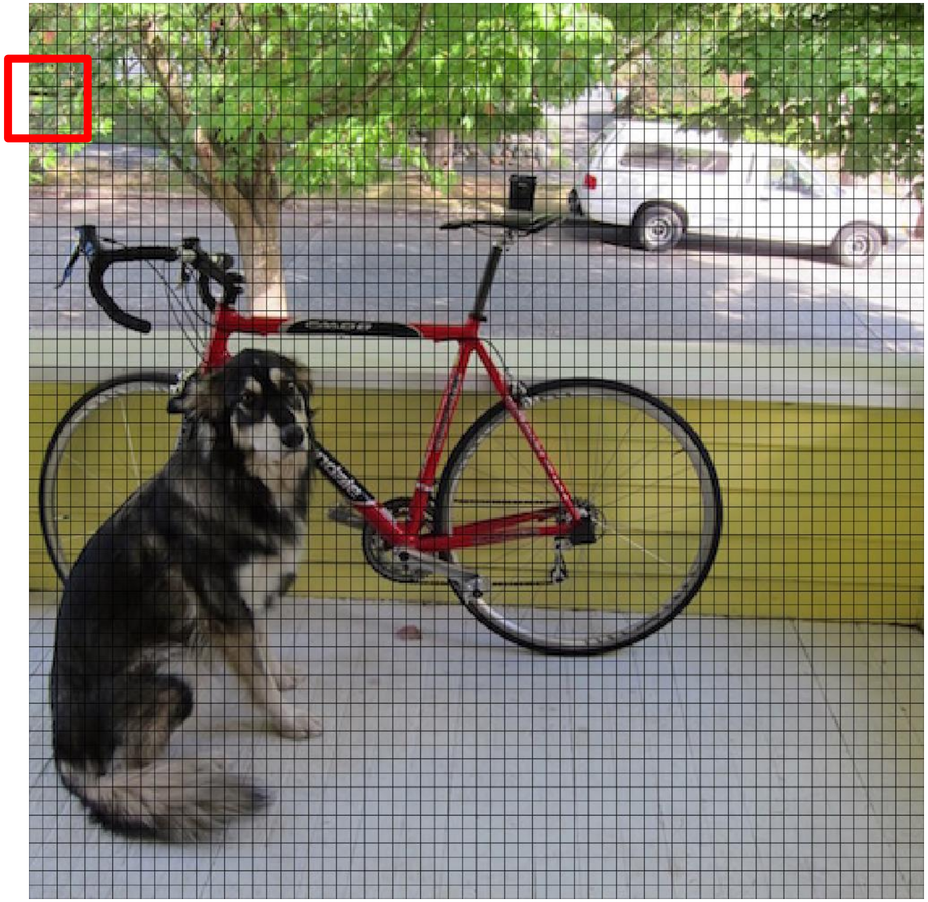


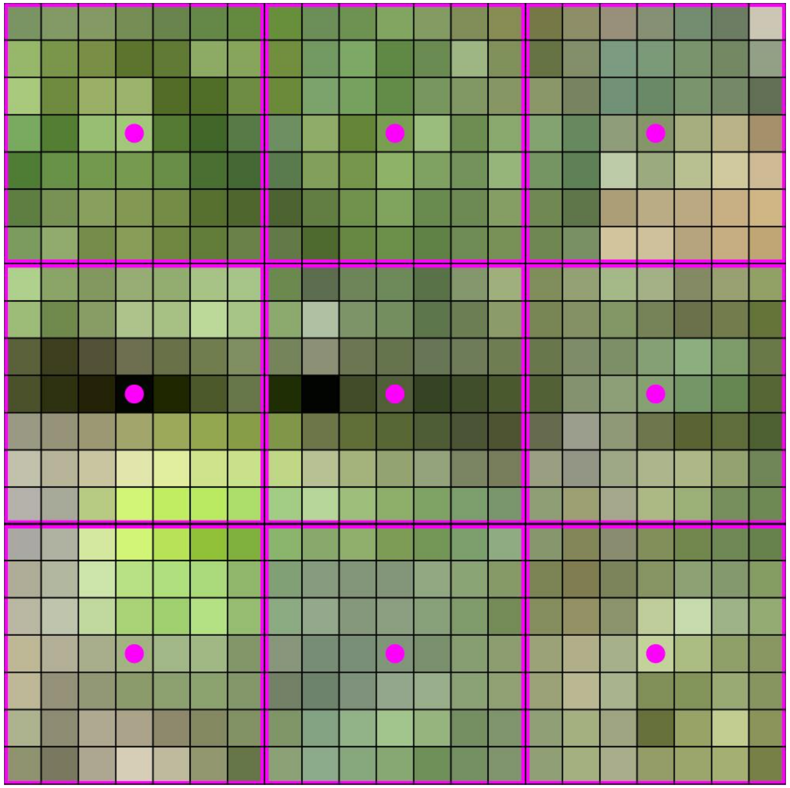
448x448 -> 64x64



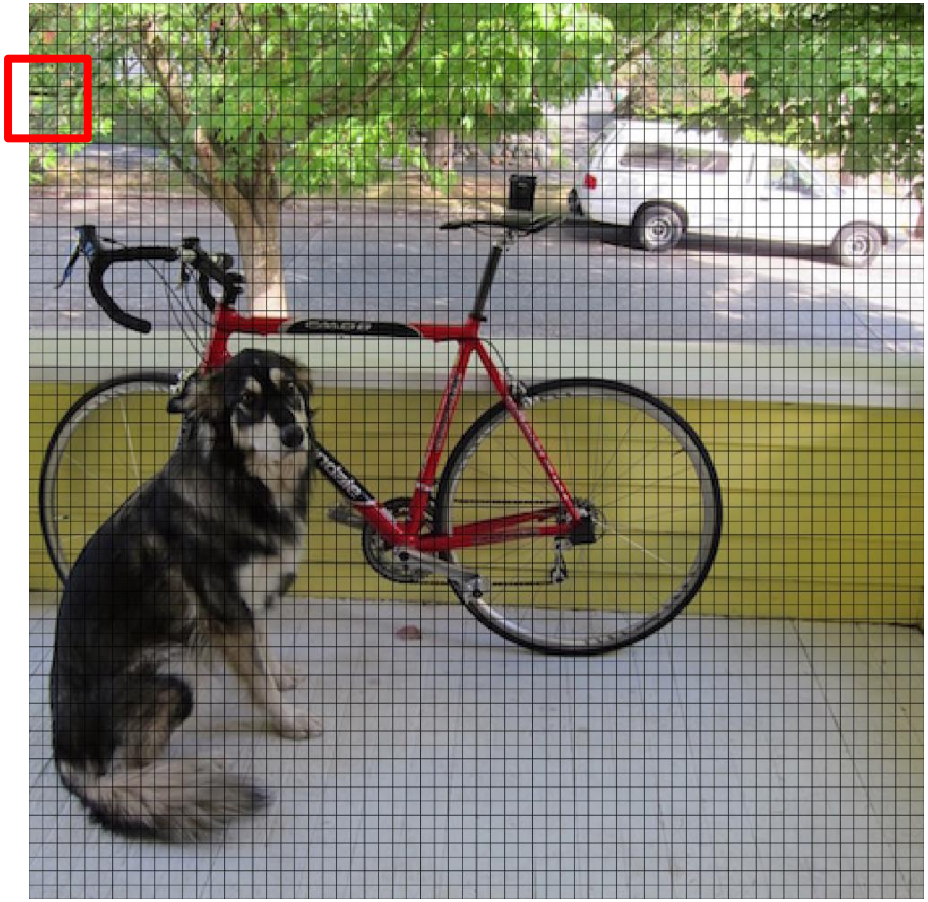


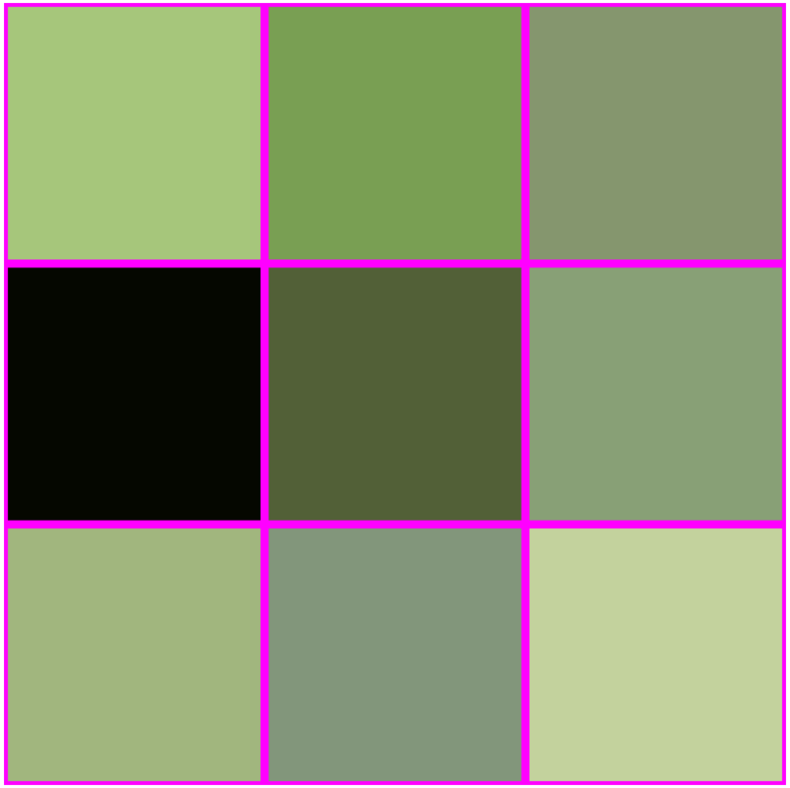
448x448 -> 64x64



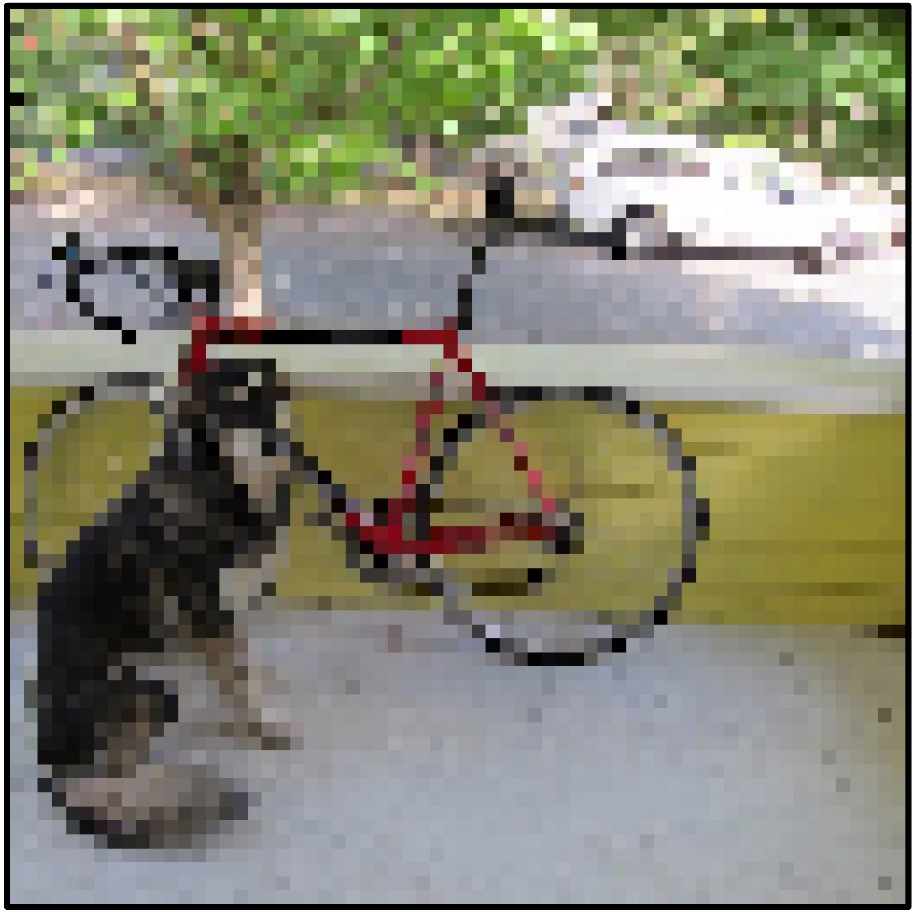


448x448 -> 64x64





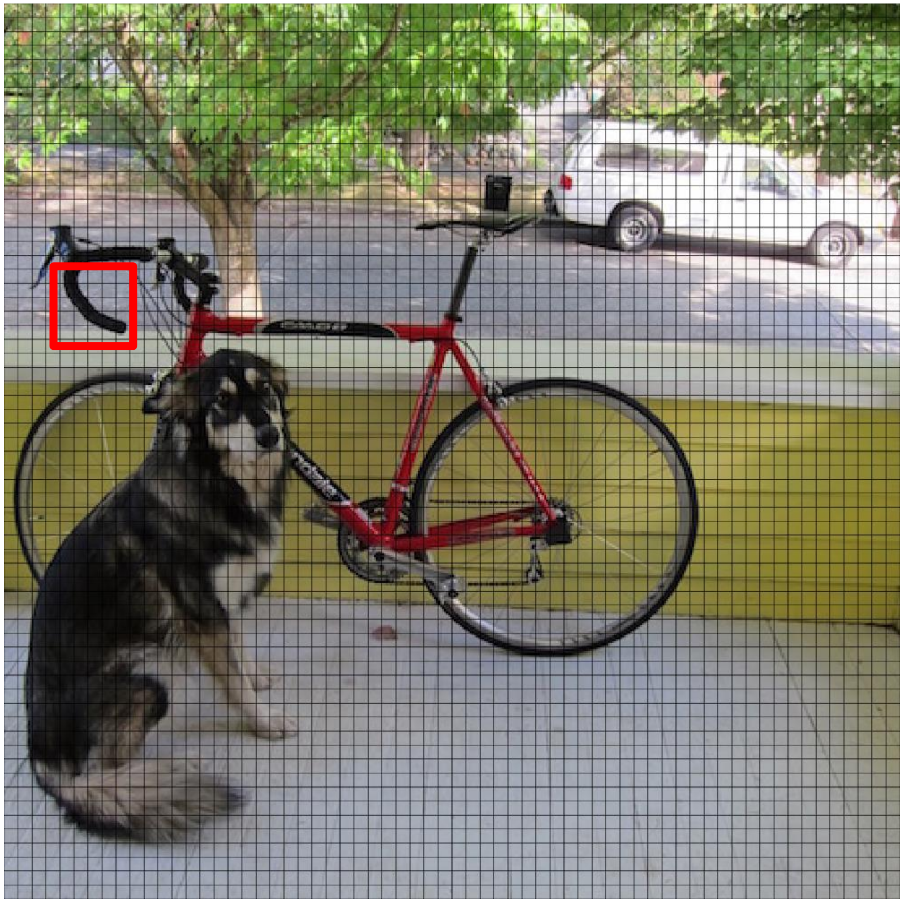
IS THIS ALL THERE IS??



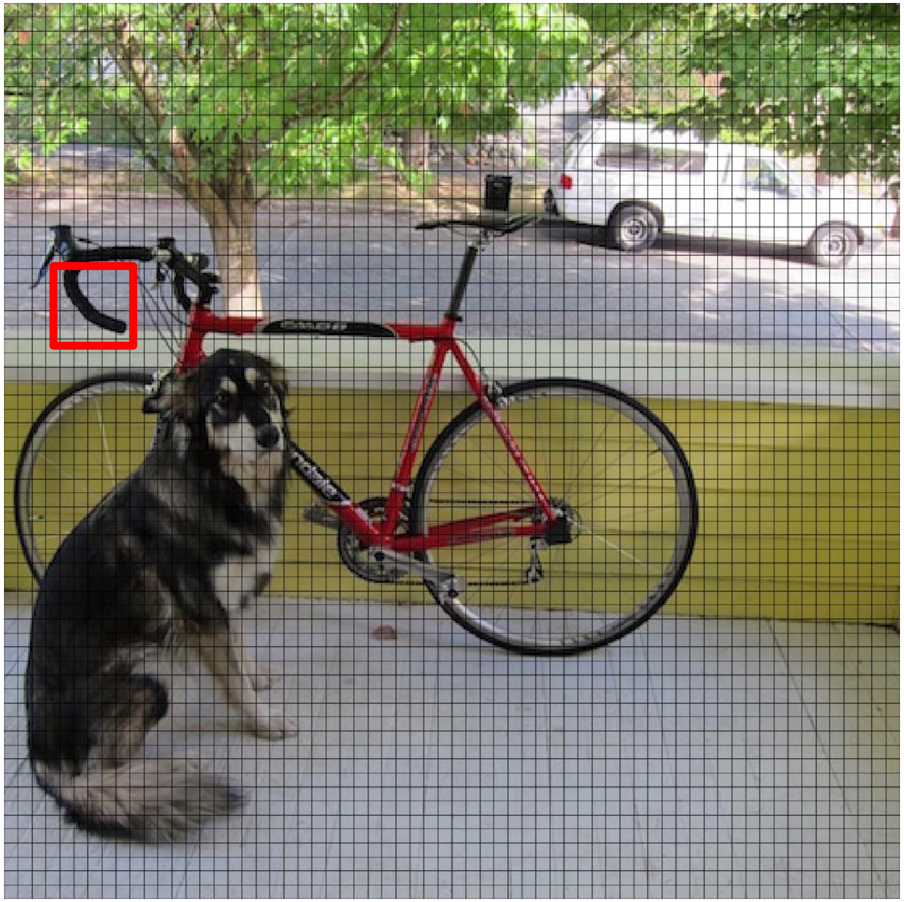
THERE IS A BETTER WAY!

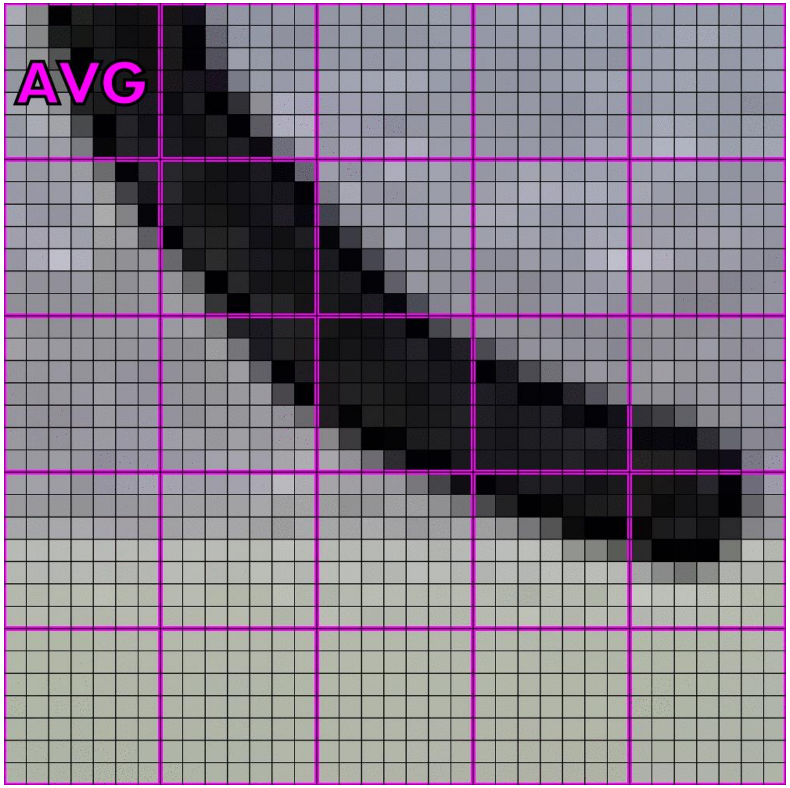
LOOK AT HOW MUCH BETTER

How do?

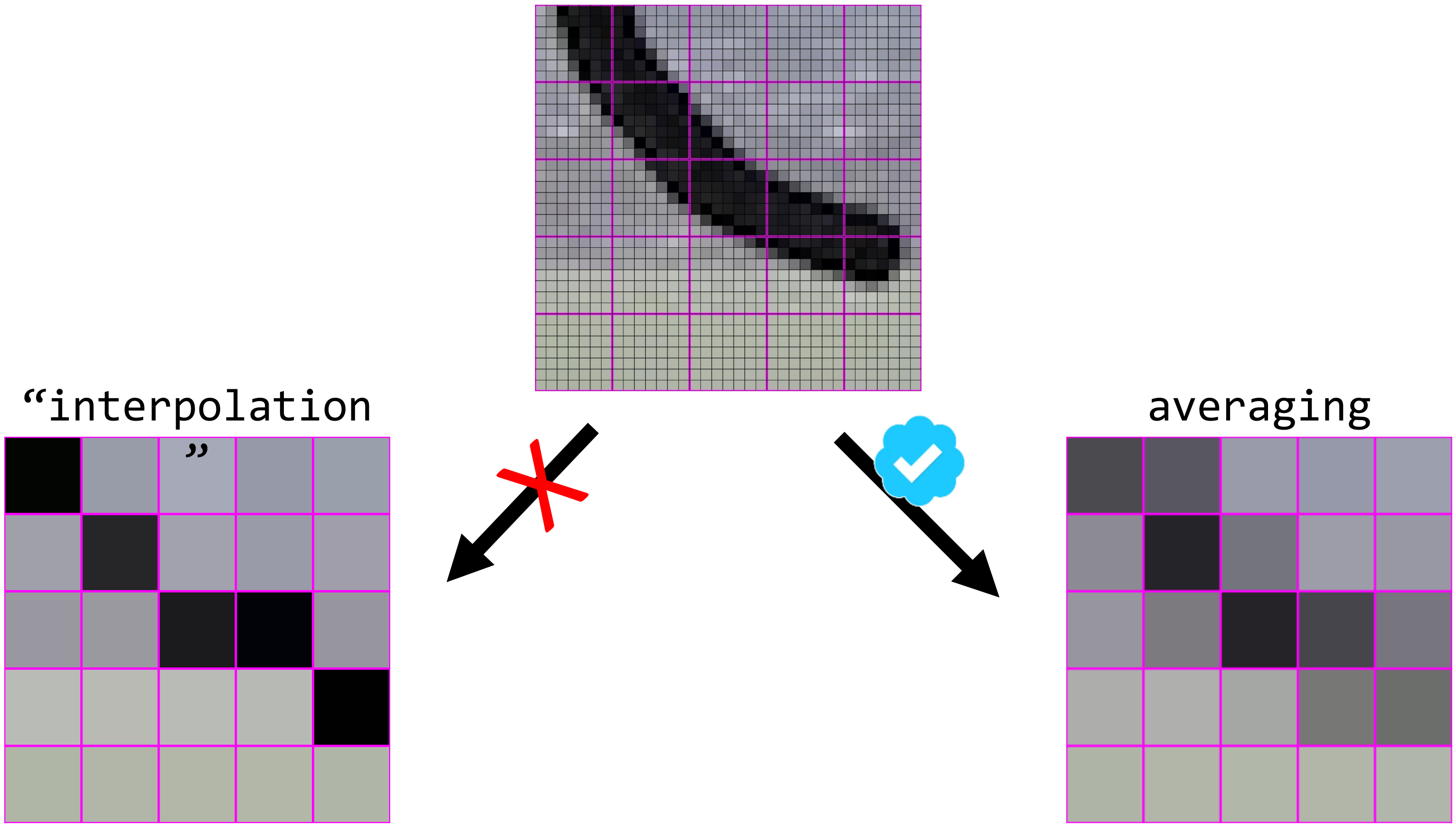


How do? Averaging!

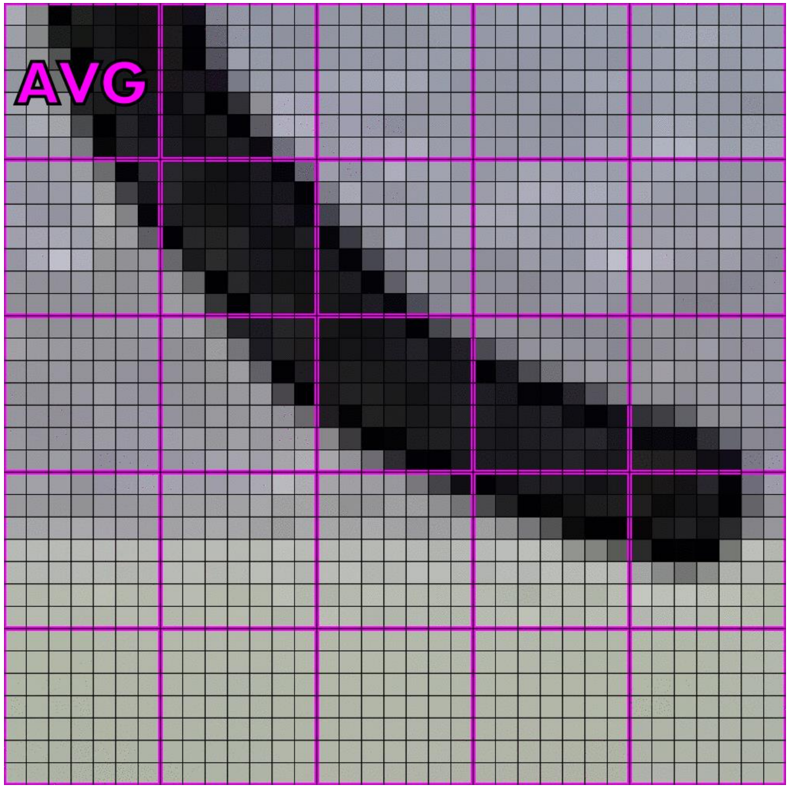




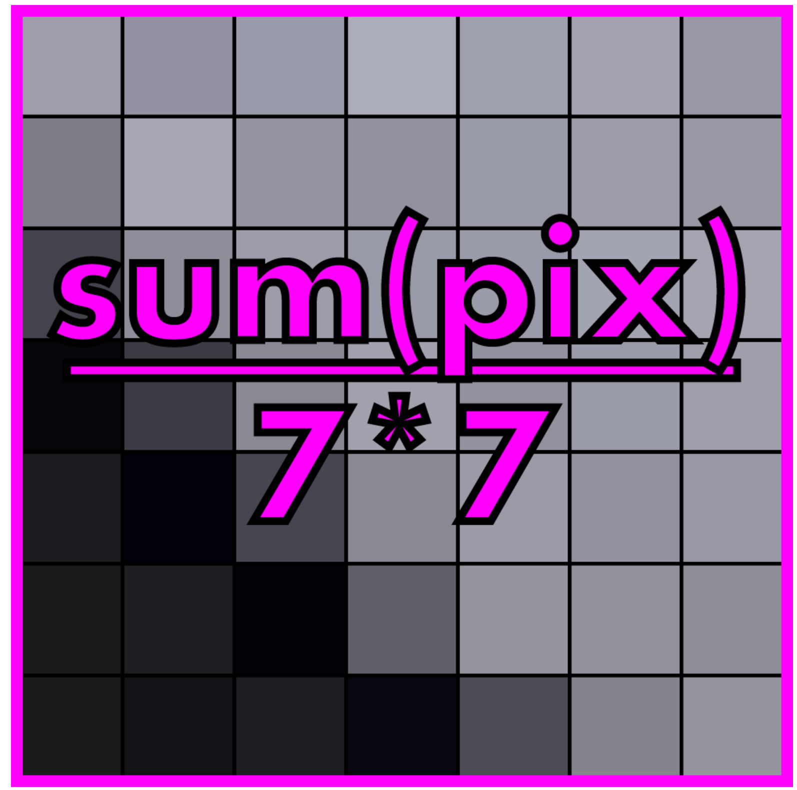
How do? Averaging!



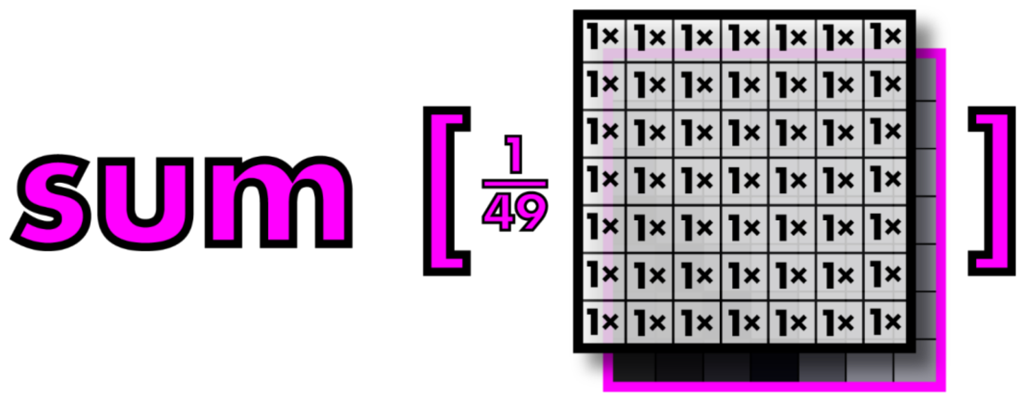
What is averaging?



What is averaging? A weighted sum

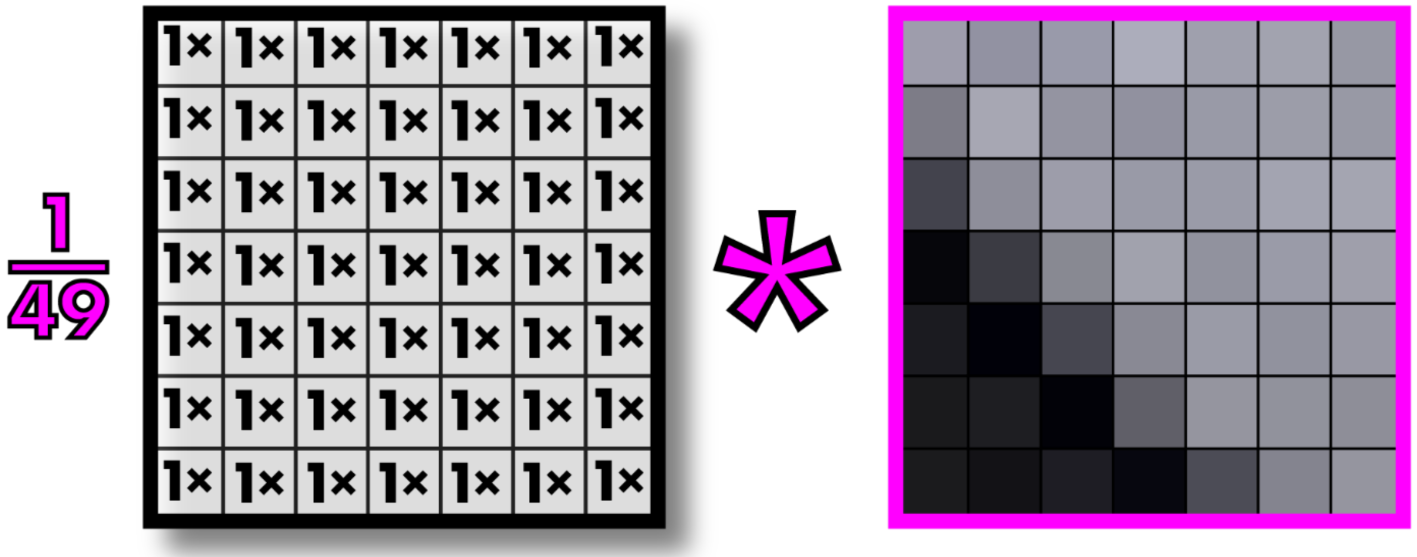


What is averaging? A weighted sum



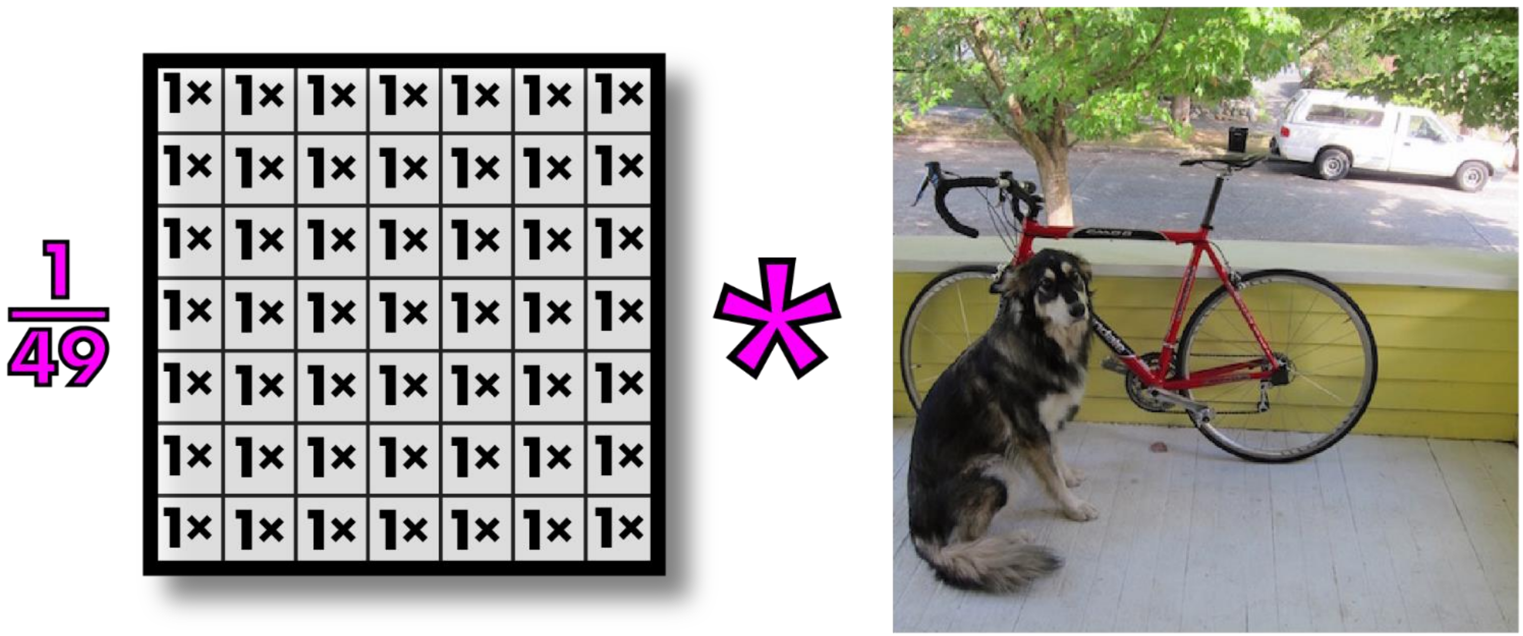
Call this operation *"convolution"*

Filter or kernel

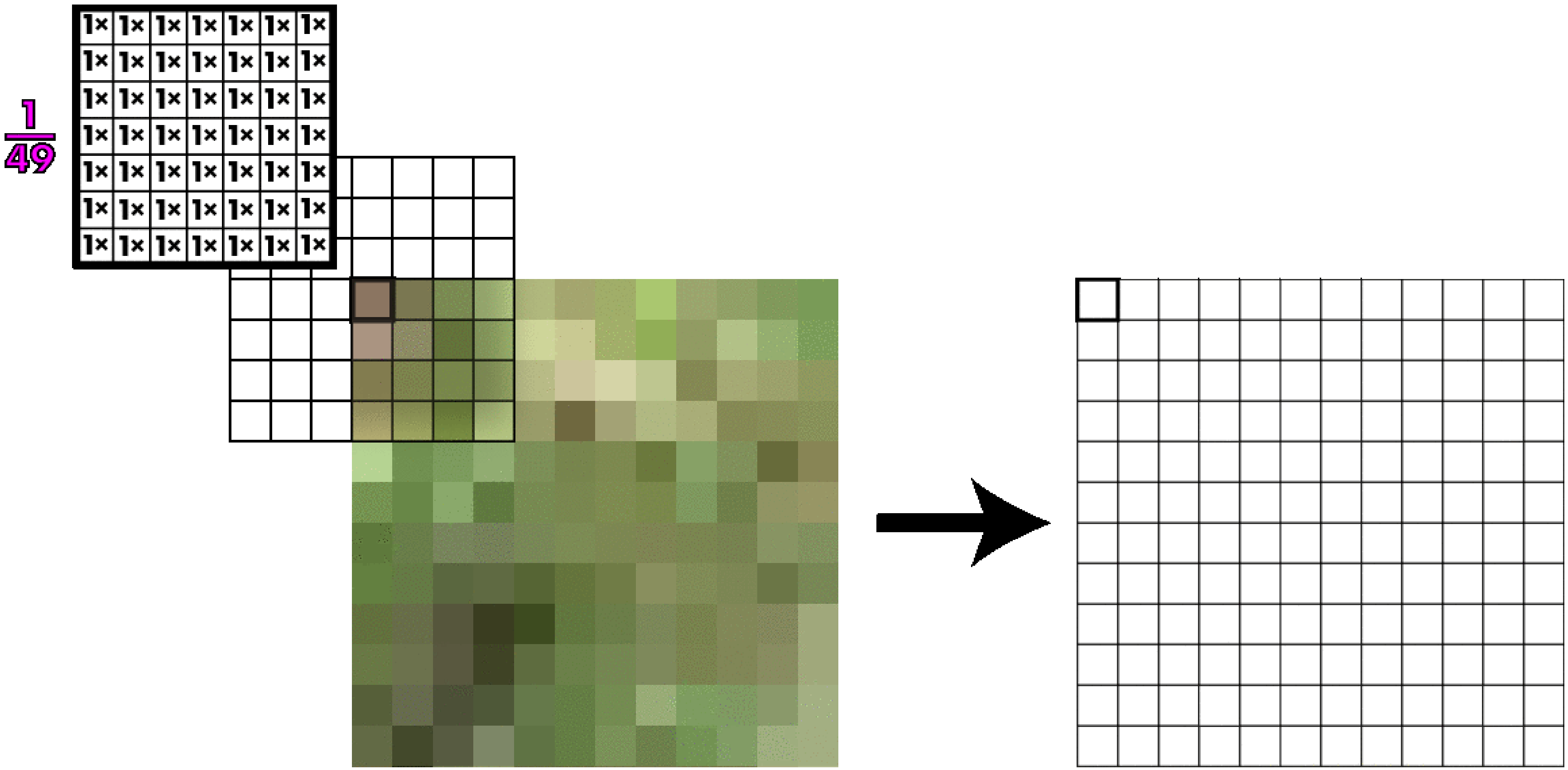


Note: multiplying an image section by a filter is actually called "correlation" and convolution Involves inverting the filter first, but since our filters are generally symmetric, we call Everything convolution. This is what all computer vision people do.

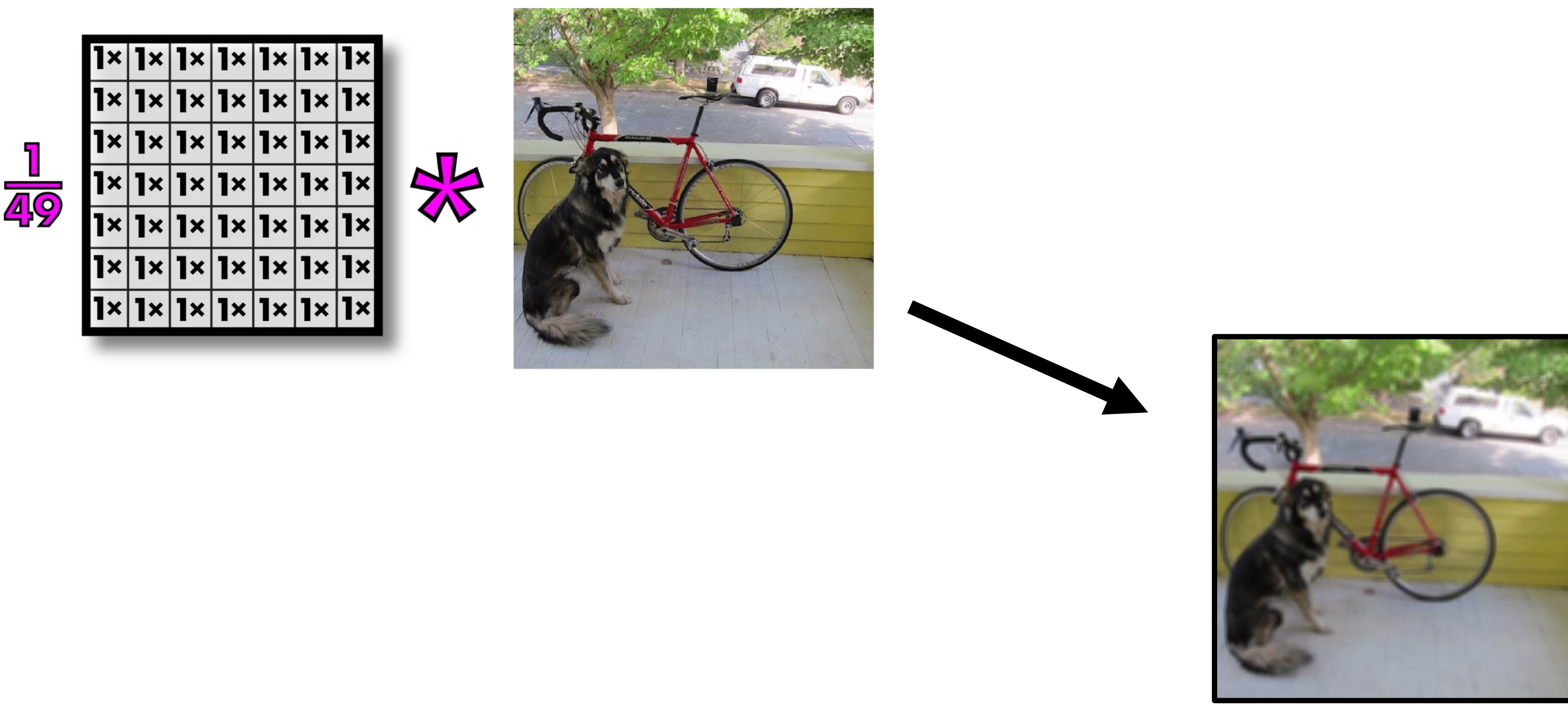
Convolutions on larger images



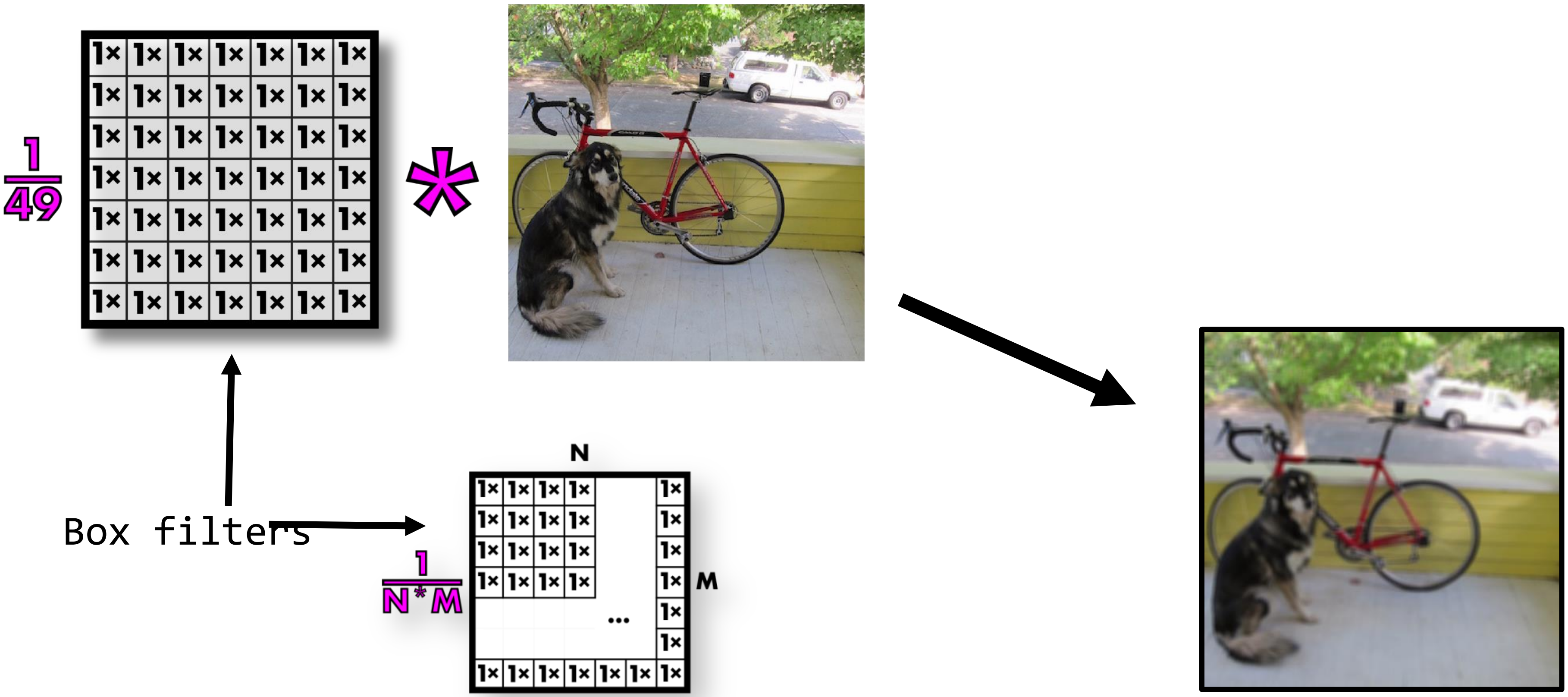
Kernel slides across image



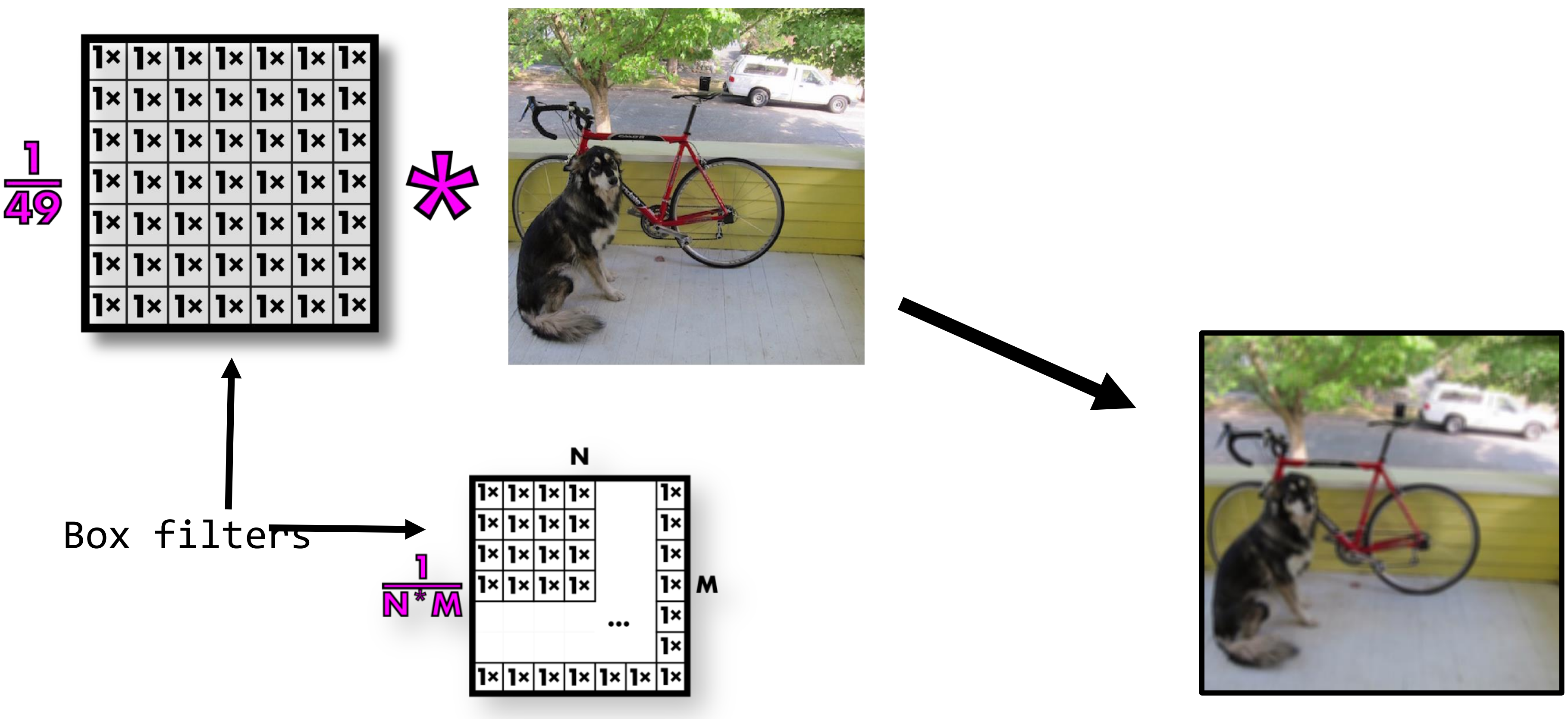
Convolutions on larger images



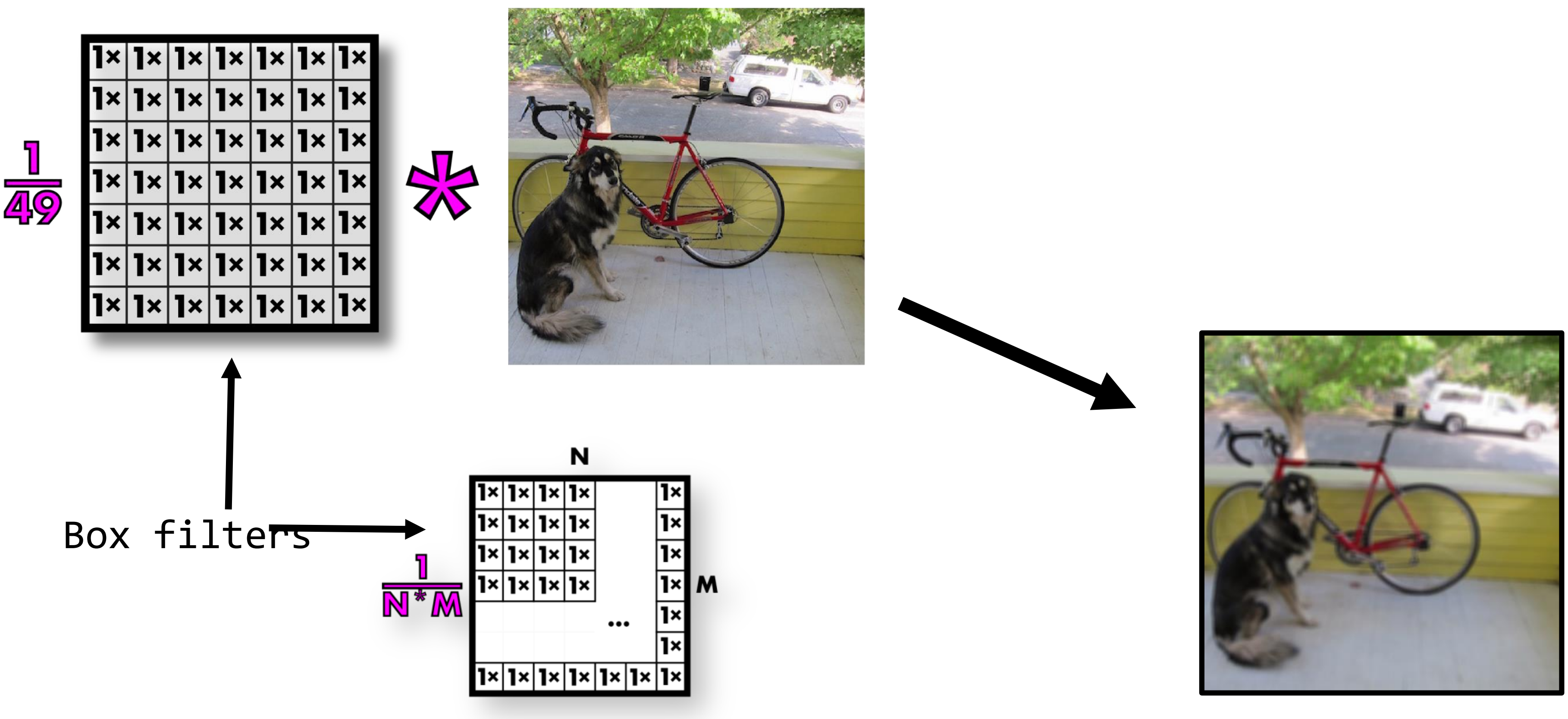
This is called box filter



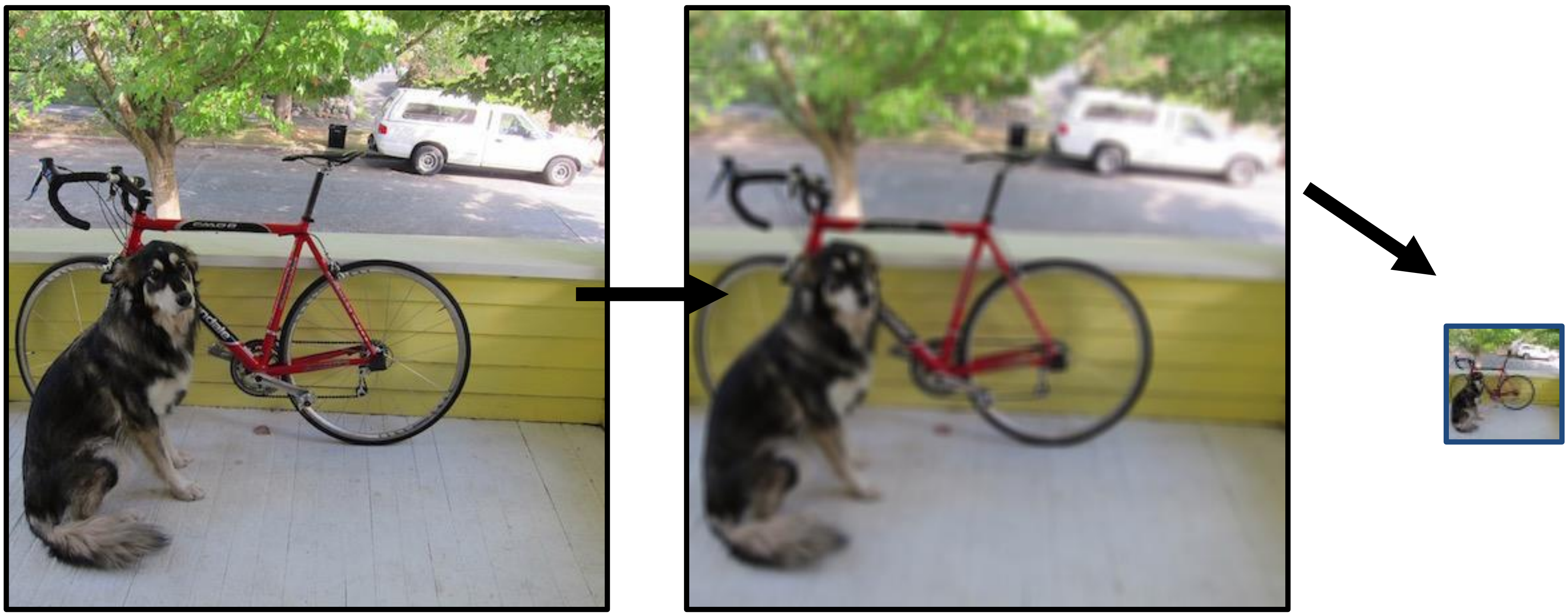
Box filters smooth image



Box filters smooth image

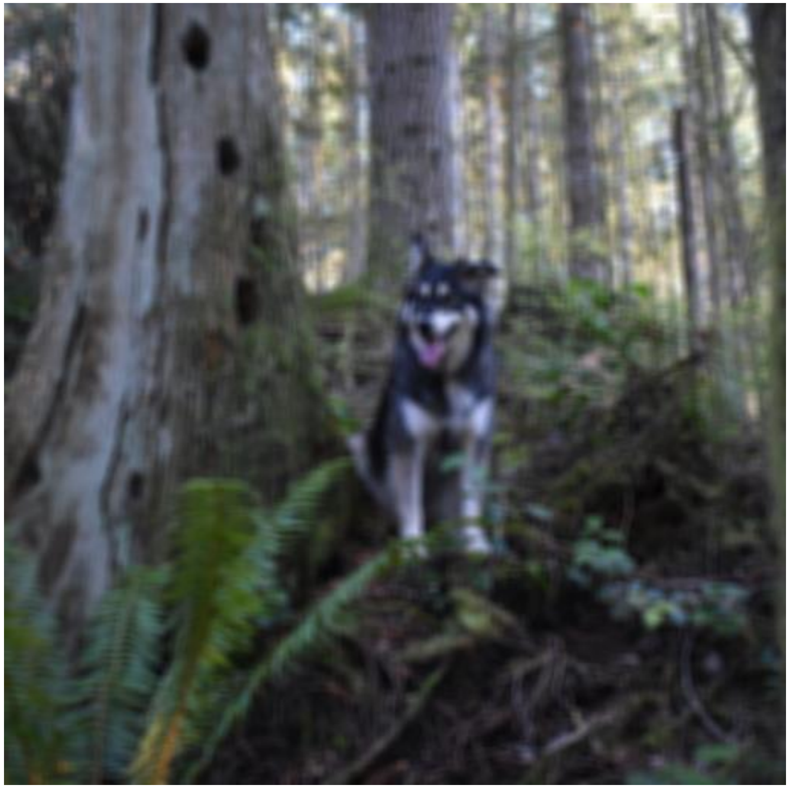


Now we resize our smoothed image



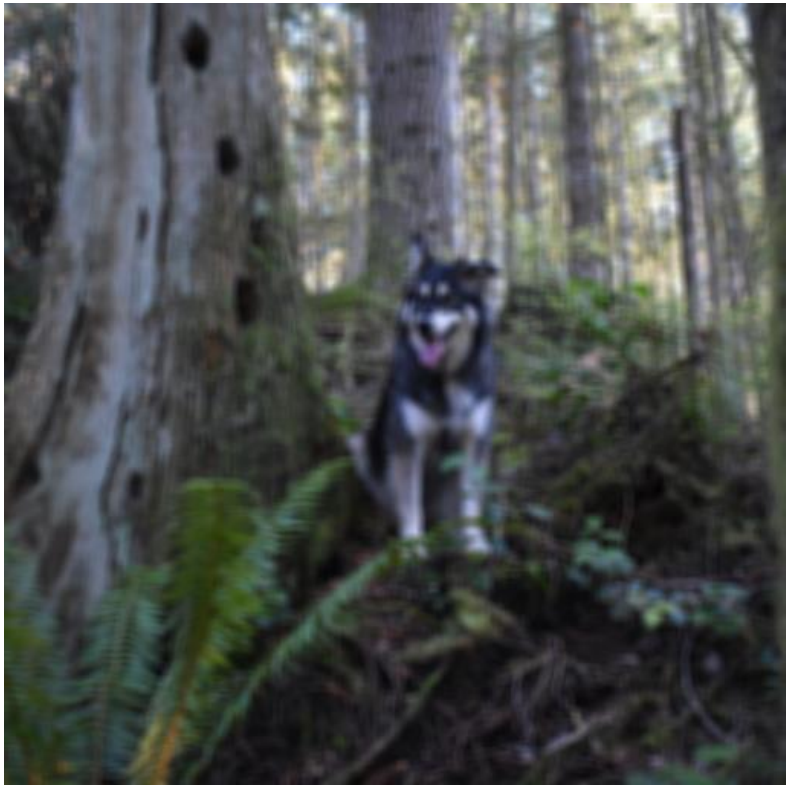
So much better!

Box filters have artifacts



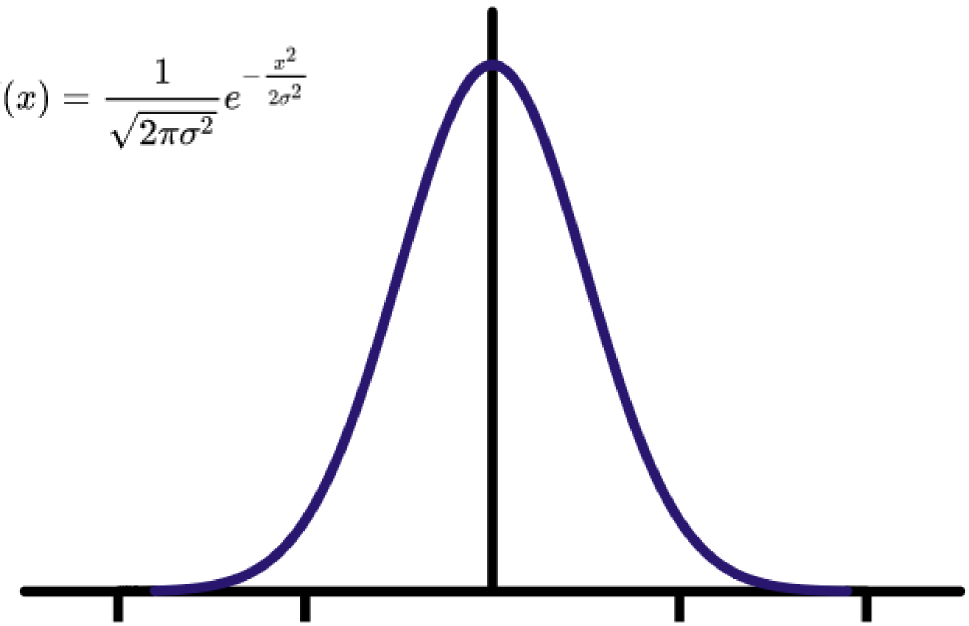
Box filters have artifacts

We want a smoothly weighted kernel



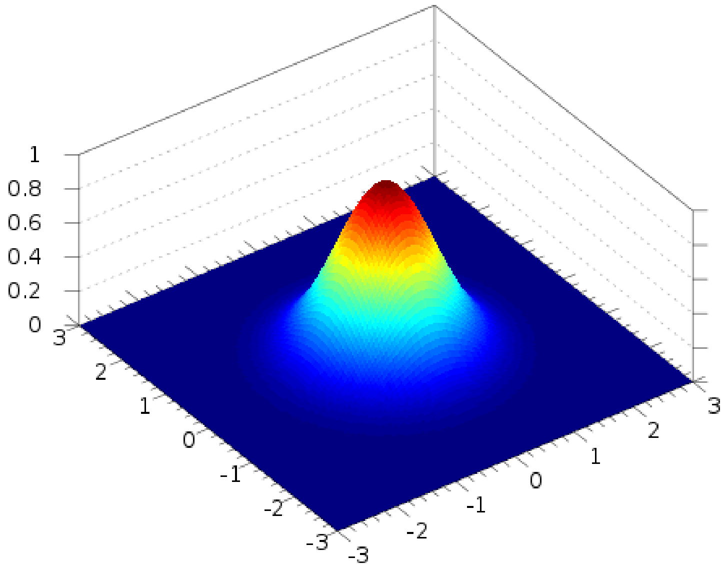
Gaussians

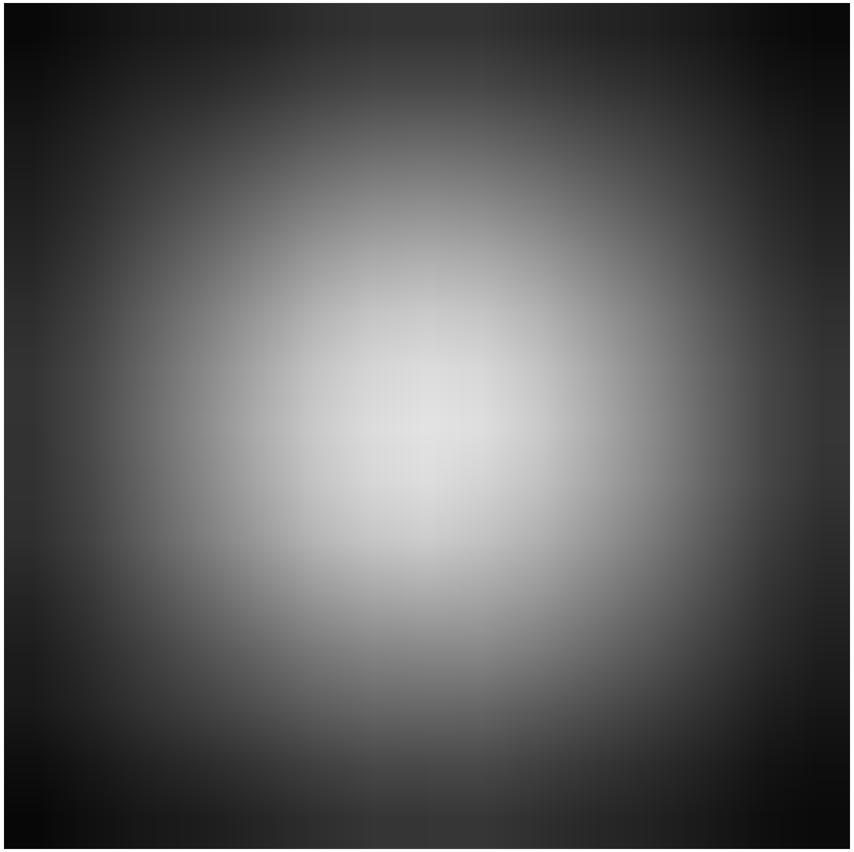
$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$



2d Gaussian

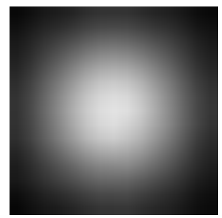
$$g(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$



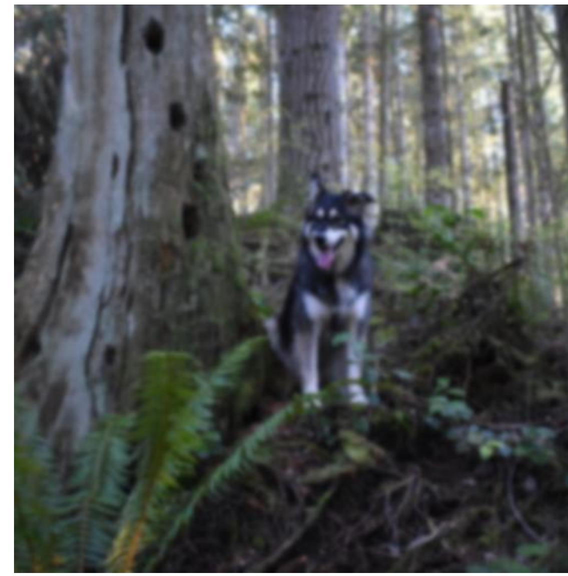


Better smoothing with Gaussians

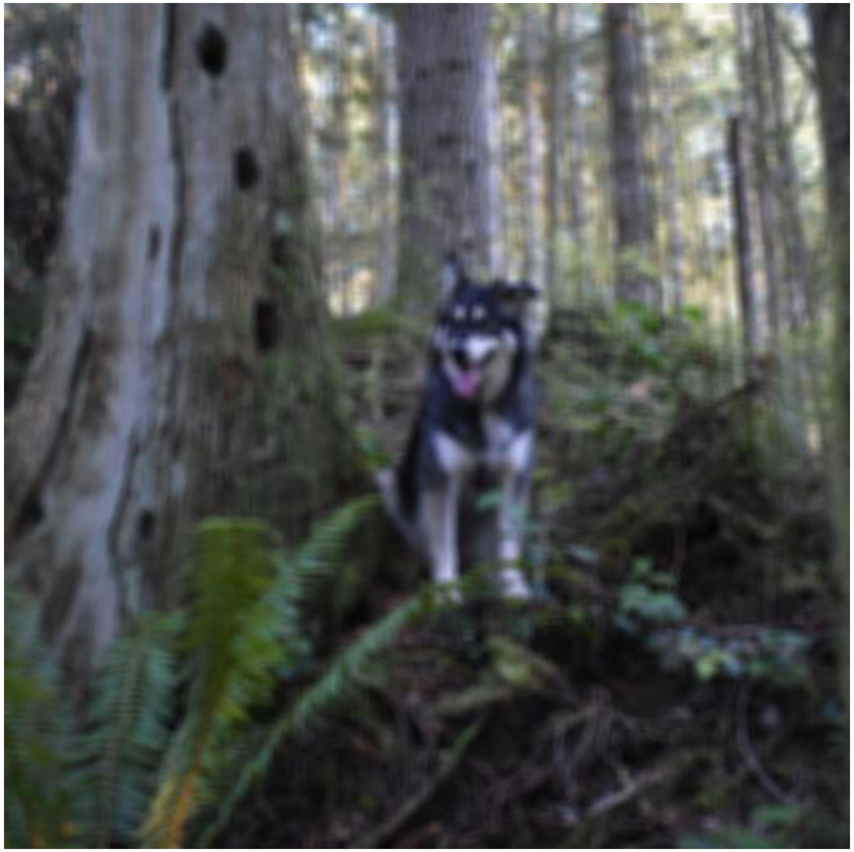
*

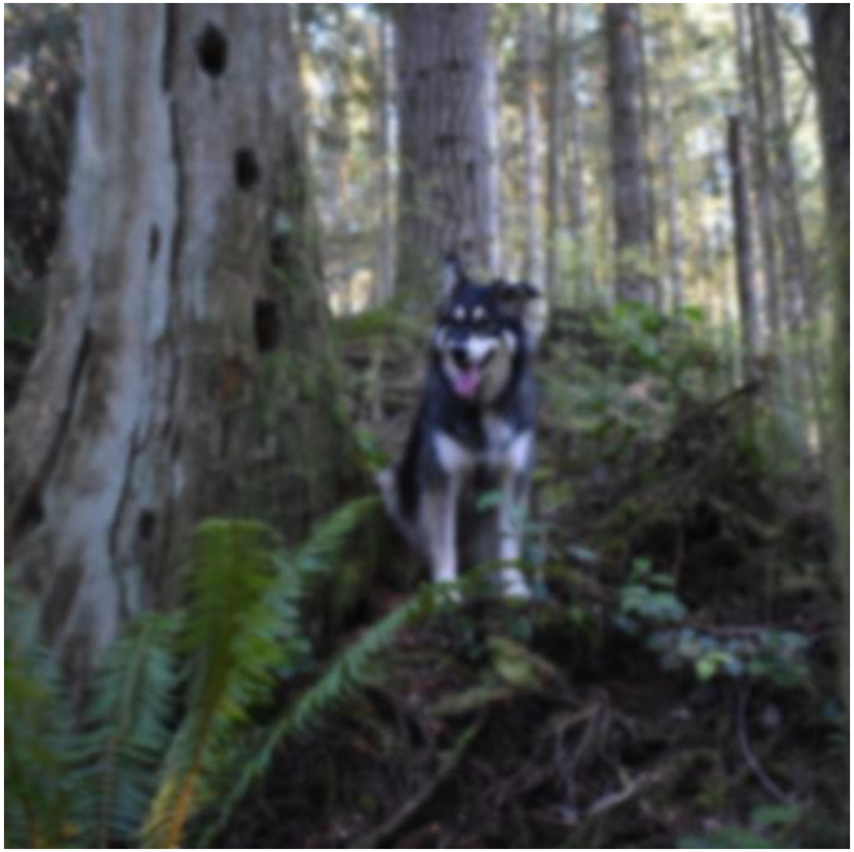


=



Better smoothing with Gaussians



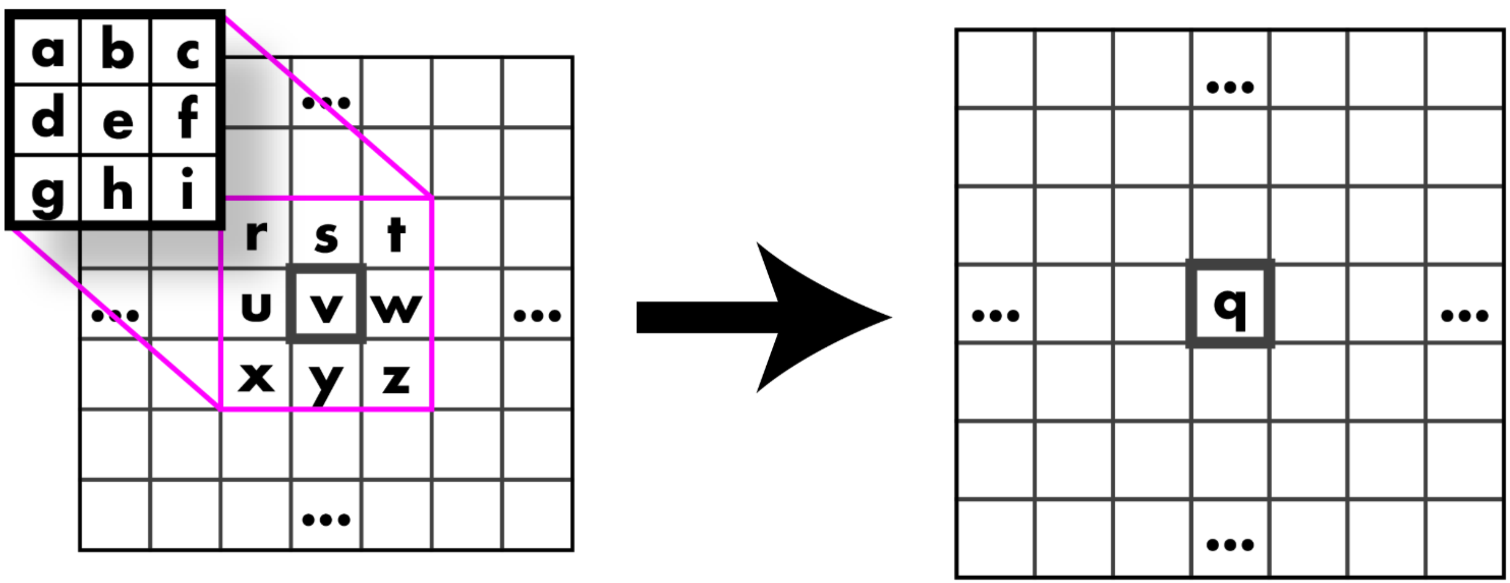


Better smoothing with Gaussians

Box Filtered

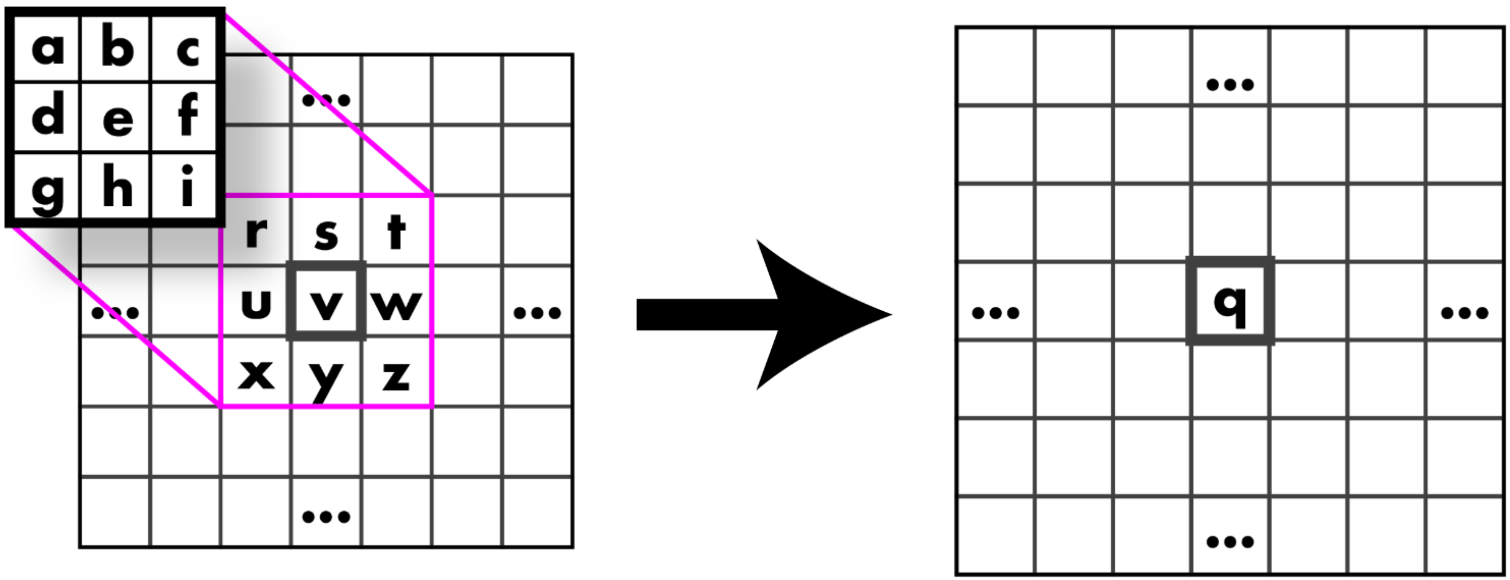
Gaussian Filtered

Wow, so what was that convolution thing??



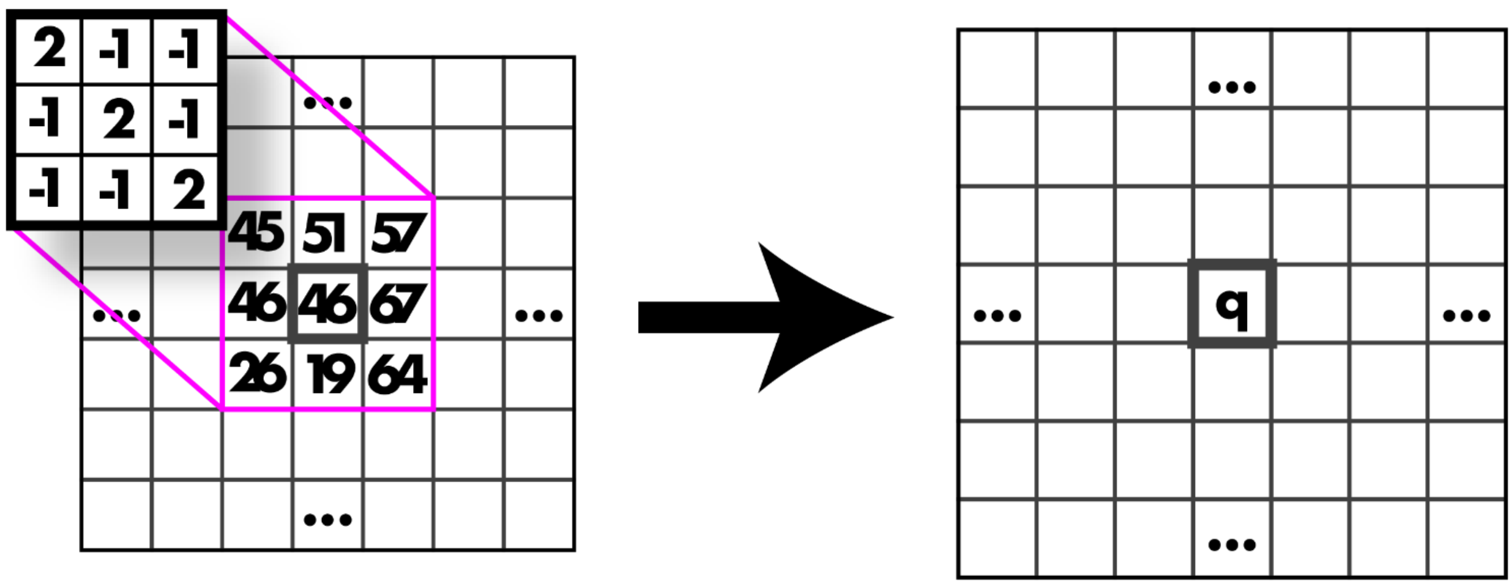
$$q = a \times r + b \times s + c \times t + d \times u + e \times v + f \times w + g \times x + h \times y + i \times z$$

Wow, so what was that convolution thing??

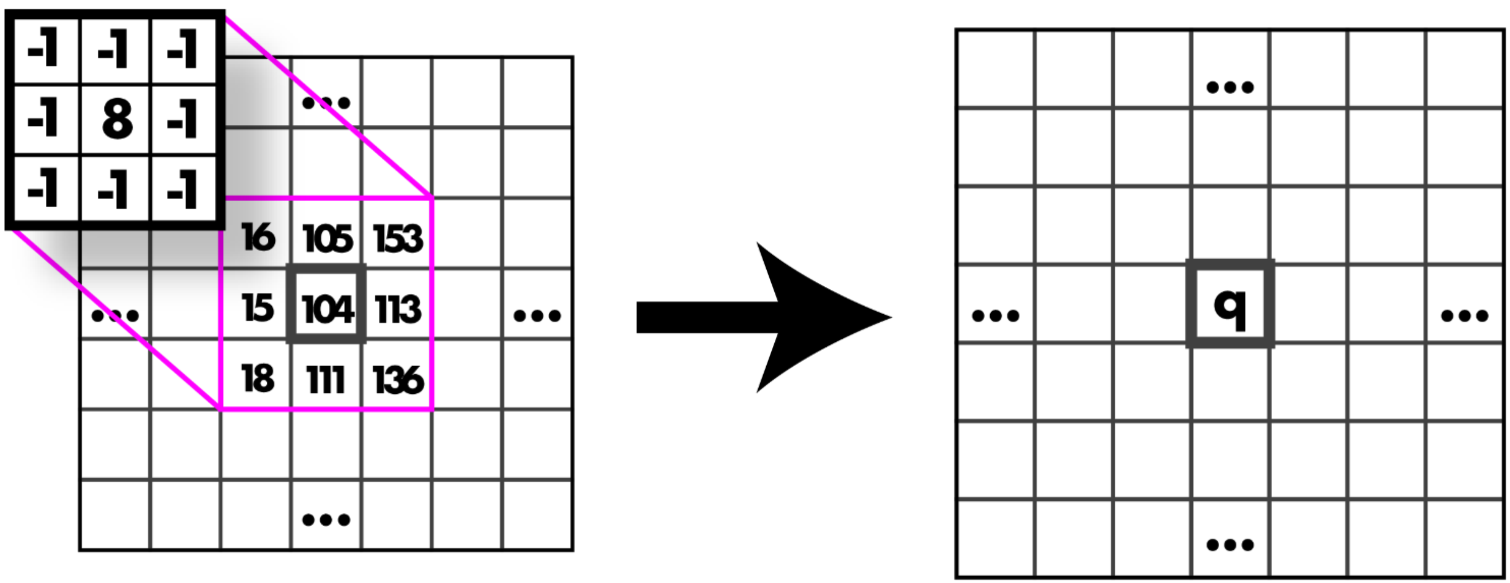


q = a×r + b×s + c×t + d×u + e×v + f×w + g×x + h×y + i×z

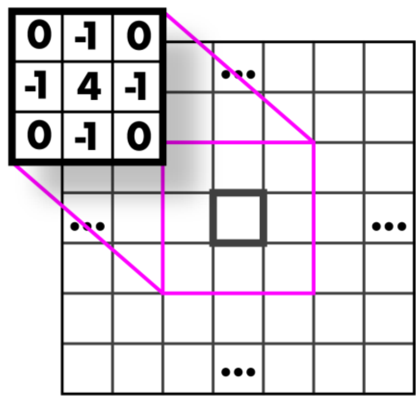
Calculate it, go!



Calculate it, go!

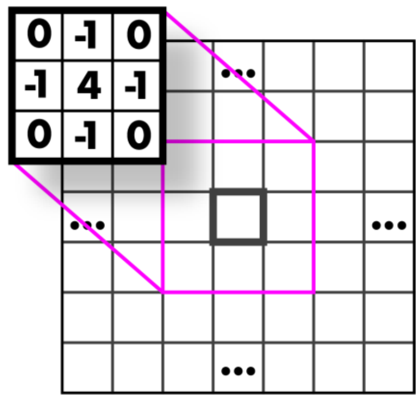


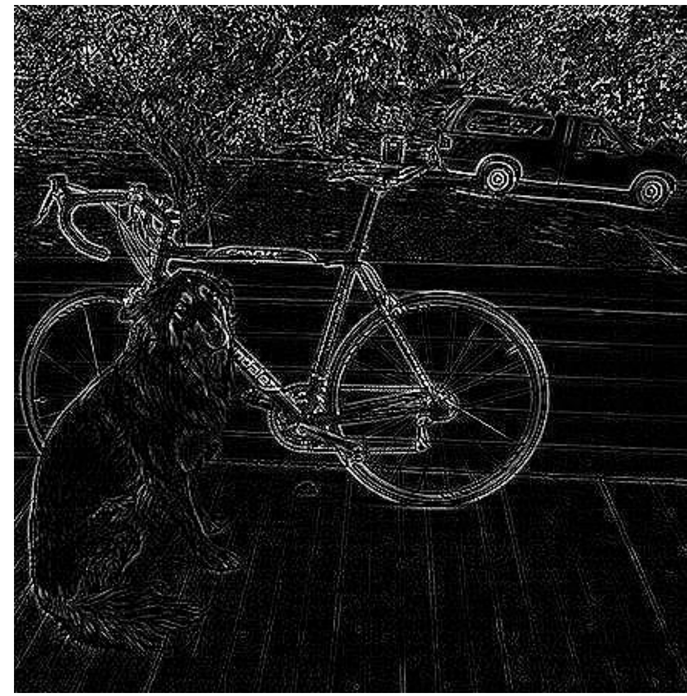
Guess that kernel!



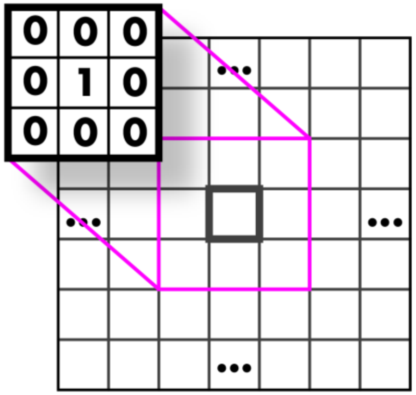
Highpass Kernel: finds edges

(applied to the graytone image!)

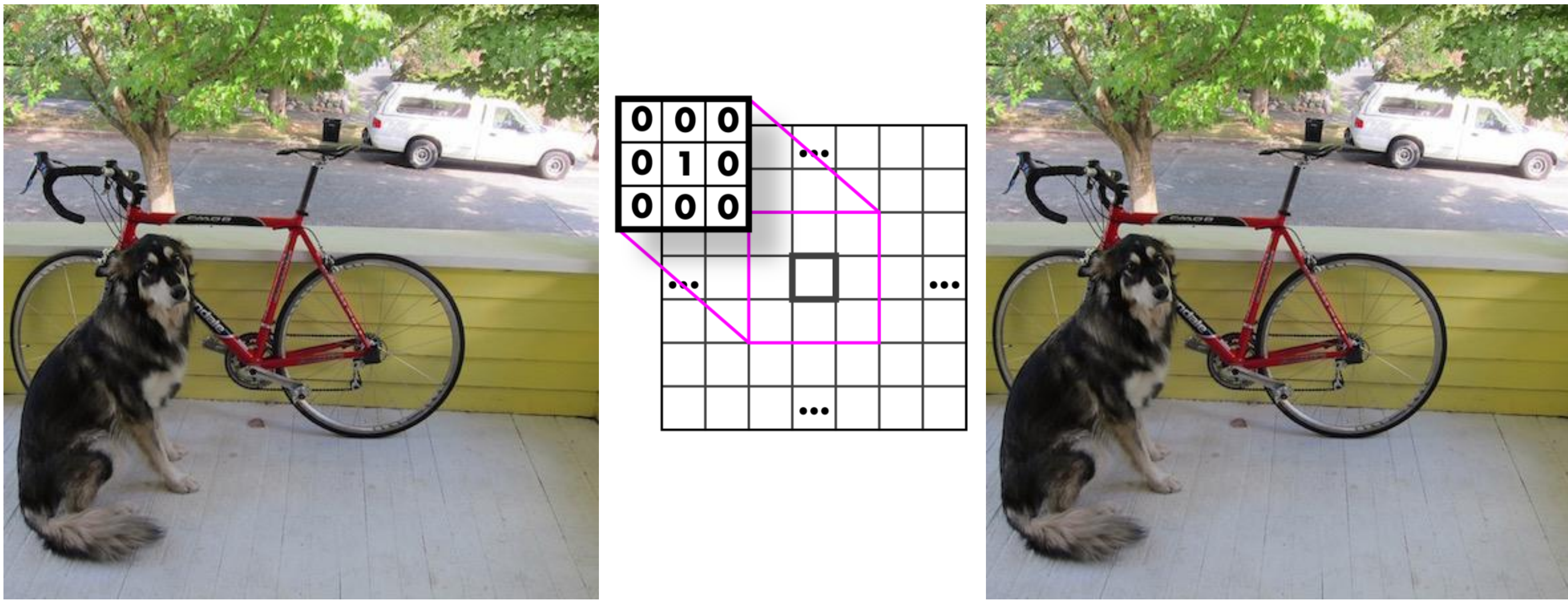




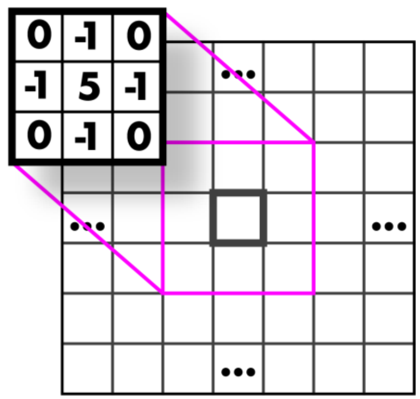
Guess that kernel!



Identity Kernel: Does nothing!

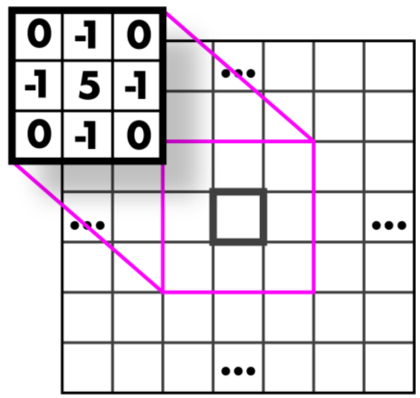


Guess that kernel!



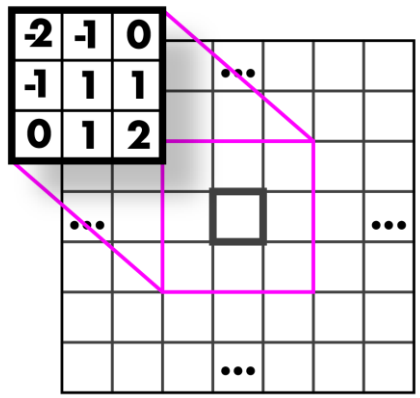
Sharpen Kernel: sharpens!

(applied to all three bands)

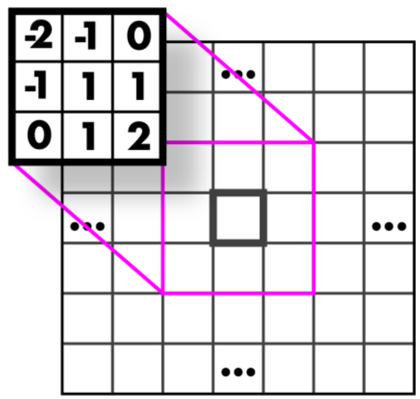


Note: sharpen = highpass + identity!

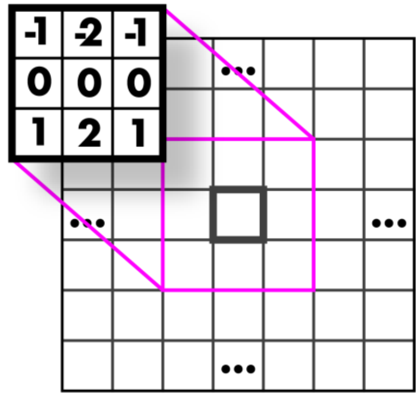
Guess that kernel!

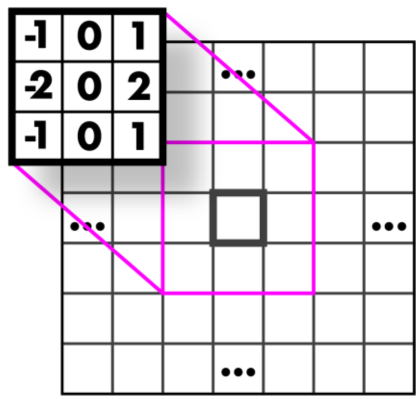


Emboss Kernel: stylin' (applied to all three bands)

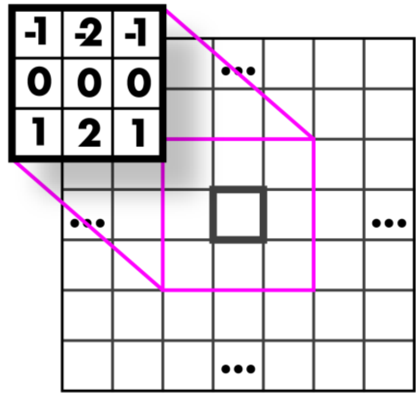


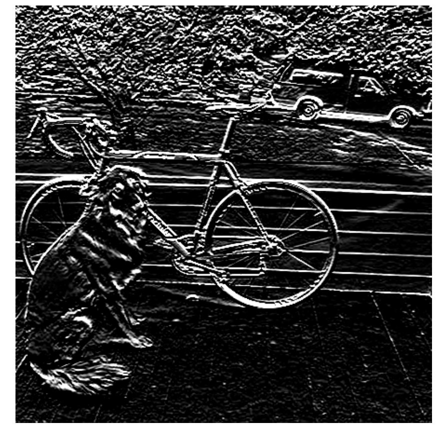
Guess those kernels!

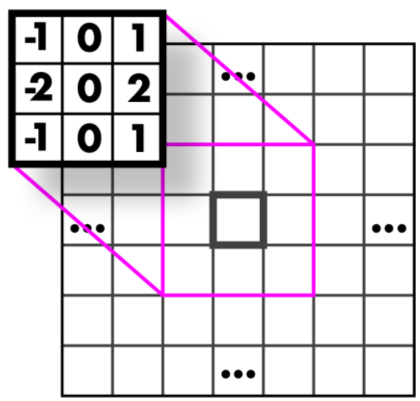




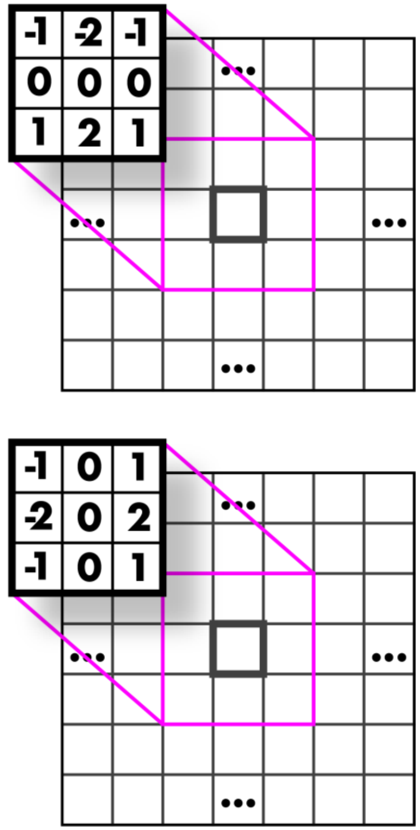
Sobel Kernels: edges (applied to a graytone image and thresholded)

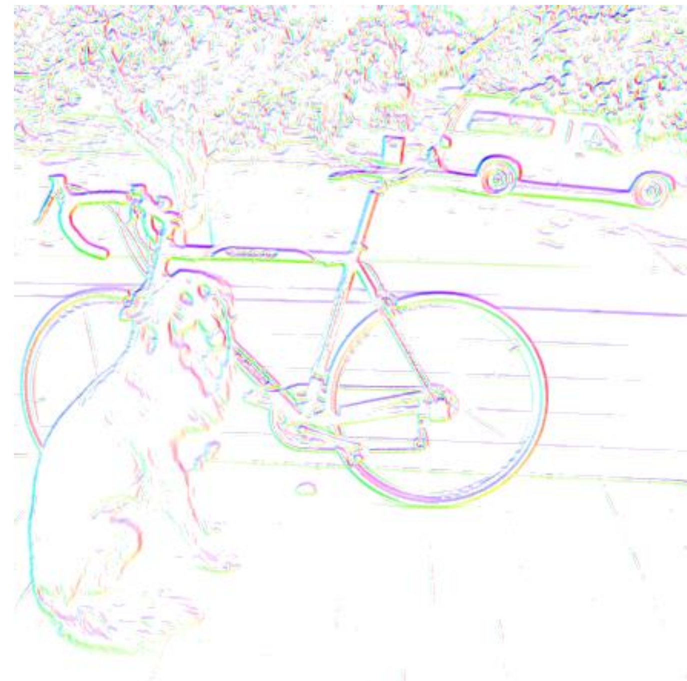




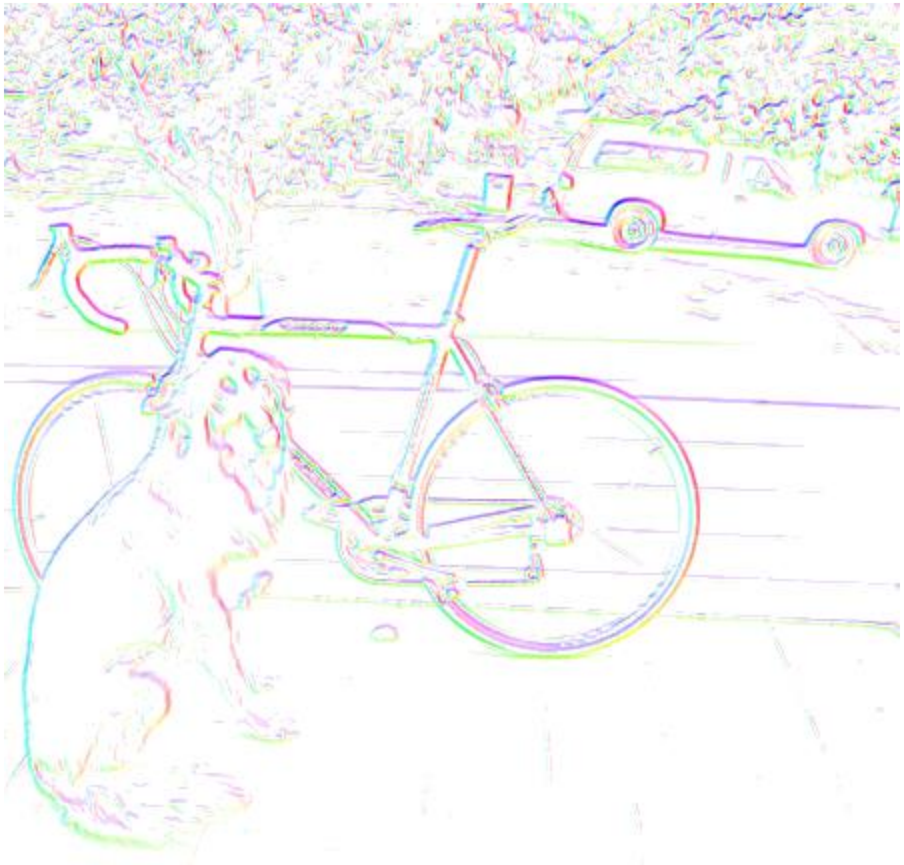


Sobel Kernels: edges and gradient!





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 1

Image resizing

and a bit of 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 hw0 in your src folder.

- Then run:
  - `make clean`
  - `make`

Assignment 1

1. Image Resizing:
   – Interpolation
     - Nearest-Neighbor (NN)
     - Bilinear
2. Image Filtering:
   – Starting out with a box filter. We'll create the box filter in this homework and will use it together with convolution function in the next homework.

To Do #1

1.1 Nearest Neighbor Interpolation

- Fill in:
  - `float nn_interpolate(image im, float x, float y, int c)`

This function performs **nearest-neighbor** interpolation on **image "im"**, given a floating **column value "x"**, **row value "y"** and integer **channel "c"**. It interpolates and returns the interpolated value.

To Do #2

1.2 Nearest Neighbor Resizing

- Fill in:
  - `image nn_resize(image im, int w, int h)`

  This function uses **nearest-neighbor** interpolation on **image "im"** to construct a new image of size **"w x h"**
  - Create **a new image** that is **"w x h"** and the **same number of channels** as "im"
  - **Loop over the pixels** and **map back** to the old coordinates.
  - Use **nearest-neighbor** interpolate to fill in the image.

To Do #2

1.2 Nearest Neighbor Resizing

- Fill in:
  - `image nn_resize(image im, int w, int h)`

You can try your function in `tryhw1.py`:

```
from uwimg import *
im = load_image("data/dogsmall.jpg")
a = nn_resize(im, im.w*4, im.h*4)
save_image(a, "dog4x-nn")
```

To Do #3

1.3 Bilinear Interpolation

- Fill in:
  - `float bilinear_interpolate(image im, float x, float y, int c)`

This function performs **bilinear** interpolation on **image "im"**, given a floating **column value "x"**, **row value "y"** and integer **channel "c"**. It interpolates and returns the interpolated value.

To Do #4

1.4 Bilinear Resizing

- Fill in:
  - `image bilinear_resize(image im, int w, int h)`

  This function uses **bilinear** interpolation on **image "im"** to construct a new image of size **"w x h”**

  - Create **a new image** that is **"w x h**" and the **same number of channels** as "im”
  - **Loop over the pixels** and **map back** to the old coordinates.
  - Use **bilinear** interpolate to fill in the image.

To Do #5

- **2.1 Create your box filter**
- Fill in:
  - `void l1_normalize(image im)`

  This function **divides each value** in an image “im” **by the sum** of all the values in the image.

To Do #6

- **2.1 Create your box filter**
- Fill in:
  - `image make_box_filter(int w)`

We will only use **square box filters**, so just make your filter w x w.

- Change the `make_image` arguments
  - image of width = height = w
  - number of channels = 1
  - all entries equal to 1.

- Then use `l1_normalize` to **normalize** your filter.

Assignment 1

- **Test your code:**
  - Use command `./main test hw1` to make sure your functions pass the tests.
  - Use python `tryhw1.py` to check out output images.

- **Turn it in:**
  - Turn in your `resize_image.c (including l1_normalize and make_box_filter`) on canvas under Homework 1.
  - Save `l1_normalize` and `make_box_filter` for your next assignment. You will need them.