Class logistics

- Quiz 2 is complete (grading still in progress)
- Project 1 is due today at 11:59pm
- Project 2 will be released tomorrow, due Feb 19th
  - Will discuss details on Wednesday
- Quiz 3 on Feb 16th

Lecture 8

Image Processing and LIDAR

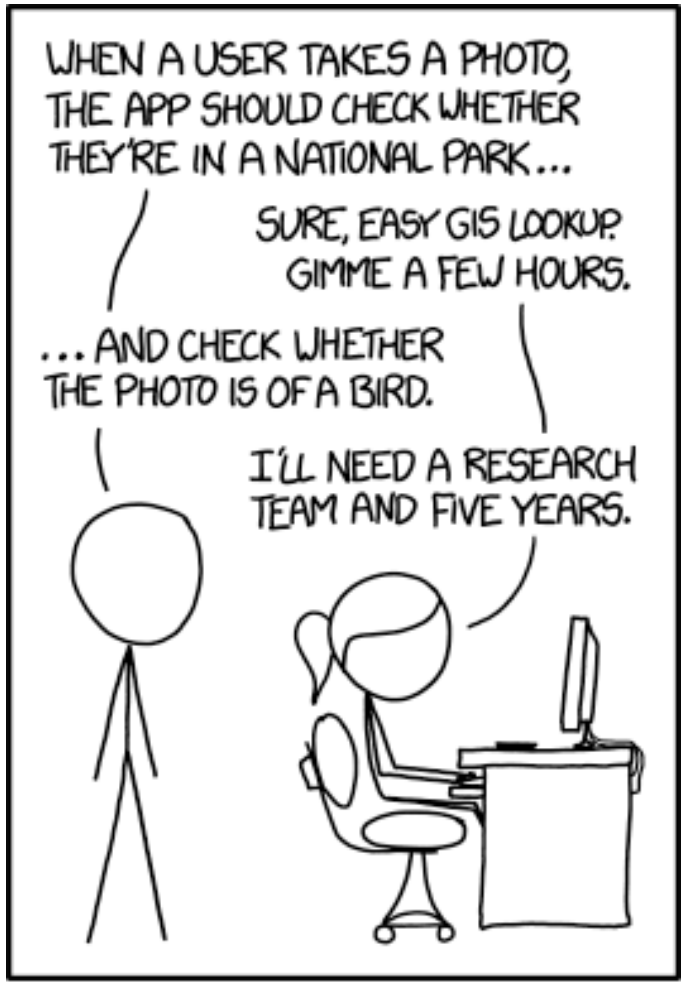
CS 3630

Basic Image Processing

Now that we know about the geometry of image formation, let's consider what information is contained in an image.

- An image is an array of pixels, which can be binary, grey-value, or color.
- Image filtering takes an image as input and performs basic computation at each pixel to construct an output image.
- Image filters can be used for:
  - Increasing contrast
  - Removing noise
  - Finding edges
- Basic image processing is a precursor for image understanding (aka perception).

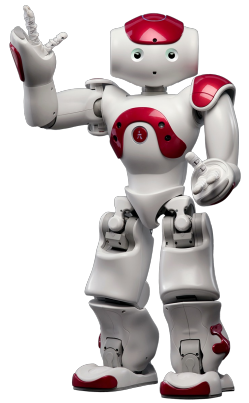
Image Processing ...



Alt Text:
In the 60s, Marvin Minsky assigned a couple of undergrads to spend the summer programming a computer to use a camera to identify objects in a scene. He figured they'd have the problem solved by the end of the summer. Half a century later, we're still working on it.

Source: https://xkcd.com/1425/ (2014)

Cameras are the primary sensor for most robotic platforms



Common camera types

For now we'll focus here

Single view
RGB image
(2D data)

Stereo
RGB-D image
(3D data)

Structured light or time of flight
RGB-D image
(3D data)

Images

- An image is basically a 2D array of intensity/color values
- Image types:

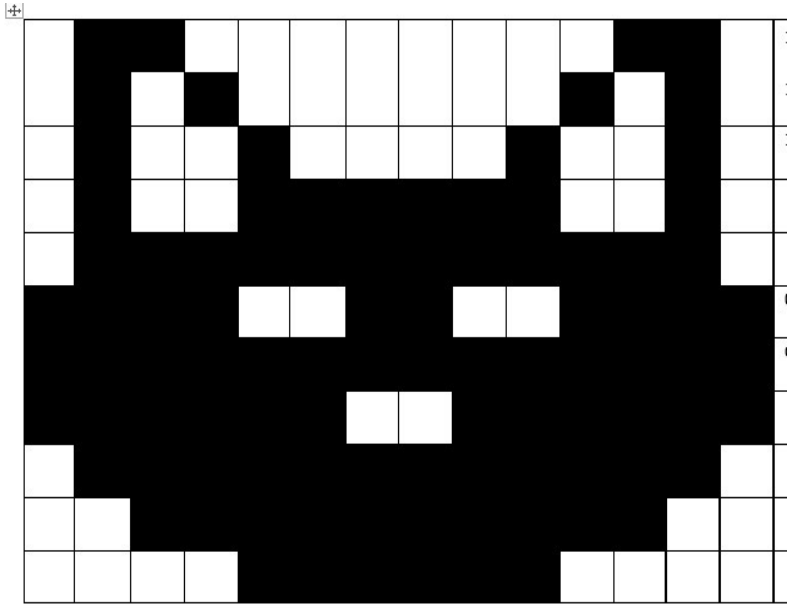
Color

Grayscale

B-W

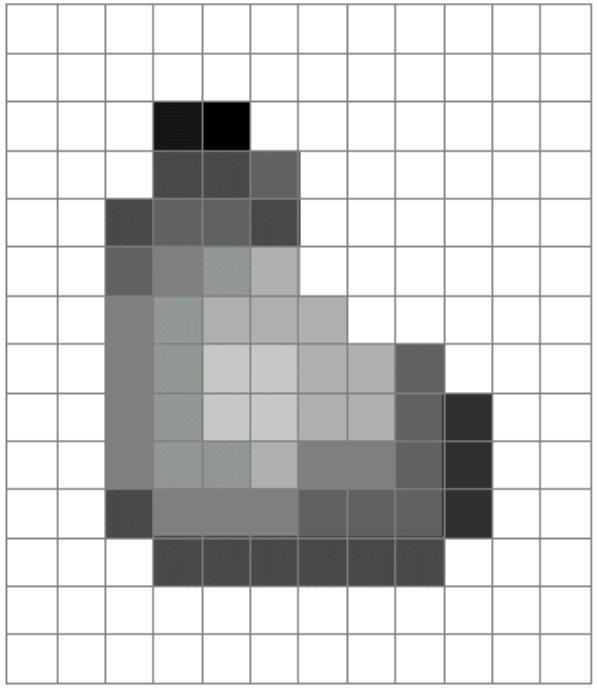
B-W Images

- A grid (matrix) of 1's and 0's



Grayscale Images

A grid (matrix) of intensity values



=

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

(common to use one byte per value: 0 = black, 255 = white)

Color Images

Three grids (matrices) of intensity values - [R,G,B]

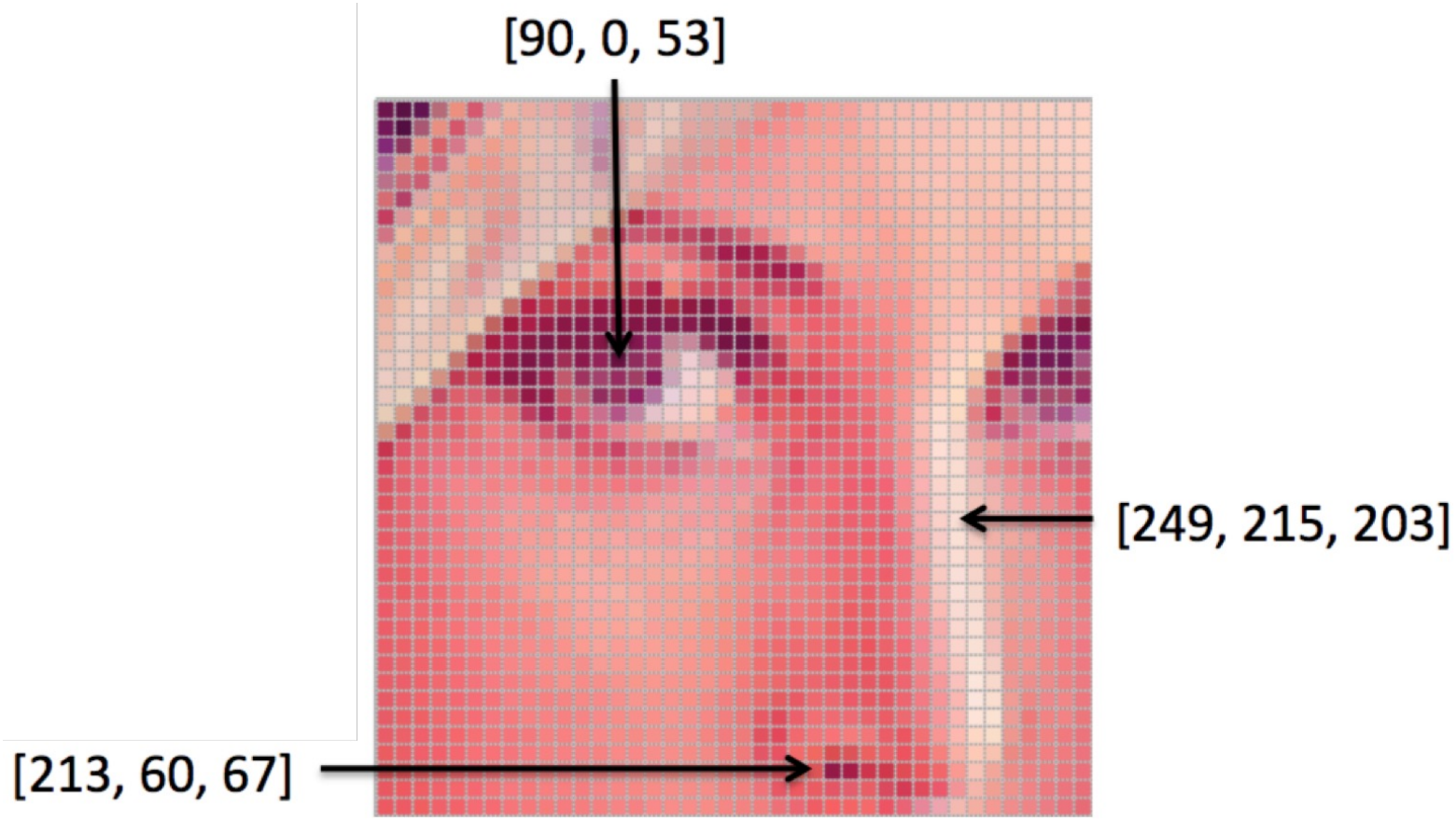


Image Classification

- One of the primary roles of vision in robotics is to recognize entities in images: people, cars, objects, etc...

- For now, we will discuss image processing in the context of image classification:

  - The problem of identifying whether an image belongs in one of some number of previously known categories

Image Adjustments...

- Scaling
- Color manipulation (color space, color->grayscale, etc.)
- Contrast
- Exposure
- ...

Image sampling

640x640

320x320

160x160

80x80

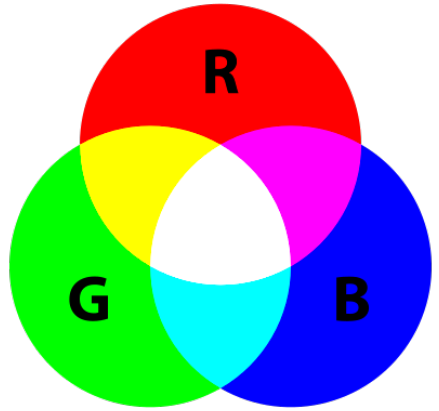
40x40

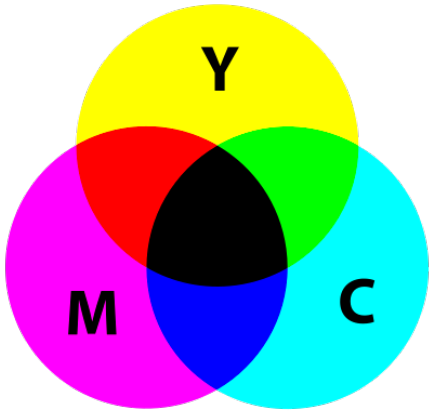
20x20

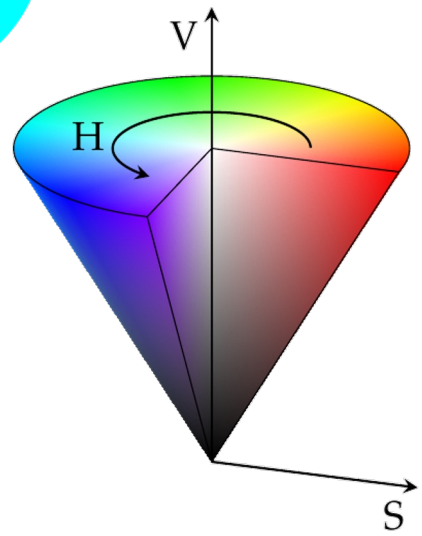
Color Models

A color model is an abstract mathematical model describing the way colors can be represented as tuples of numbers, typically as three or four values or color components.

- RGB – Red, green, blue
- CMYK – cyan, magenta, yellow, black
- HSV – hue, saturation, value







- No one color model is always "better" than another.
- For specific applications, one model might be more suitable than another.
  - e.g., HSV would likely work better when looking for a dark object on top of light background

Explore color models: http://colorizer.org/

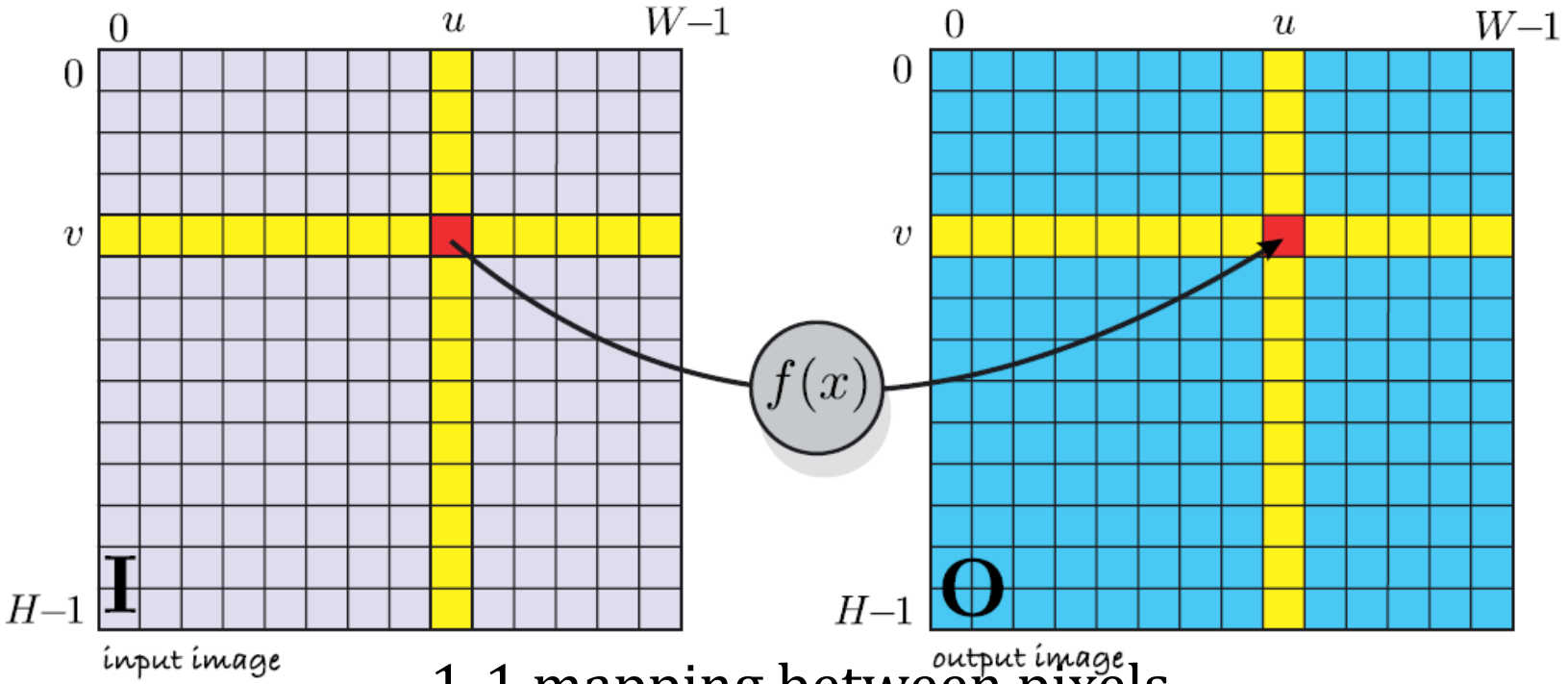
Filtering

- Types of filtering:
  - Noise removal
  - Edge detection
  - Texture description
  - Multi-scale algorithms
  - Feature detection
  - Matched filters
  - …

- We will only focus on a few specific methods useful for our application

Monadic operators for filtering

- Operations that take a single pixel as input and output, and do not consider neighboring pixel values



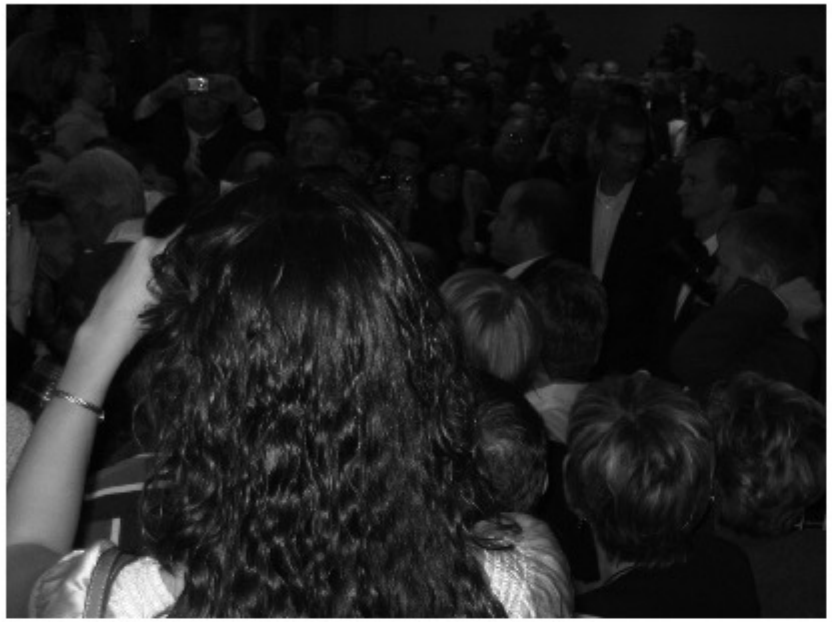
1-1 mapping between pixels

Example: Sepia Tone

$$\begin{aligned}
\text{outRed} &= (\text{inRed} * .393) + (\text{inGreen} * .769) + (\text{inBlue} * .189) \\
\text{outGreen} &= (\text{inRed} * .349) + (\text{inGreen} * .686) + (\text{inBlue} * .168) \\
\text{outBlue} &= (\text{inRed} * .272) + (\text{inGreen} * .534) + (\text{inBlue} * .131)
\end{aligned}$$

Example: Histogram Normalization

Input Image

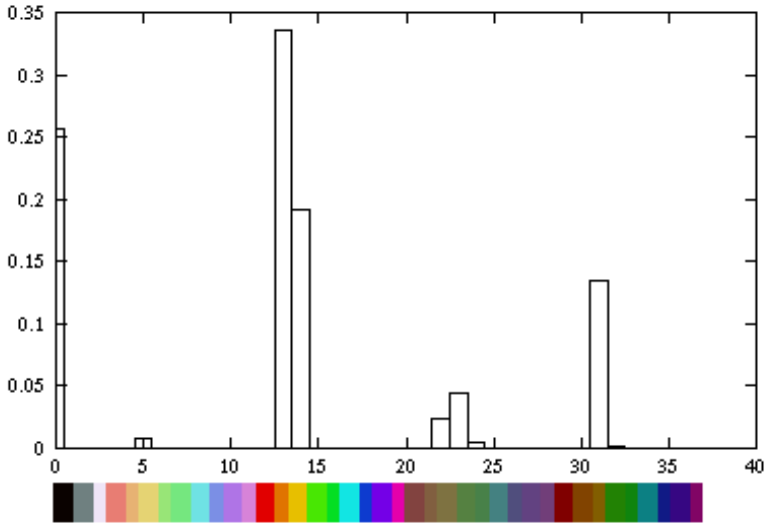


Output Image

What is an image histogram?

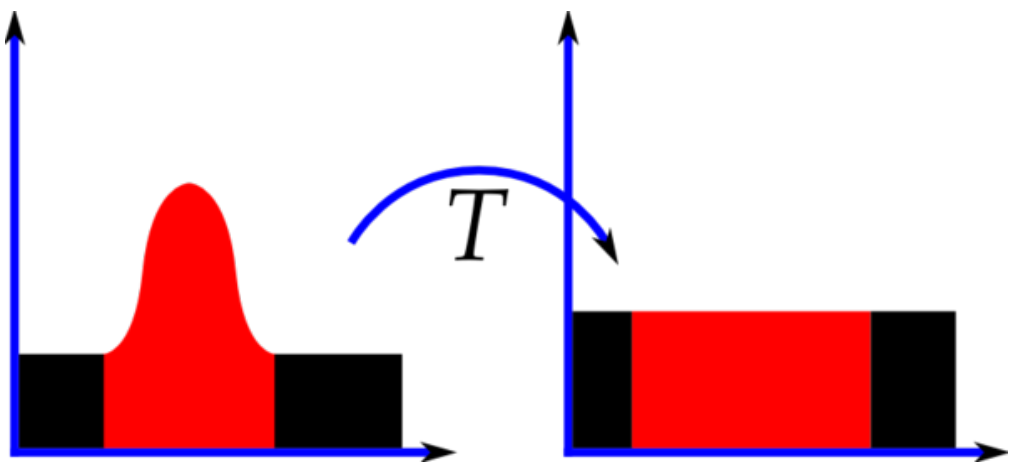
For each color C_i, $\mathcal{H}_{ci}(img)$ provides the number of pixels of color C_i in *img*.

More generally, a color histogram represents the distribution of various colors (or *intensities* if grayscale) in the image.

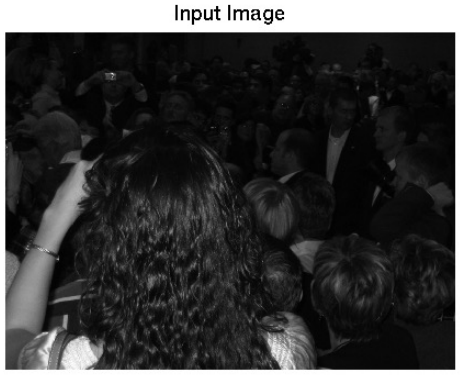


Histogram filter

- A histogram can be used to adjust or redistribute the intensity or color distribution of an image
- Usually increases the contrast of an image by effectively spreading out the most frequent intensity values
- Useful in images with backgrounds and foregrounds that are both bright or both dark



Histograms of an image before and after equalization.



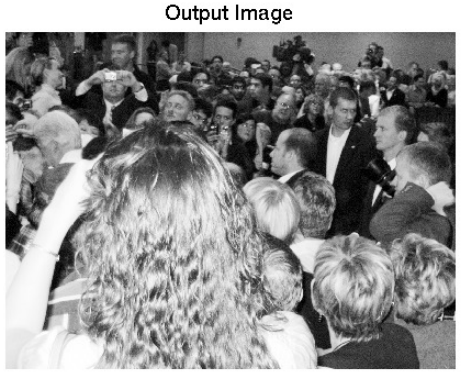


Image Contrast

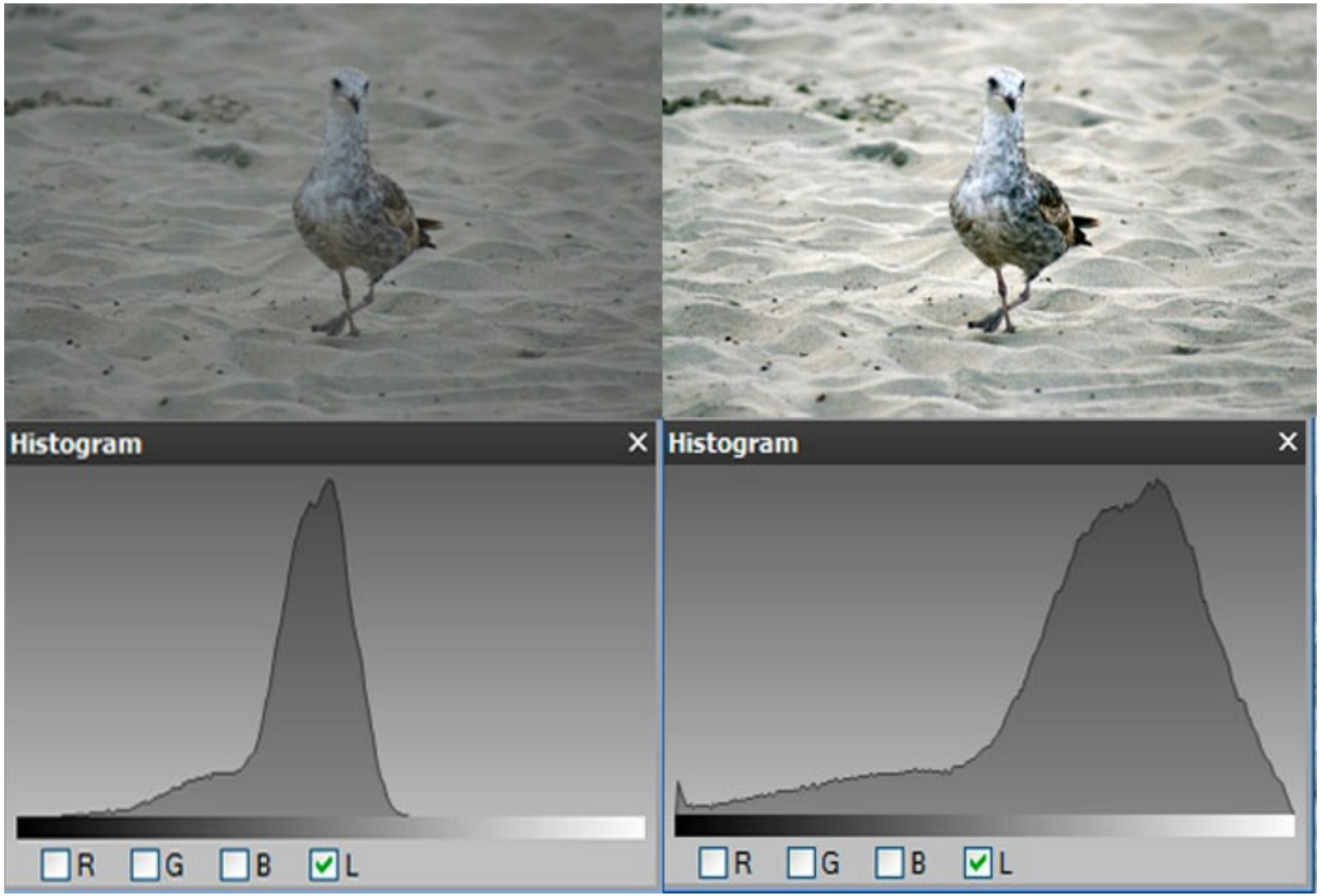
Contrast is the difference between maximum and minimum pixel intensity in a given region

Low contrast

High contrast

Low contrast

High contrast



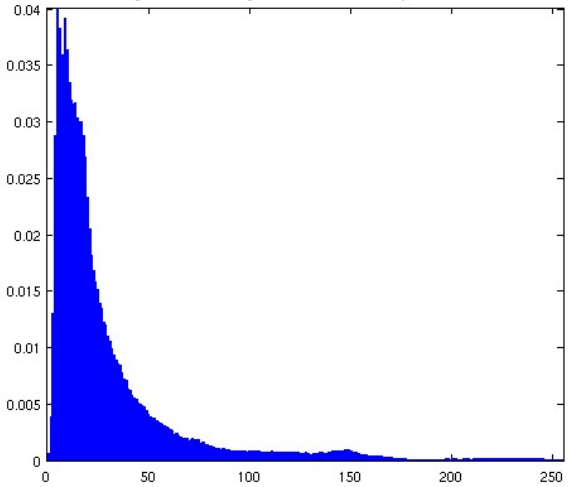
Narrow

Wide

Example: Histogram Normalization

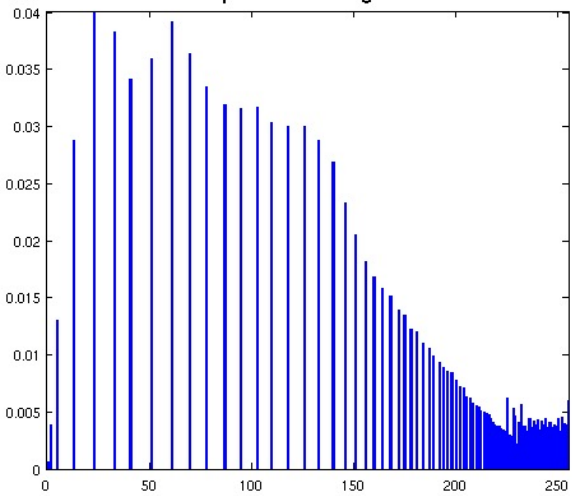
Input Image

Original Histogram before equalization



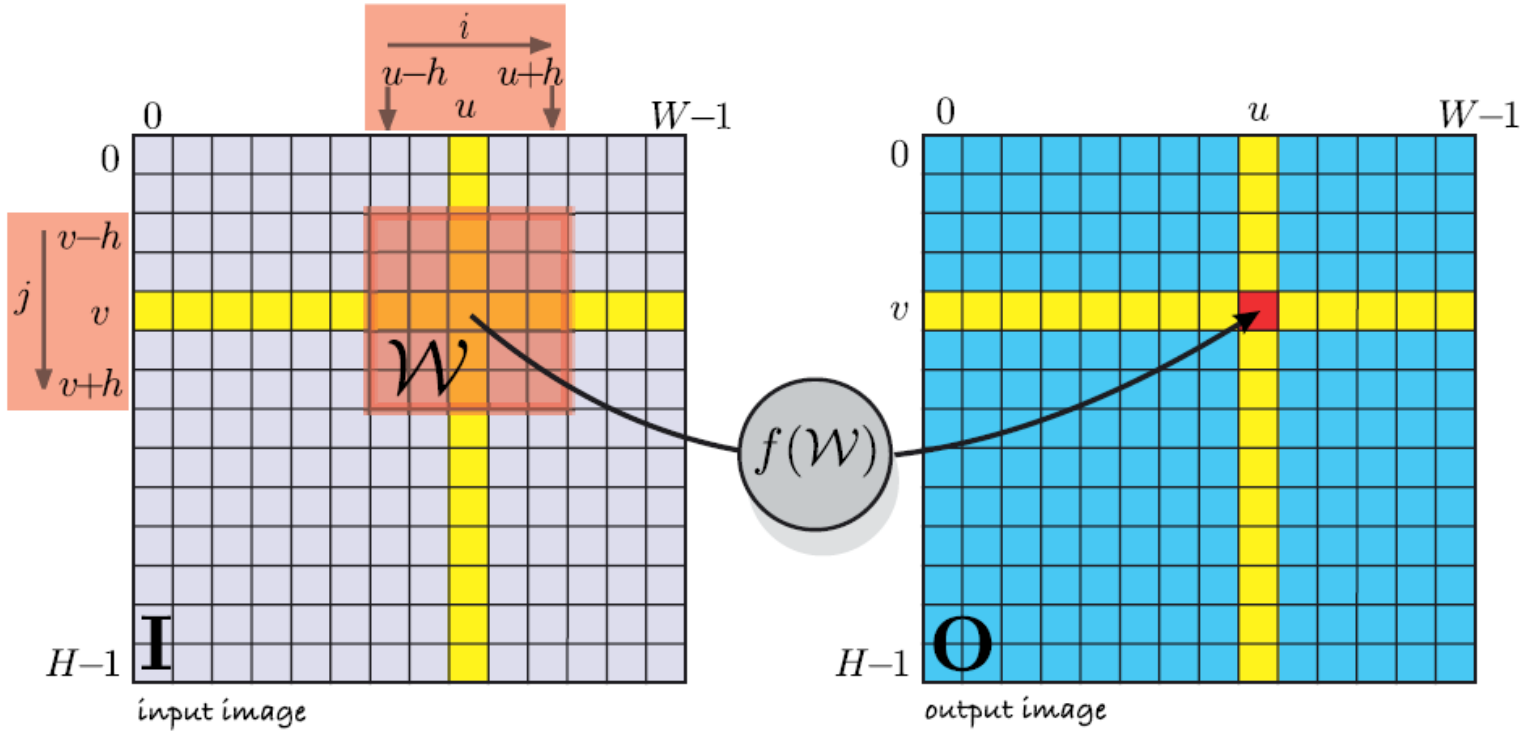
Output Image

Equalized histogram



Local operators

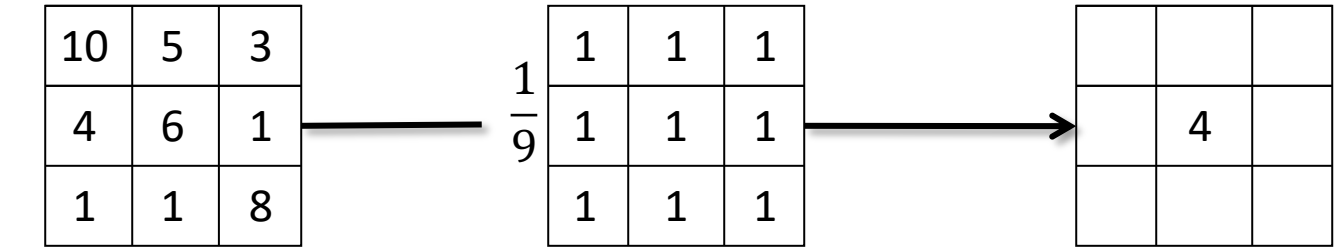
Image processing operations that use a region of pixels in the input image to determine the value of a single pixel in the output image



many-to-one mapping between pixels

Example: Linear averaging filter

- Replace each pixel by a linear combination of its neighbors
- The matrix of the linear combination is called the "kernel," "mask", or "filter"



What do you think will happen?

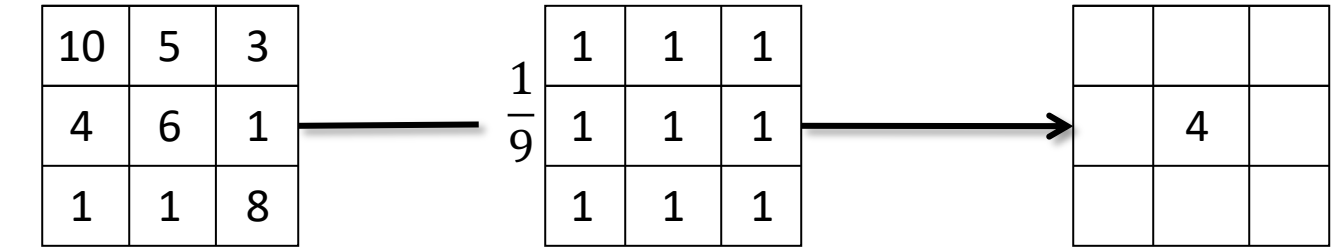
Smoothing and blur!

Source: L. Zhang

Example: Linear averaging filter

Let F be the image, H be the kernel (of size $2k+1$ by $2k+1$), and G be the output image

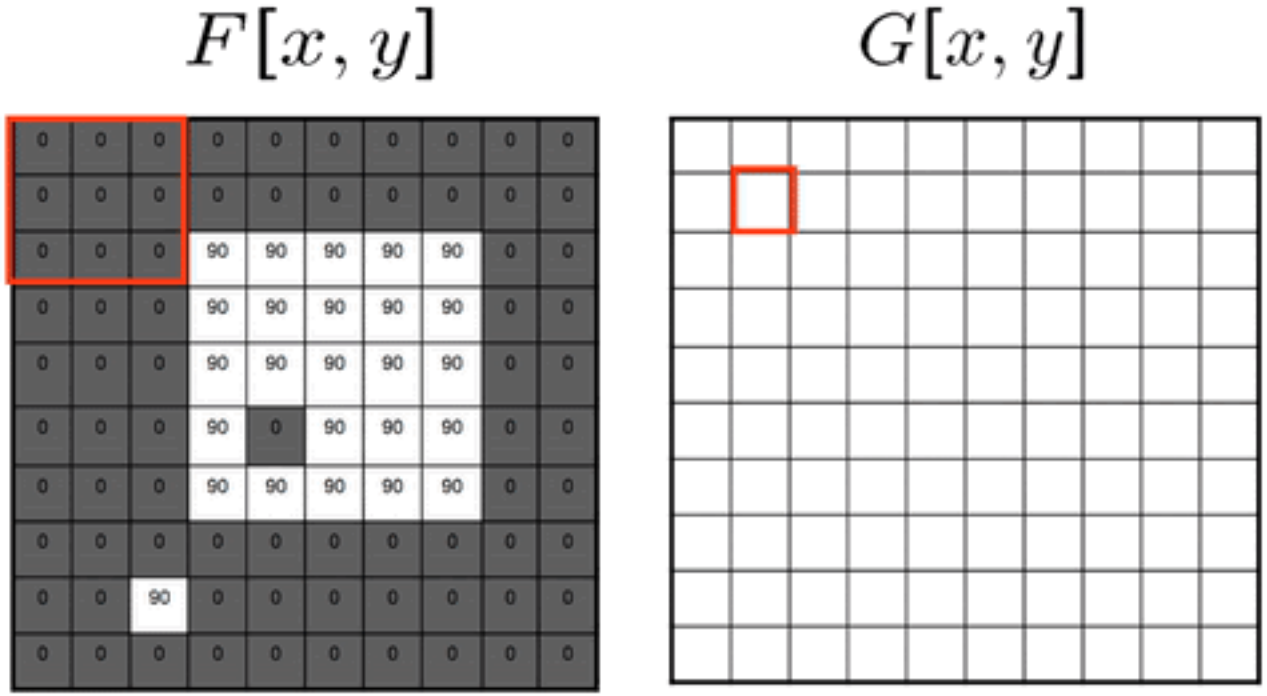
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$



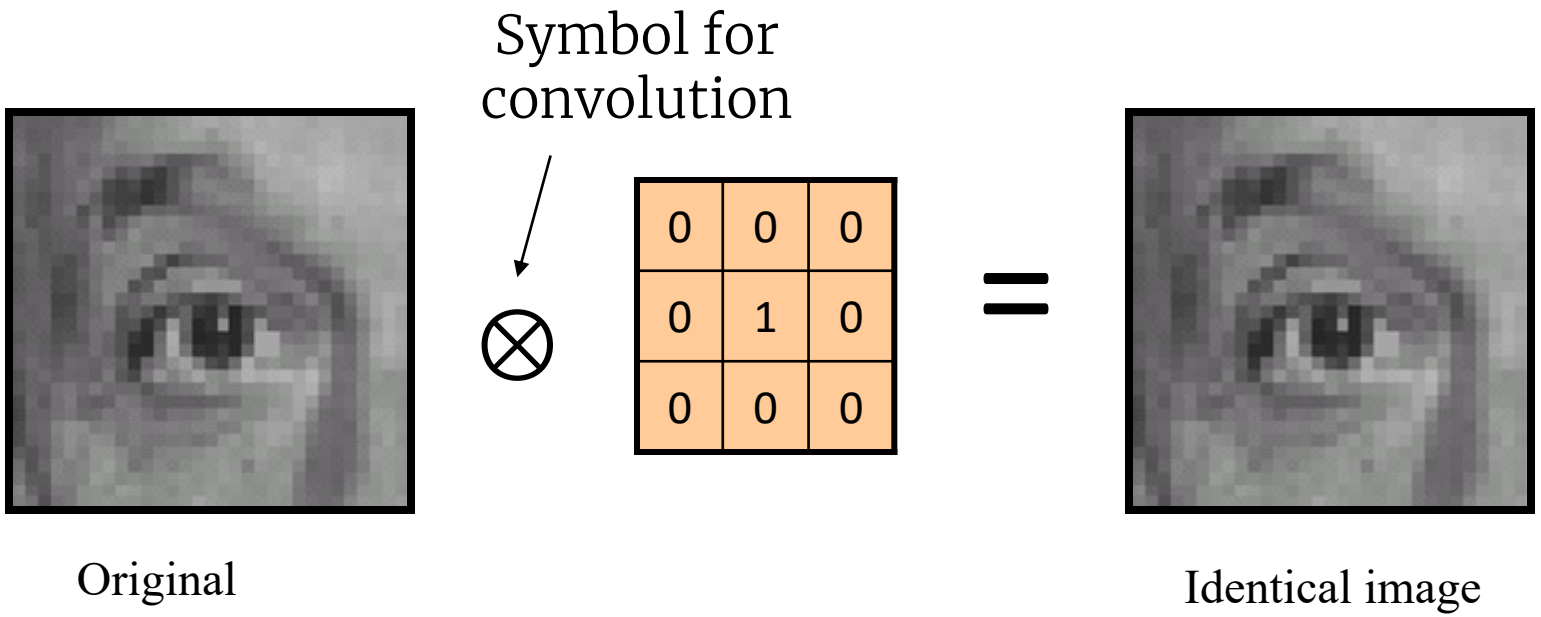
Source: L. Zhang

Convolution

- Applying a kernel to an image in this way is called **convolution**.
- NOTE: Convolution can produce unwanted artifacts along the edges of the image.
- Techniques for addressing this include zero padding, edge value replication, mirror extension, and others.

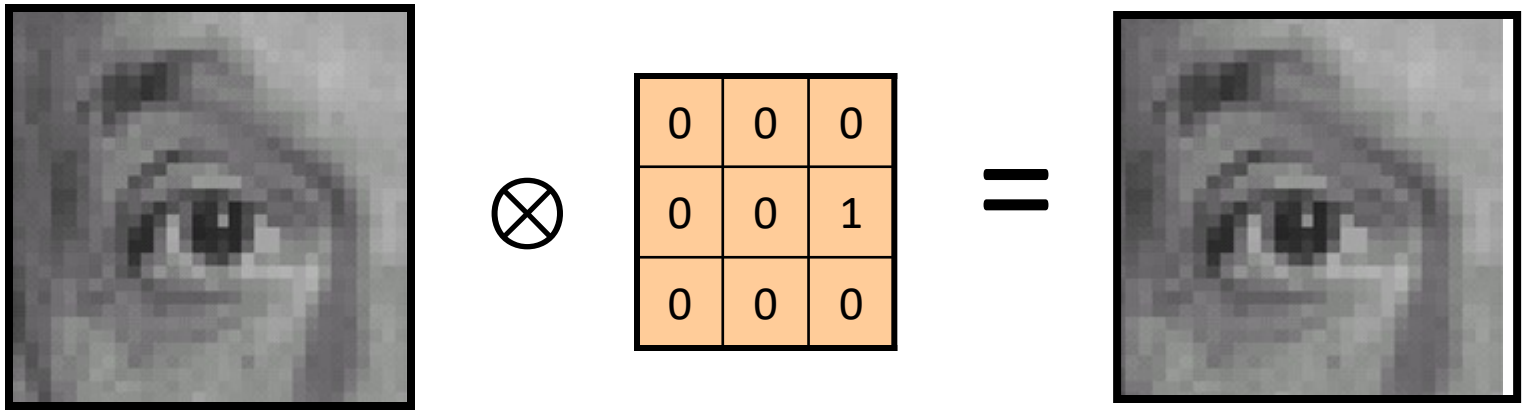


Linear filters: examples



Source: D. Lowe

Linear filters: examples

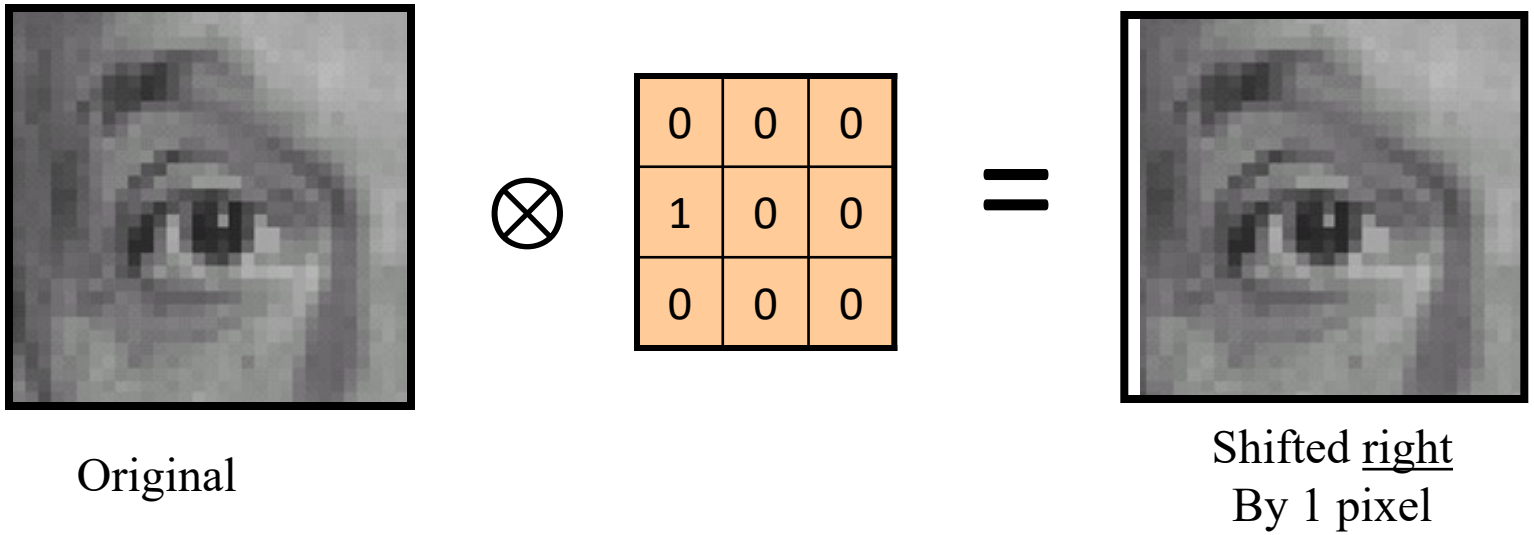


Original

Shifted left
By 1 pixel

Source: D. Lowe

Linear filters: examples

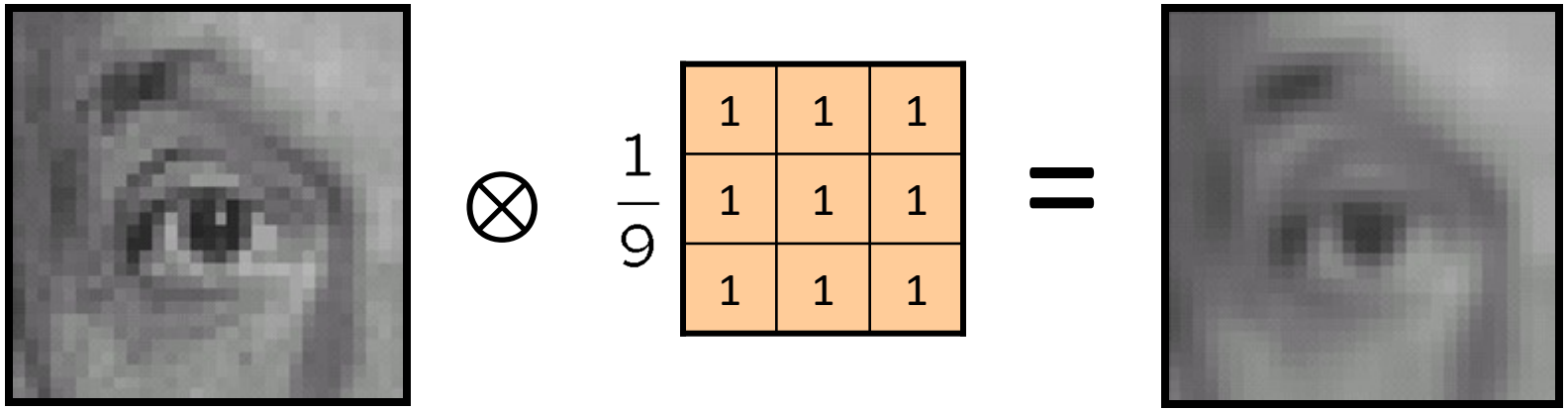


Original

Shifted right
By 1 pixel

Source: D. Lowe

Linear filters: examples

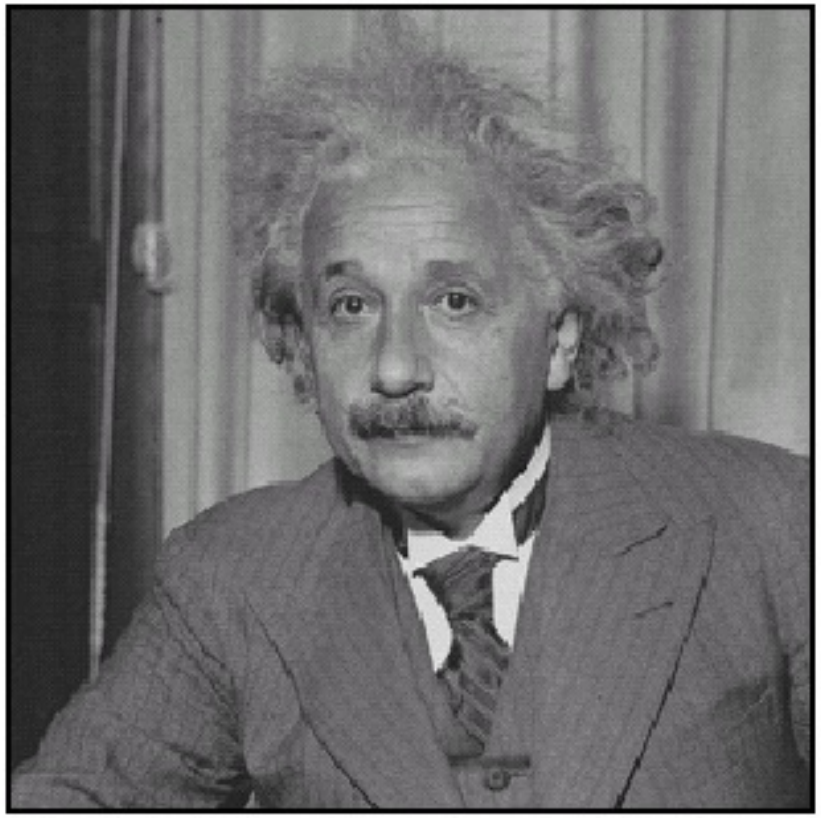


Original

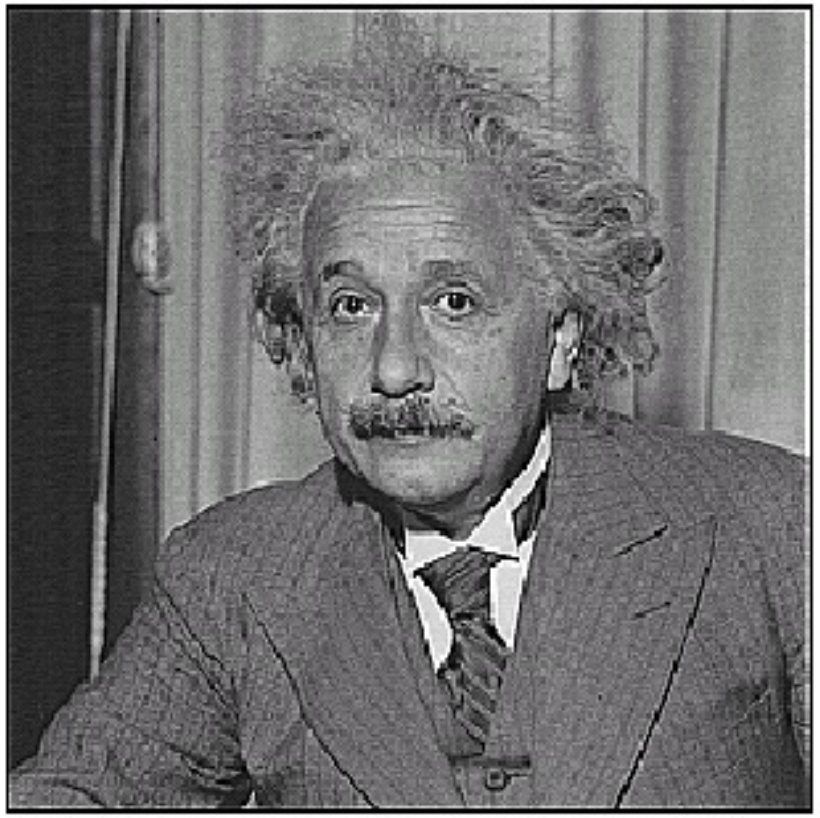
Mean filter (blurring)

Source: D. Lowe

Sharpening



before



after

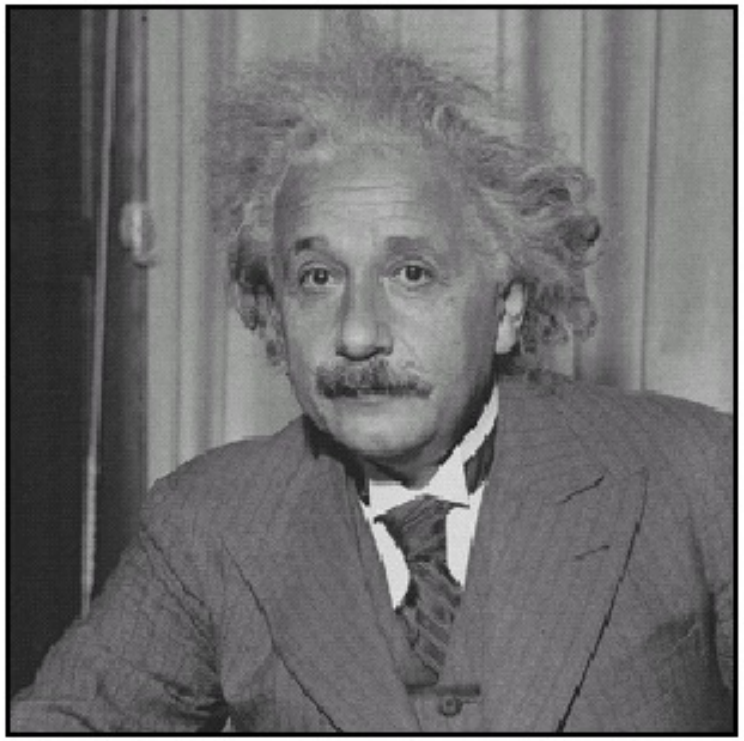
Source: D. Lowe

Sharpening

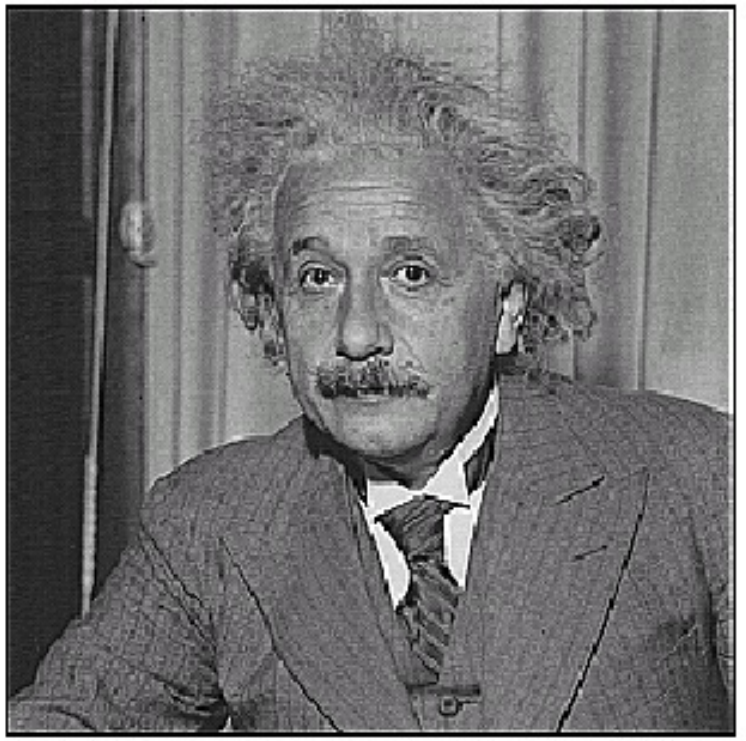
$$\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix} + \begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} = \begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

Original image

(Original image) – (average of neighbors) = highlights



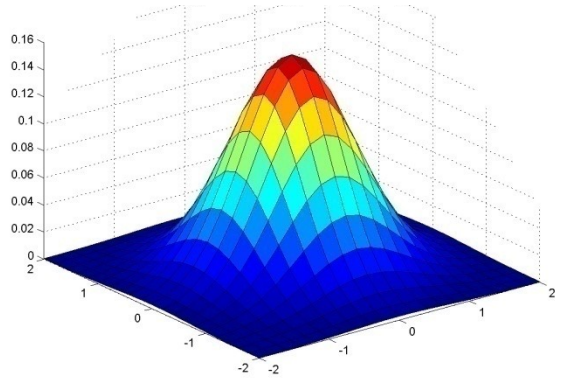
before

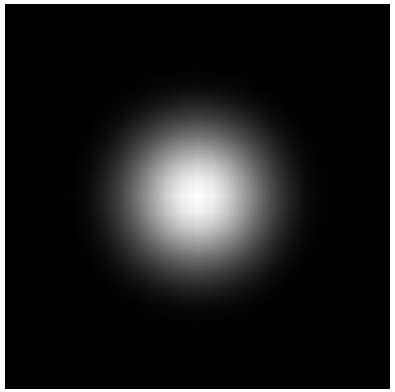


after

Source: D. Lowe

Gaussian Kernel





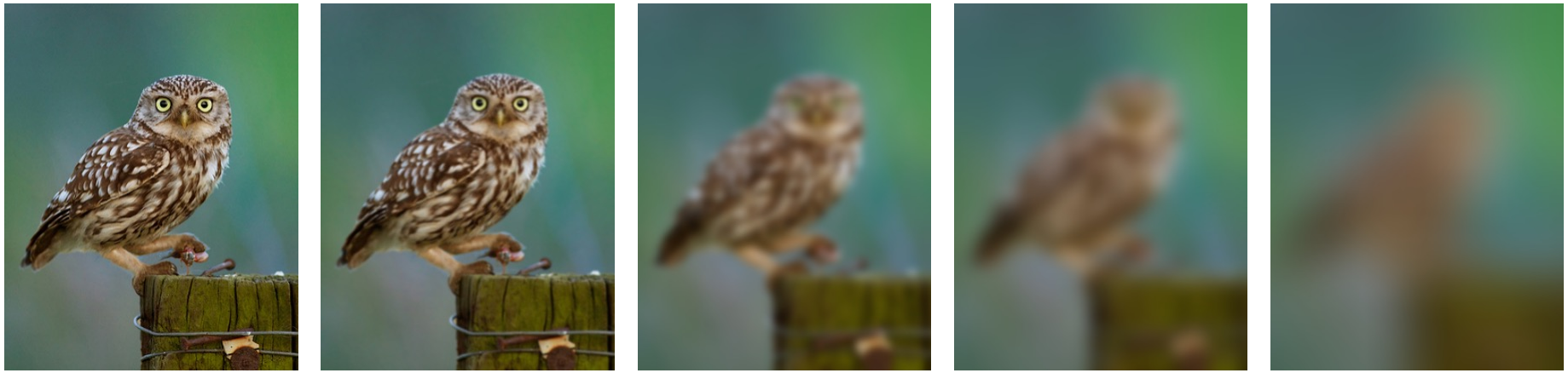
Approximated by:

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

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

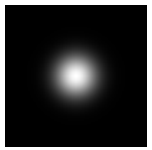
Source: C. Rasmussen

Gaussian filter



σ = 1 pixel

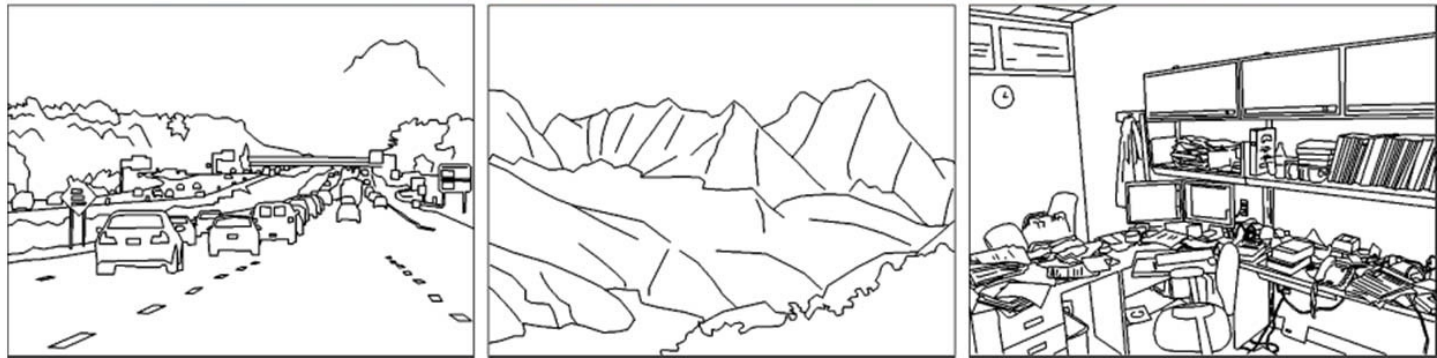
σ = 5 pixels



σ = 10 pixels

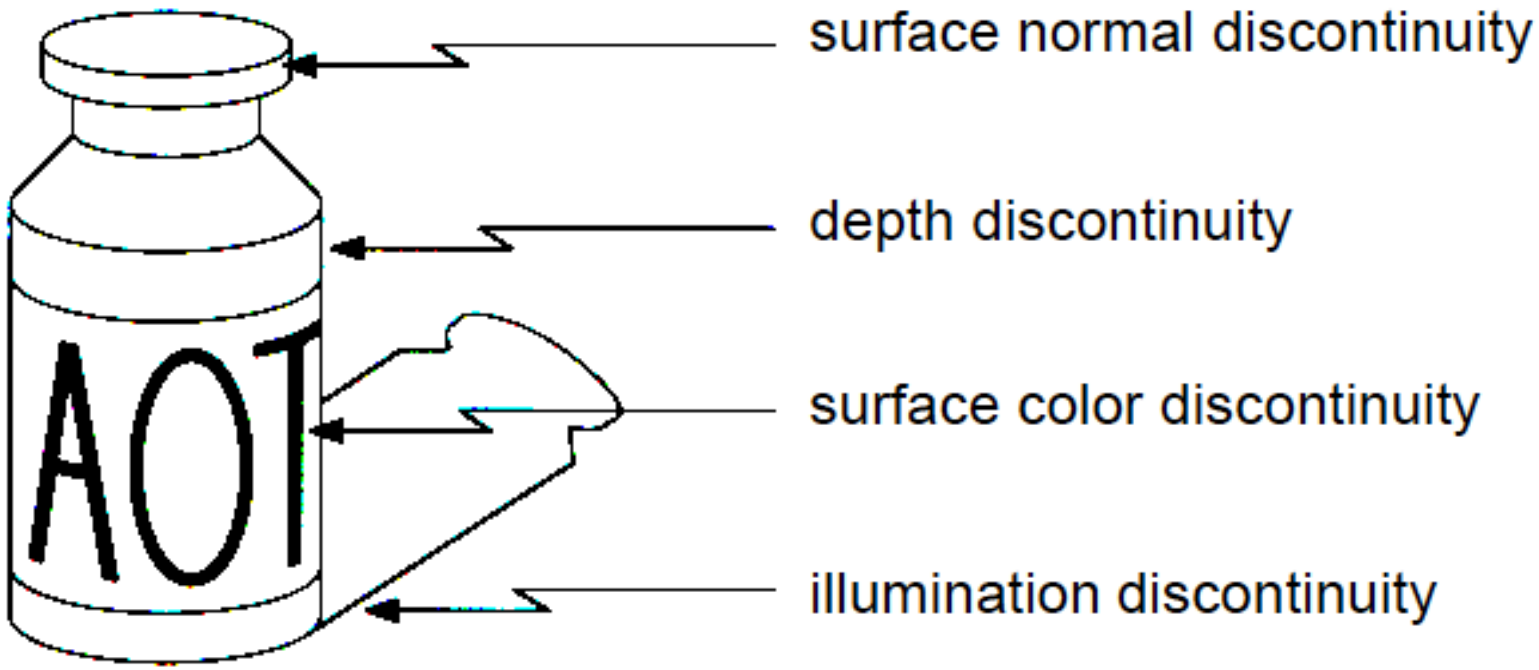
σ = 30 pixels

Edge Detection



Edge Detection

Origin of Edges



Edges

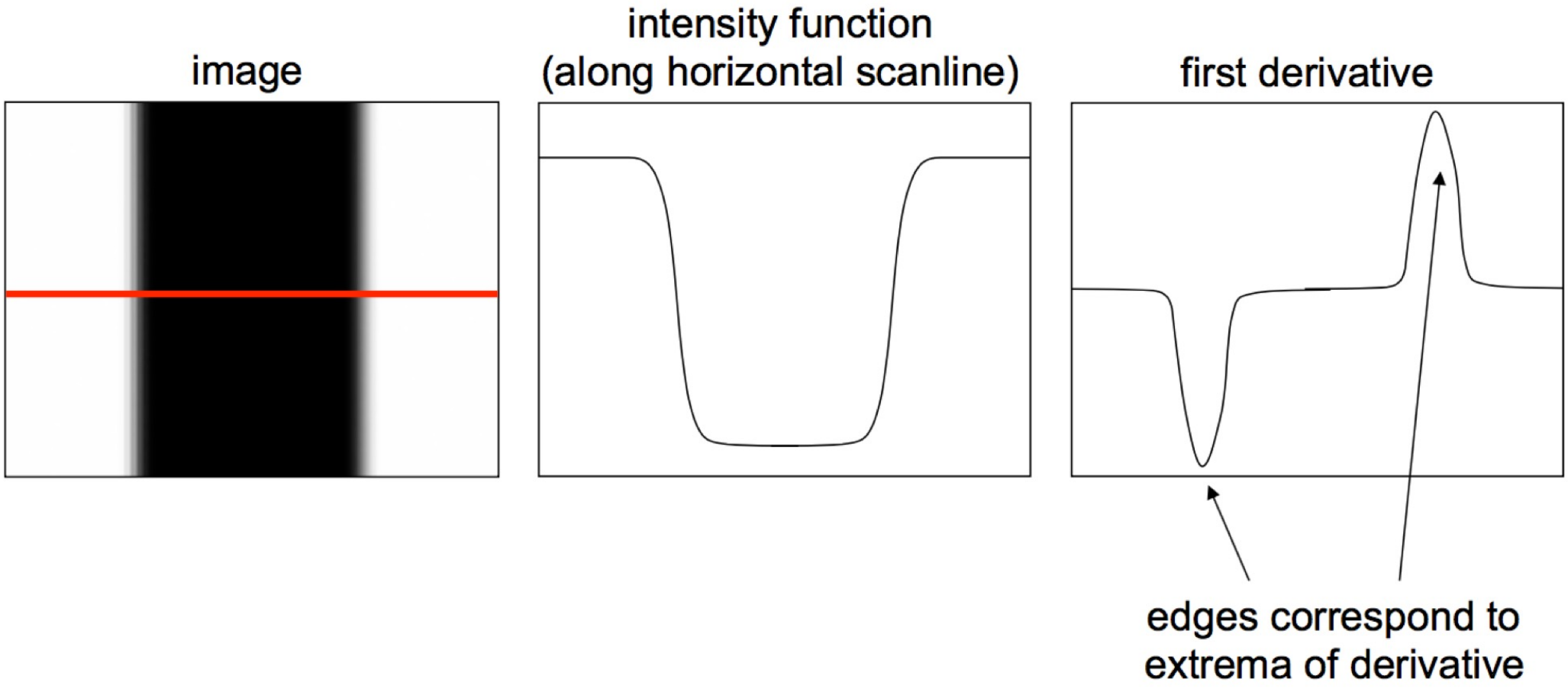


image
intensity function
(along horizontal scanline)
first derivative
edges correspond to
extrema of derivative

Edge Detection

Through Convolution

Sobel:

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

Prewitt:

-1	0	1
-1	0	1
-1	0	1

Canny:

more complex multi-stage algorithm that uses a Gaussian filter and the intensity gradient in an image. One of the most widely-used techniques.

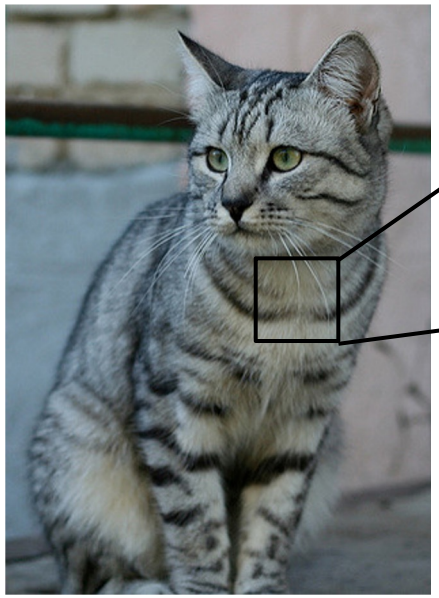
1. Supervised Learning

- Example: classification

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

cat

The Problem: Semantic Gap




```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

An image classifier

```
def classify_image(image):
  # Some magic here?
  return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

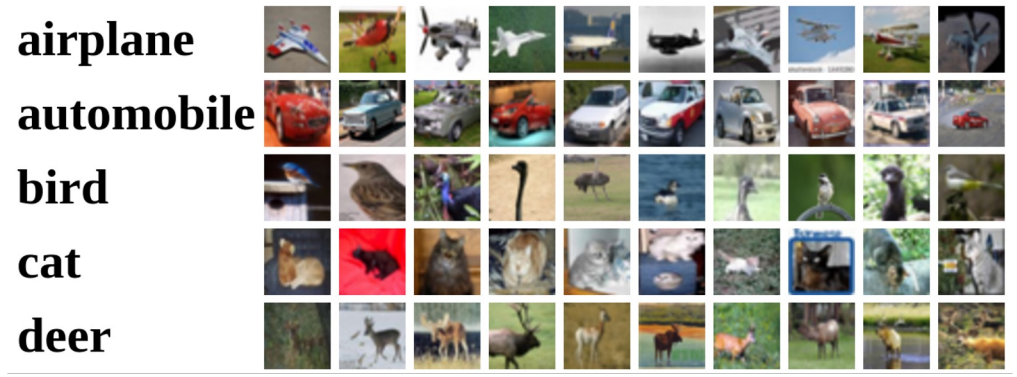
ML: A Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

```
def train(images, labels):
  # Machine learning!
  return model
```

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

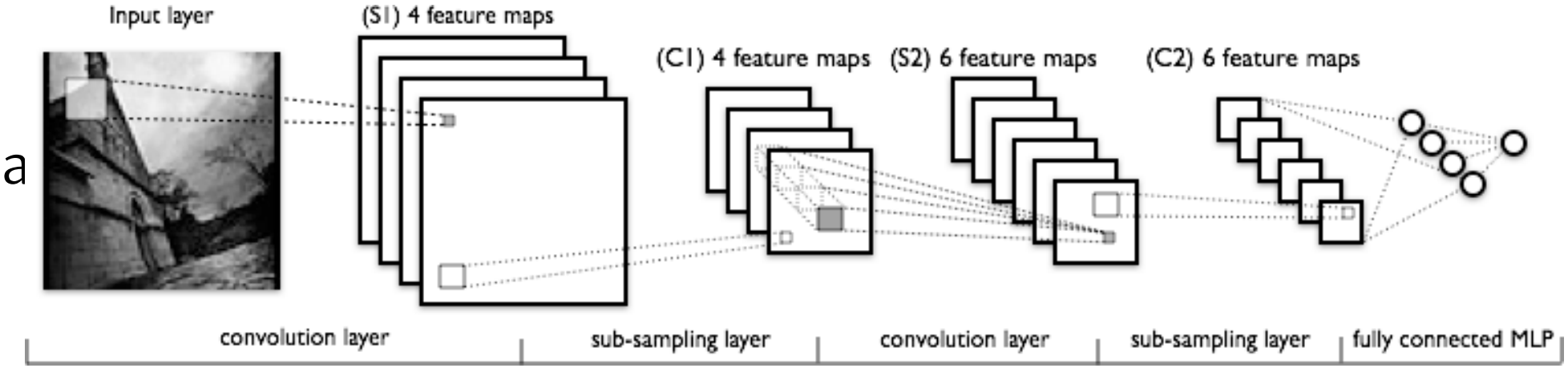
Example training set



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

2. Convolutional Neural Networks

a



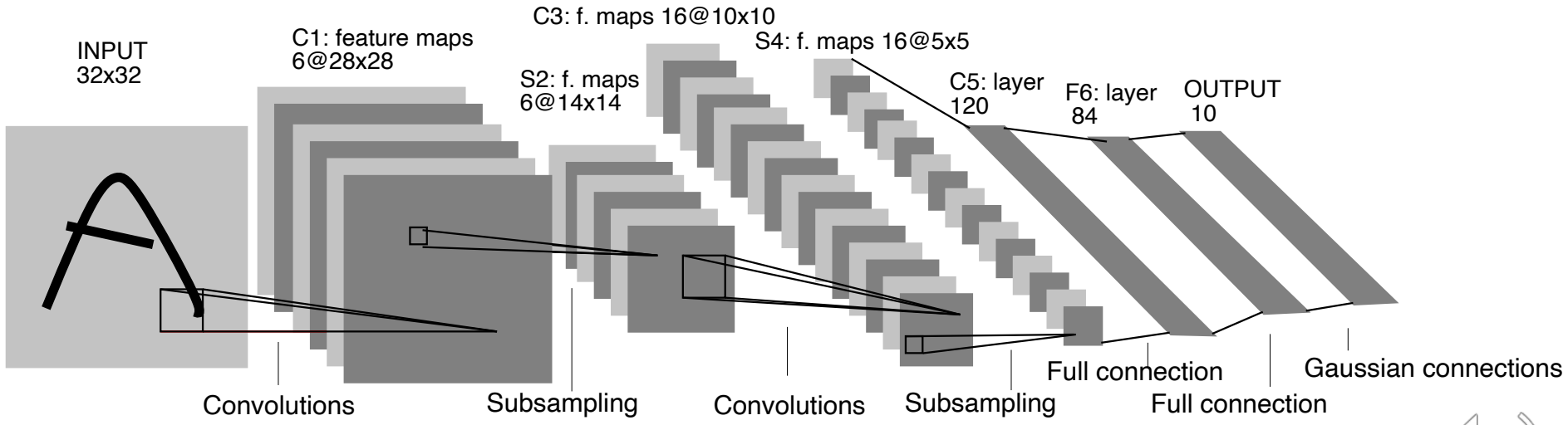
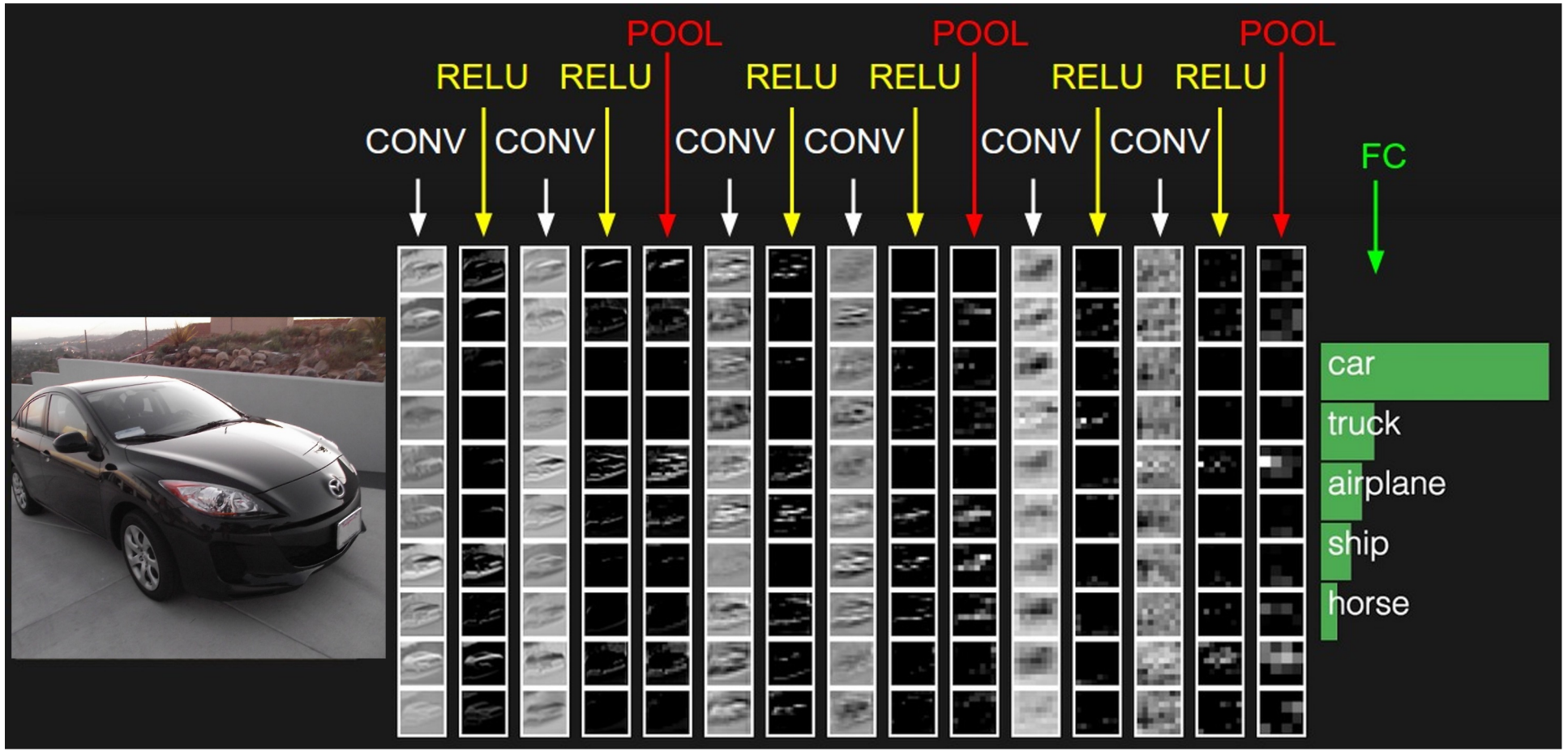


Image Credit: Yann LeCun, Kevin Murphy

preview:



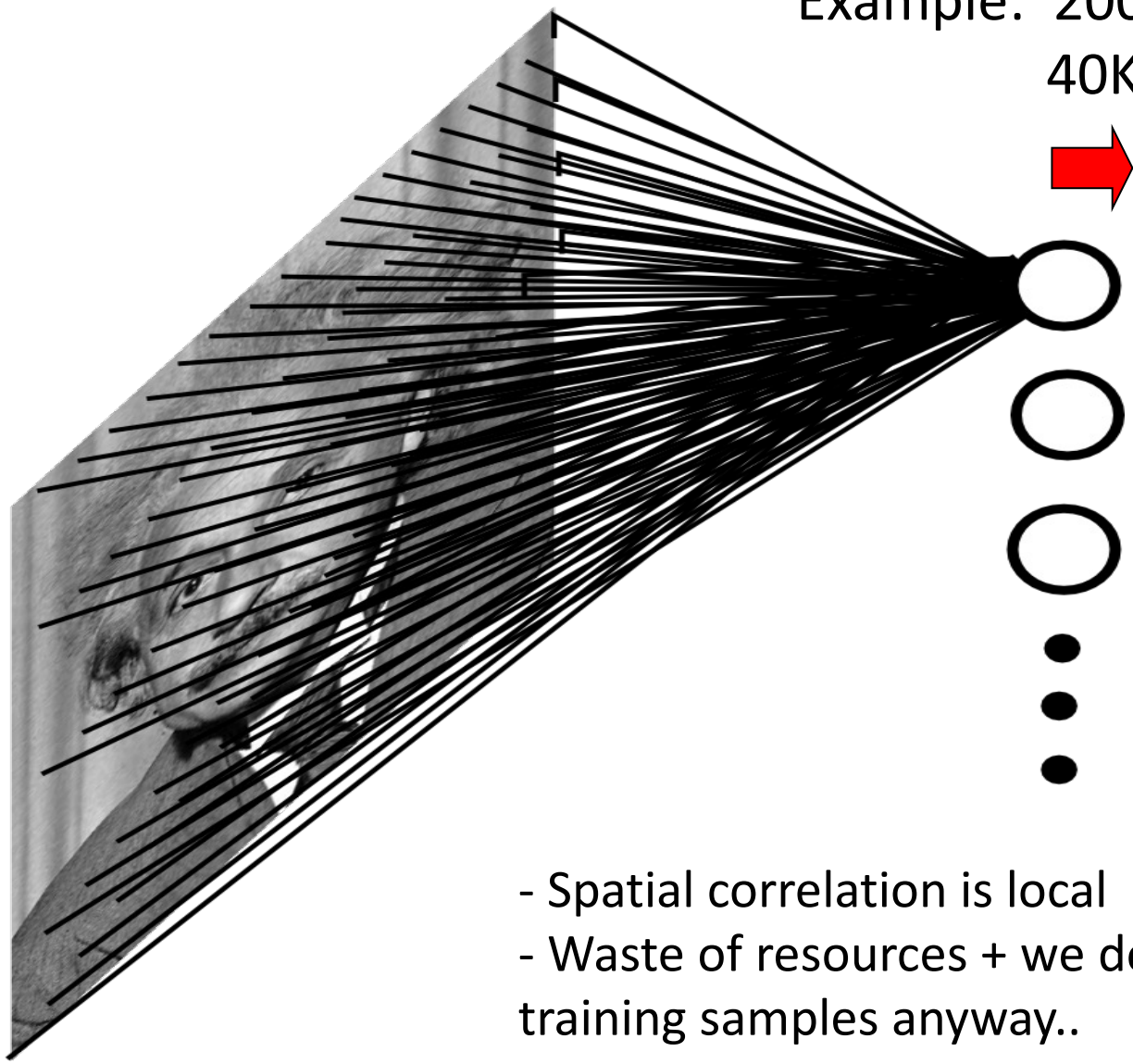
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Fully Connected Layer

Example: 200x200 image

40K hidden units

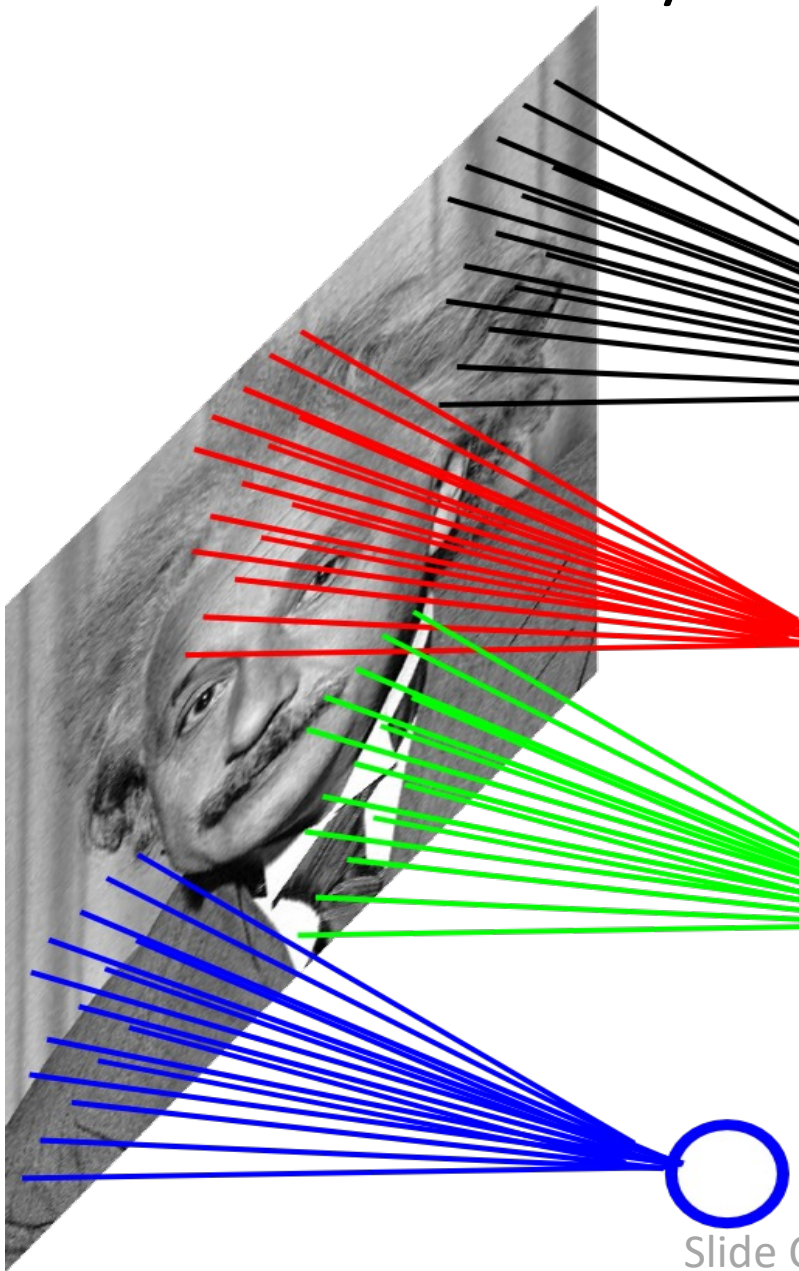
~2.4B parameters!!!



- Spatial correlation is local
- Waste of resources + we do not have enough training samples anyway..

Slide Credit: Marc'Aurelio Ranzato

Locally Connected Layer

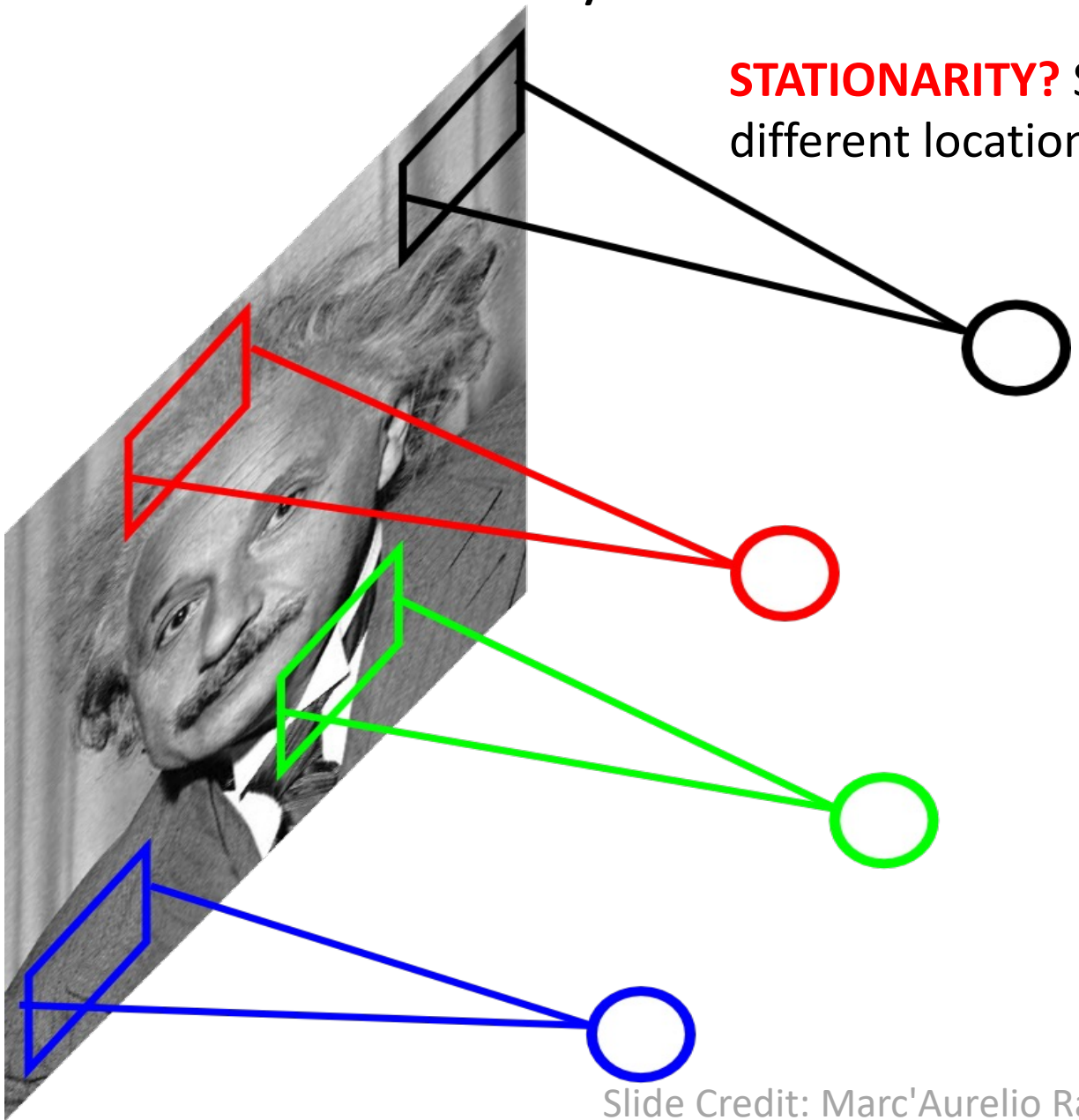


Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Slide Credit: Marc'Aurelio Ranzato

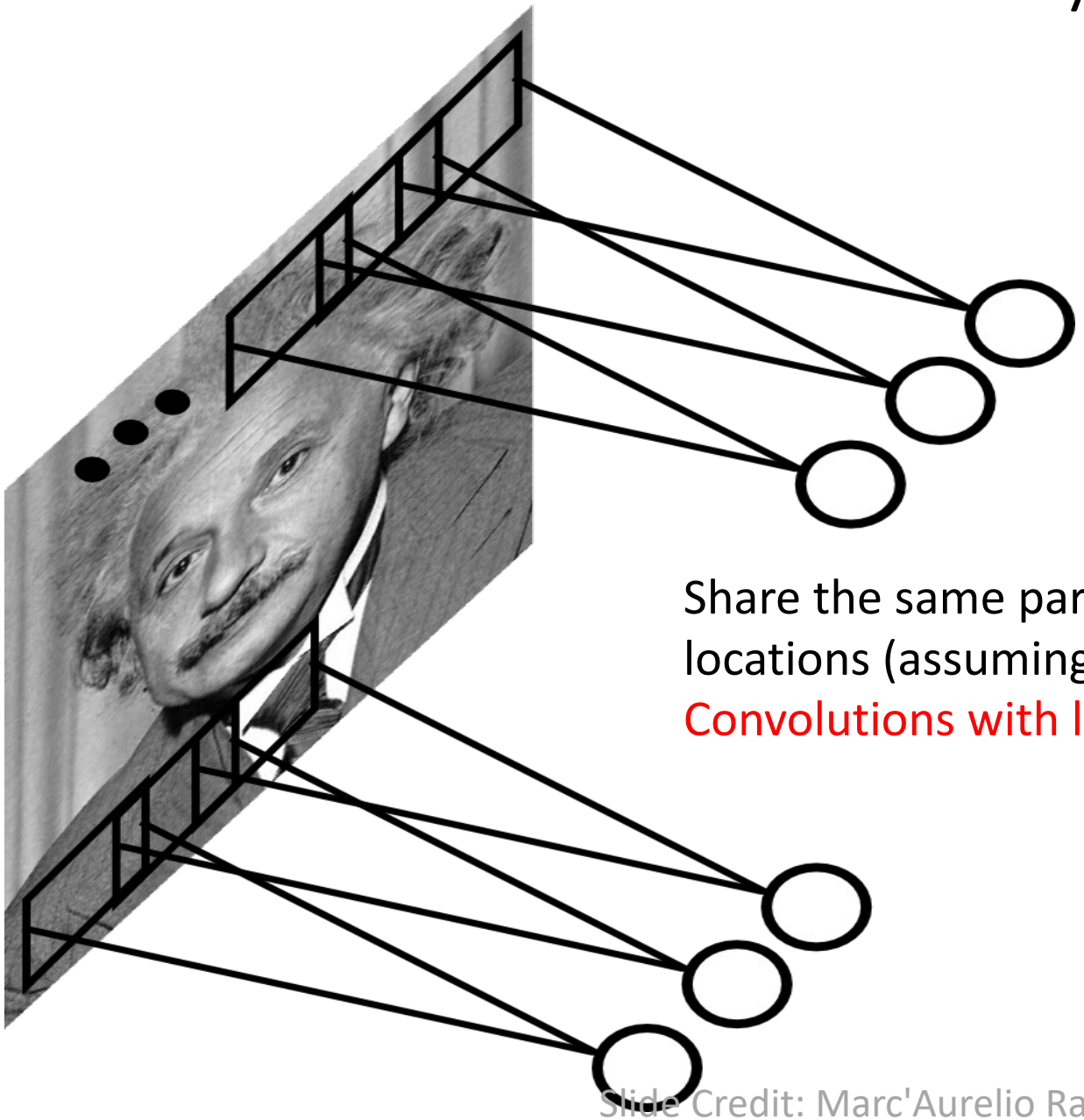
Locally Connected Layer



STATIONARITY? Statistics is similar at different locations

Slide Credit: Marc'Aurelio Ranzato

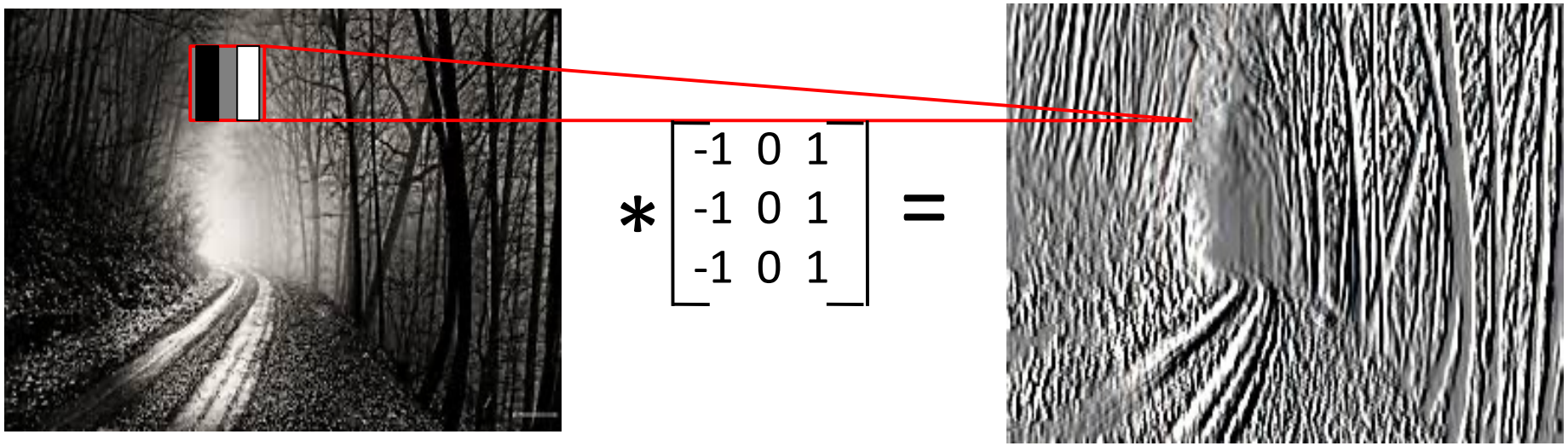
Convolutional Layer



Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

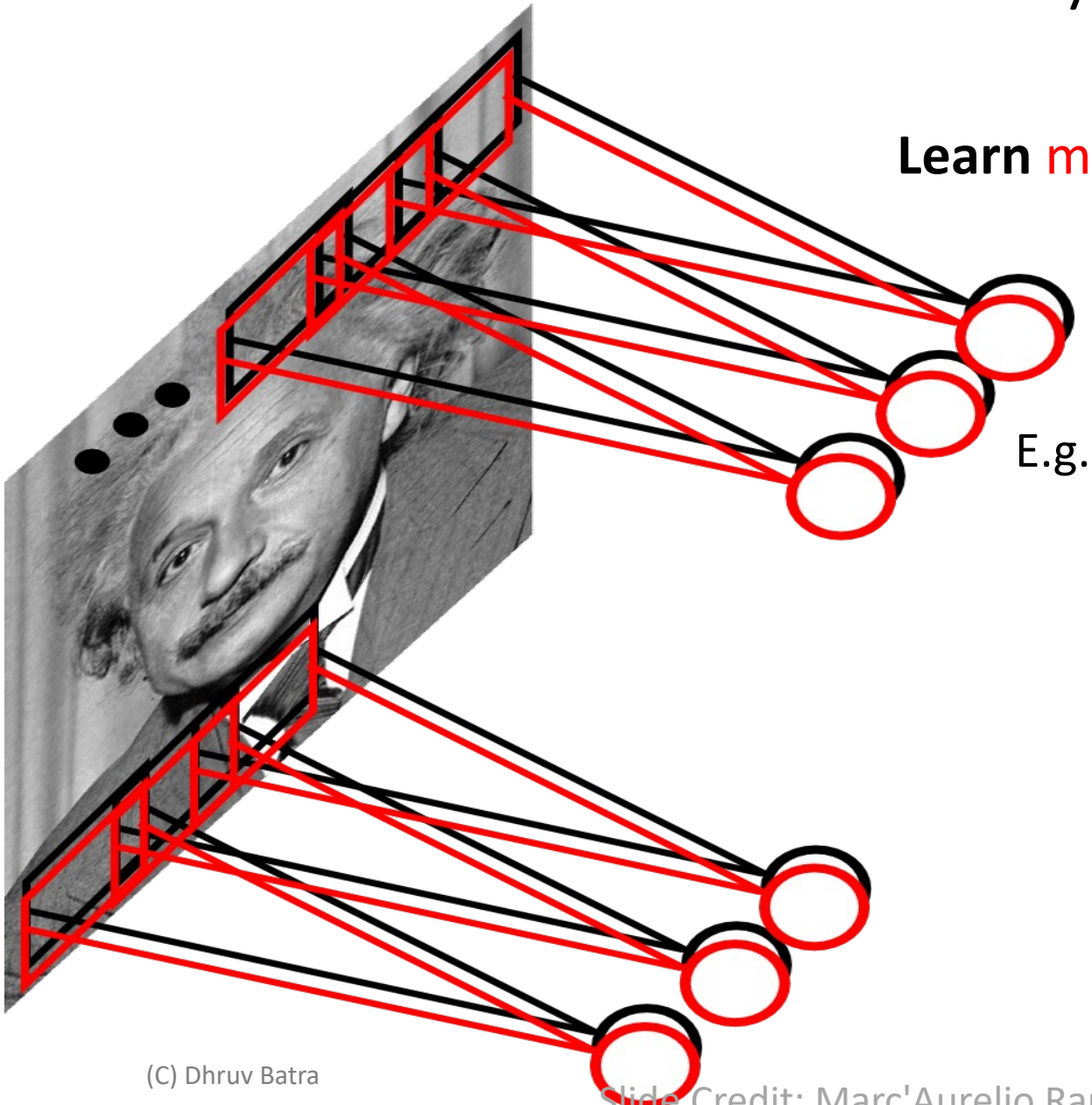
Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



Slide Credit: Marc'Aurelio Ranzato

Convolutional Layer



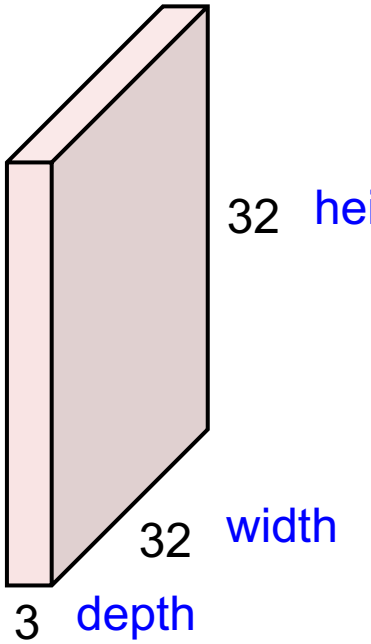
Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

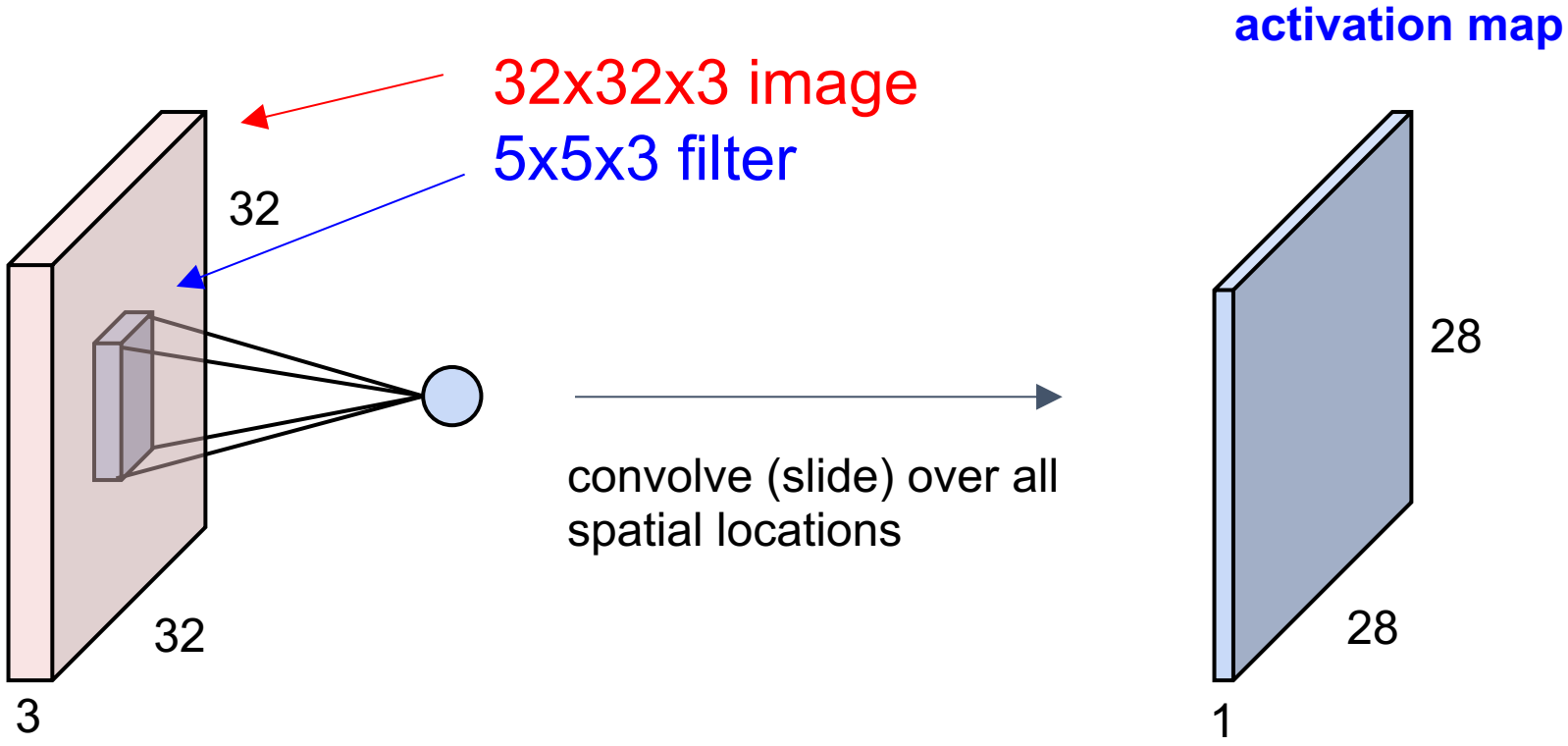
Slide Credit: Marc'Aurelio Ranzato

Convolution Layer

32x32x3 image -> preserve spatial structure

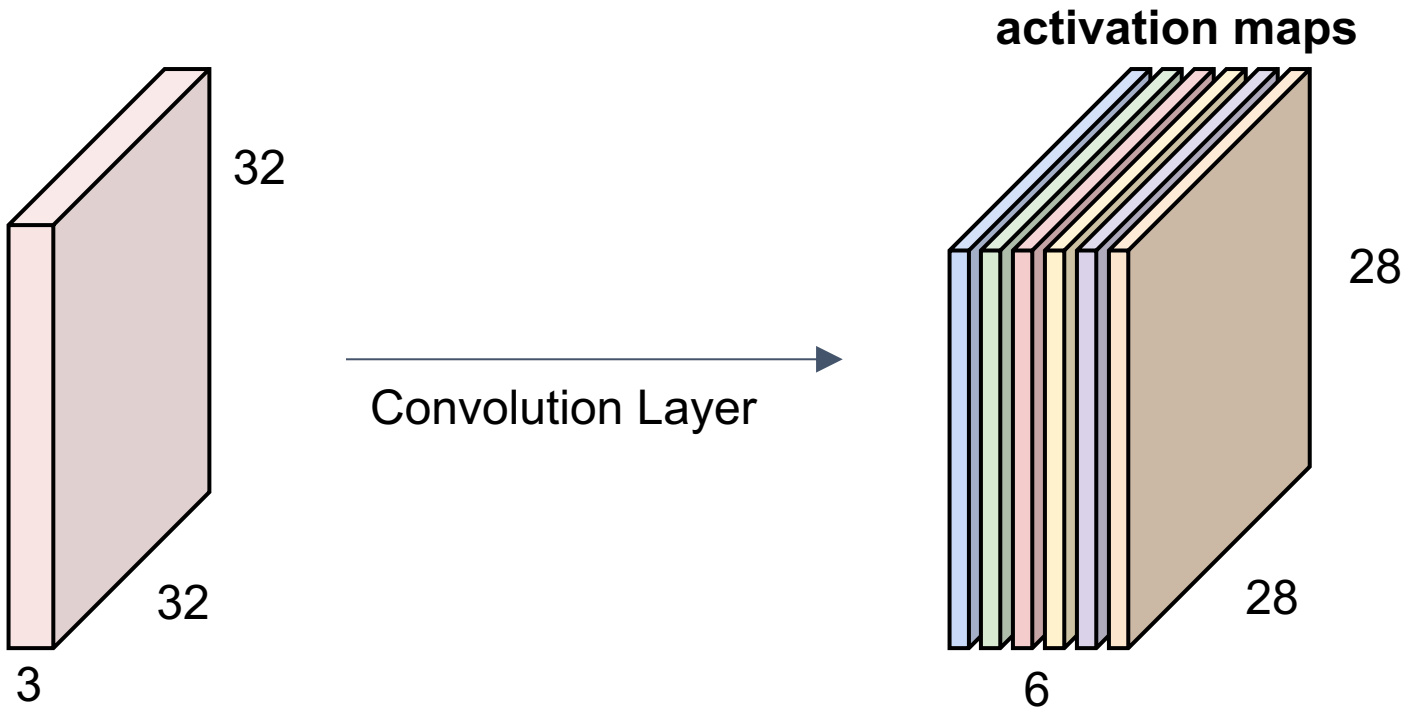


Convolution Layer



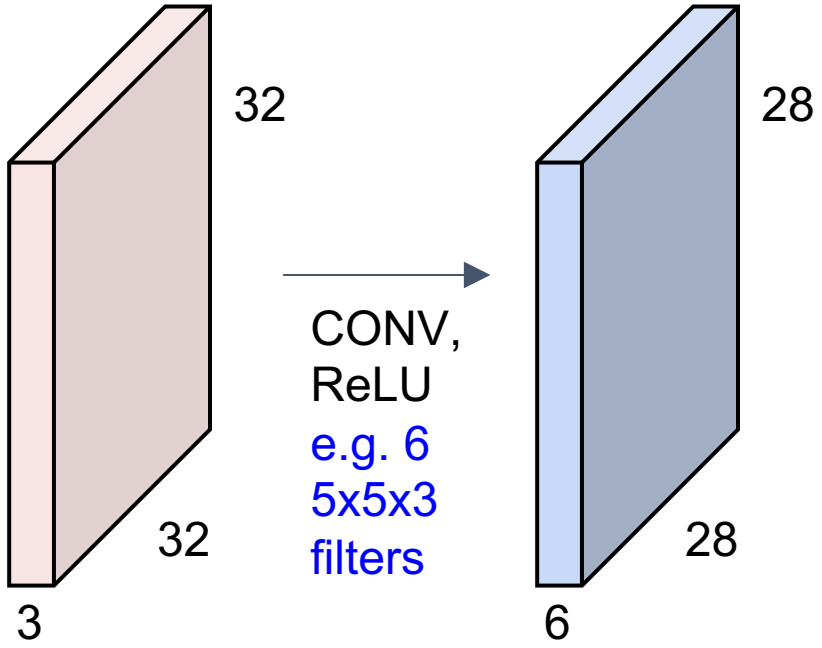
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Multiple filters: if we have 6 5x5 filters, we'll get 6 separate activation maps:

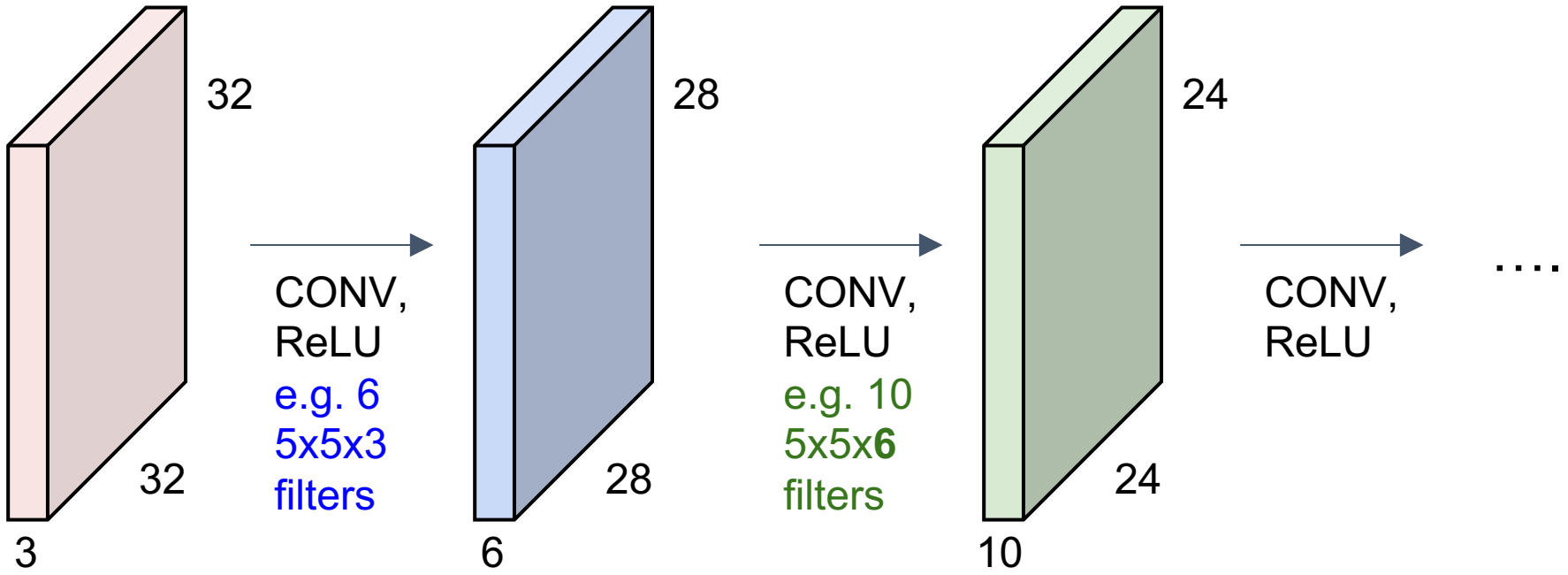


We stack these up to get a "new image" of size 28x28x6!

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



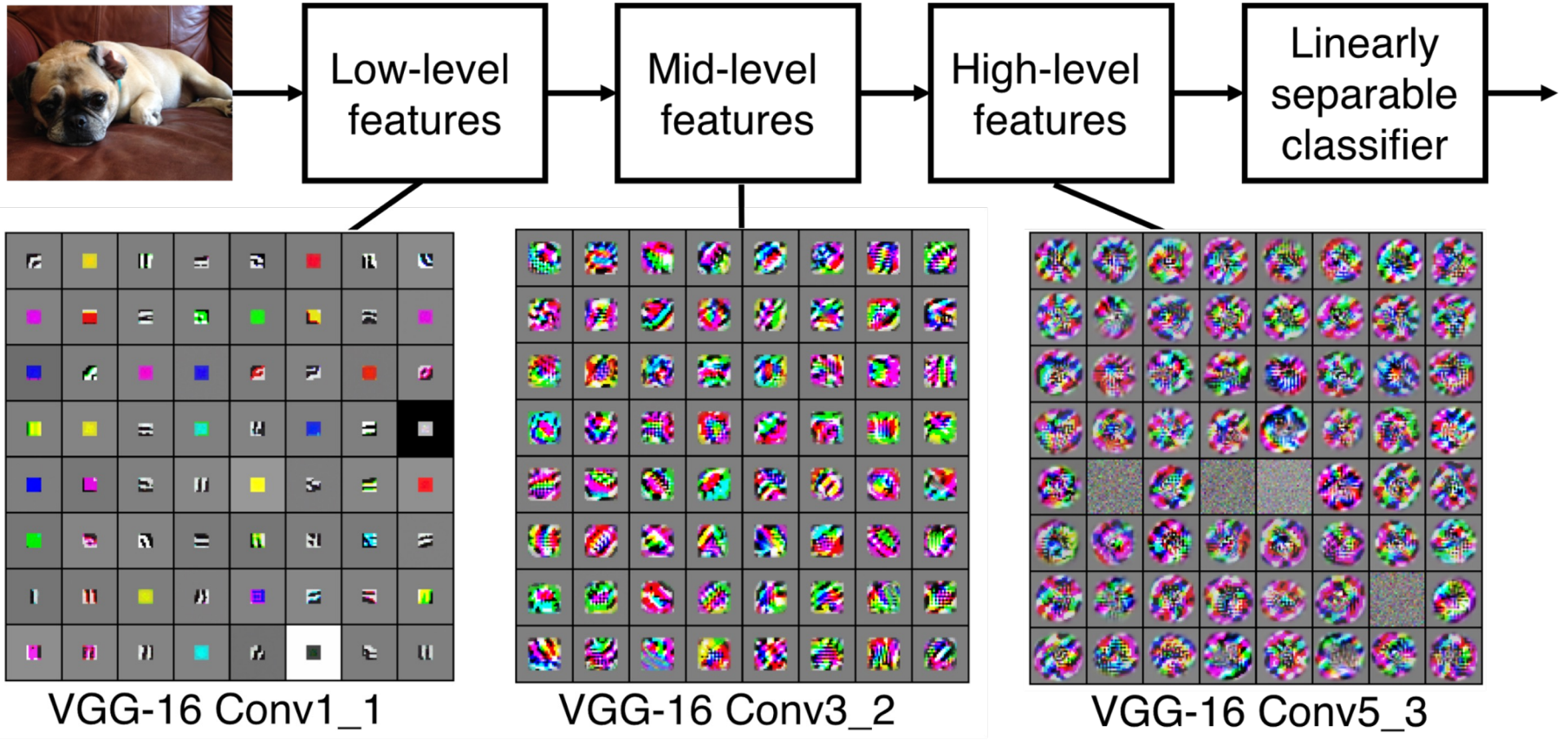
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Preview

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



Visualizing Learned Filters

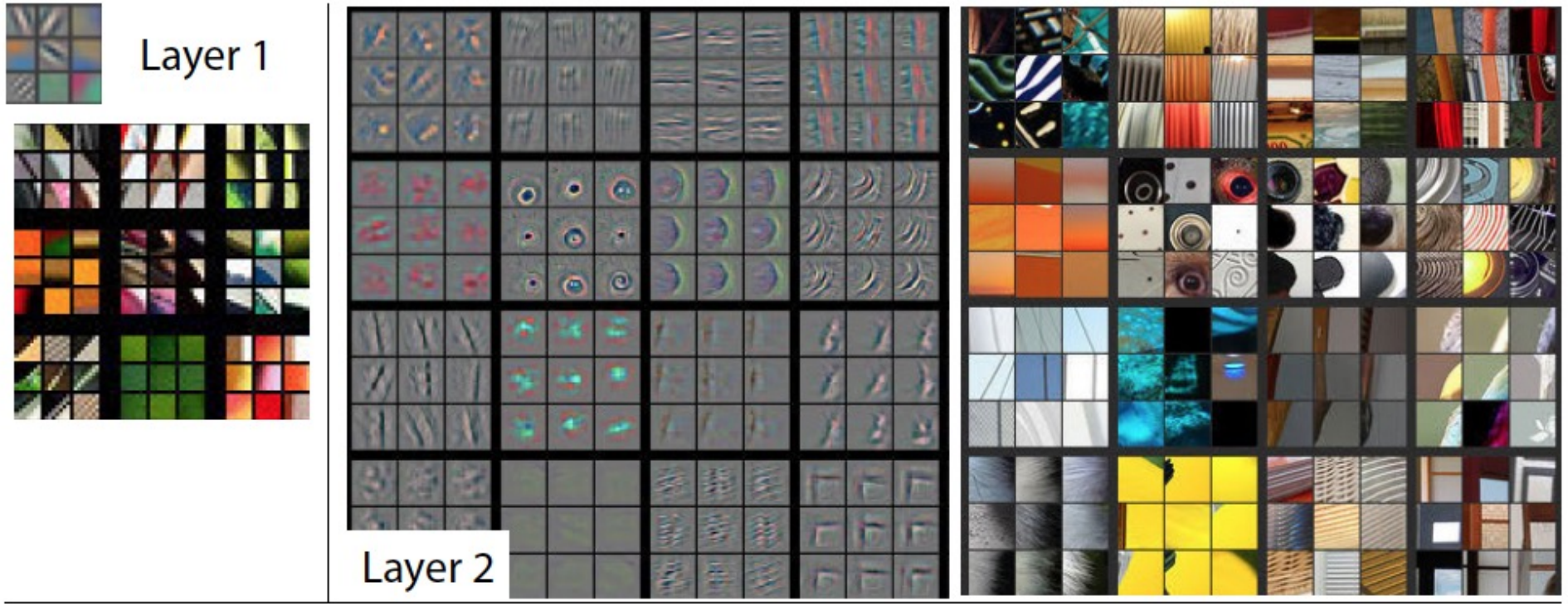


Figure Credit: [Zeiler & Fergus ECCV14]

Visualizing Learned Filters

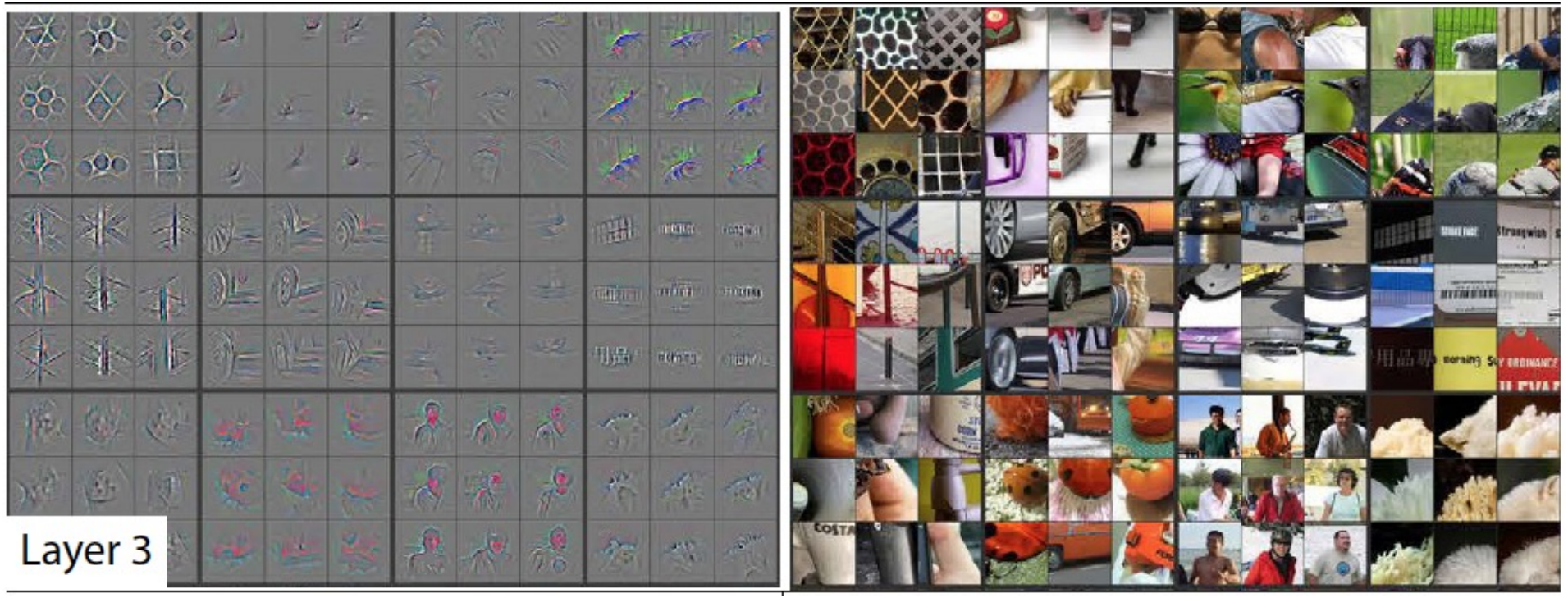


Figure Credit: [Zeiler & Fergus ECCV14]

Vi

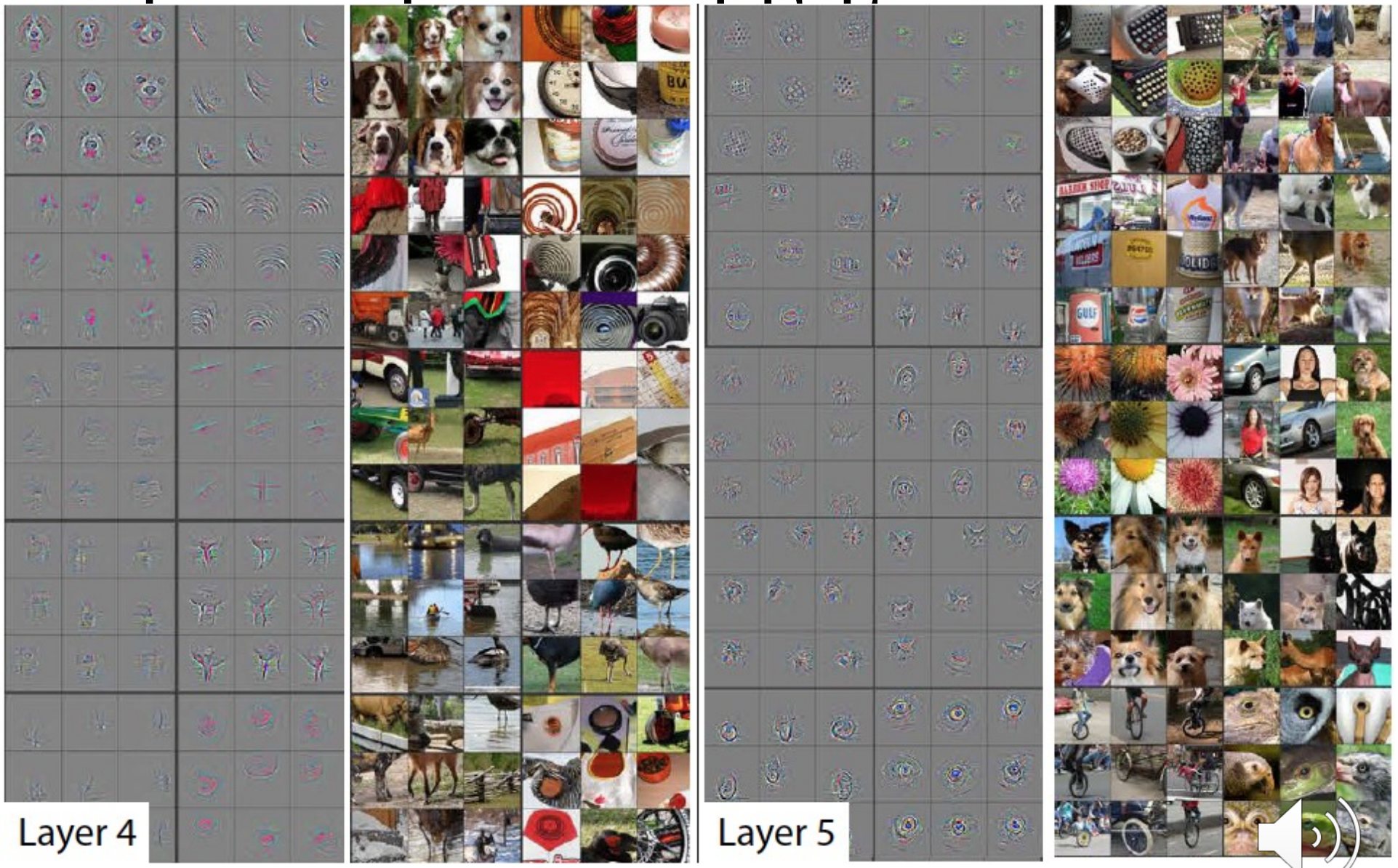
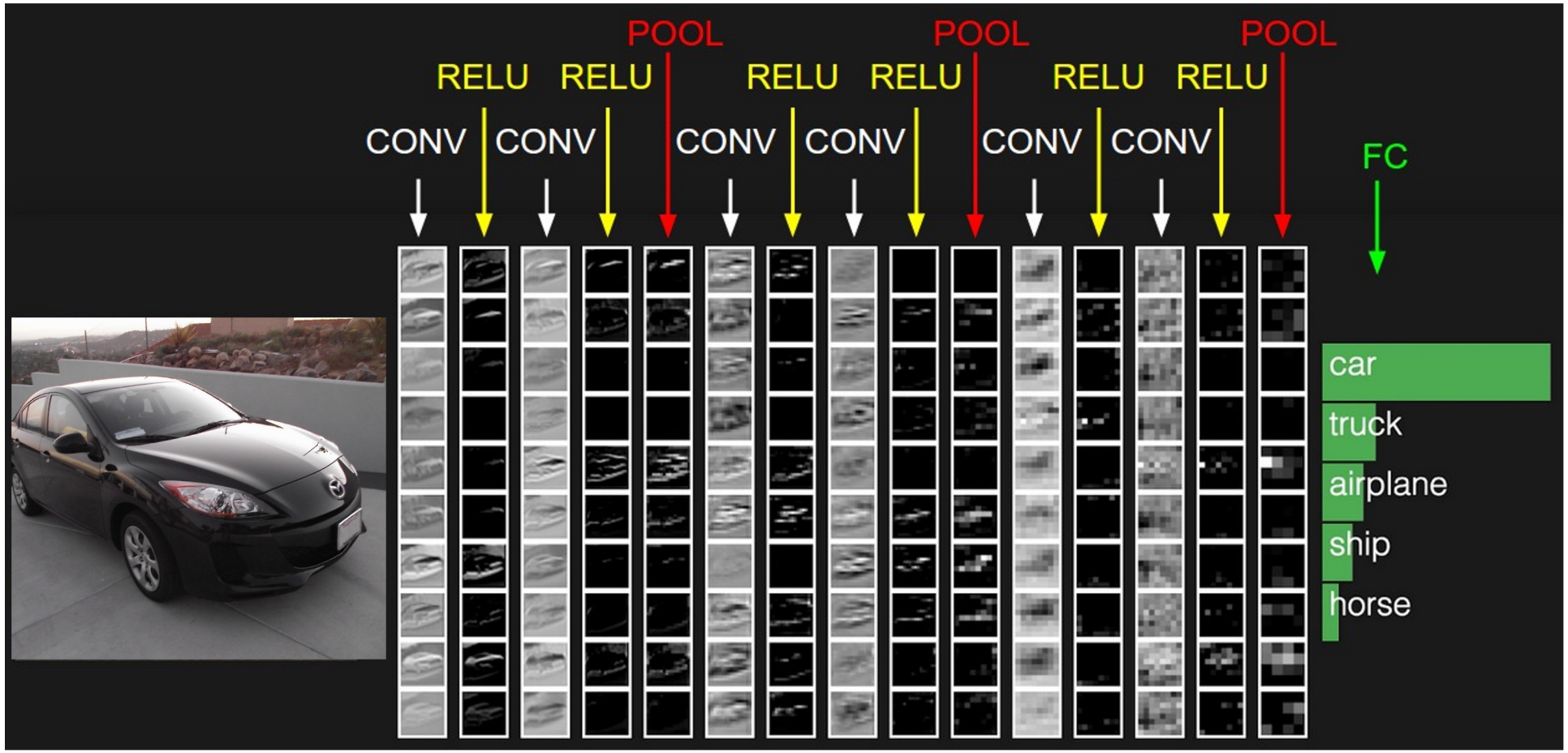


Figure Credit: [Zeiler & Fergus ECCV14]

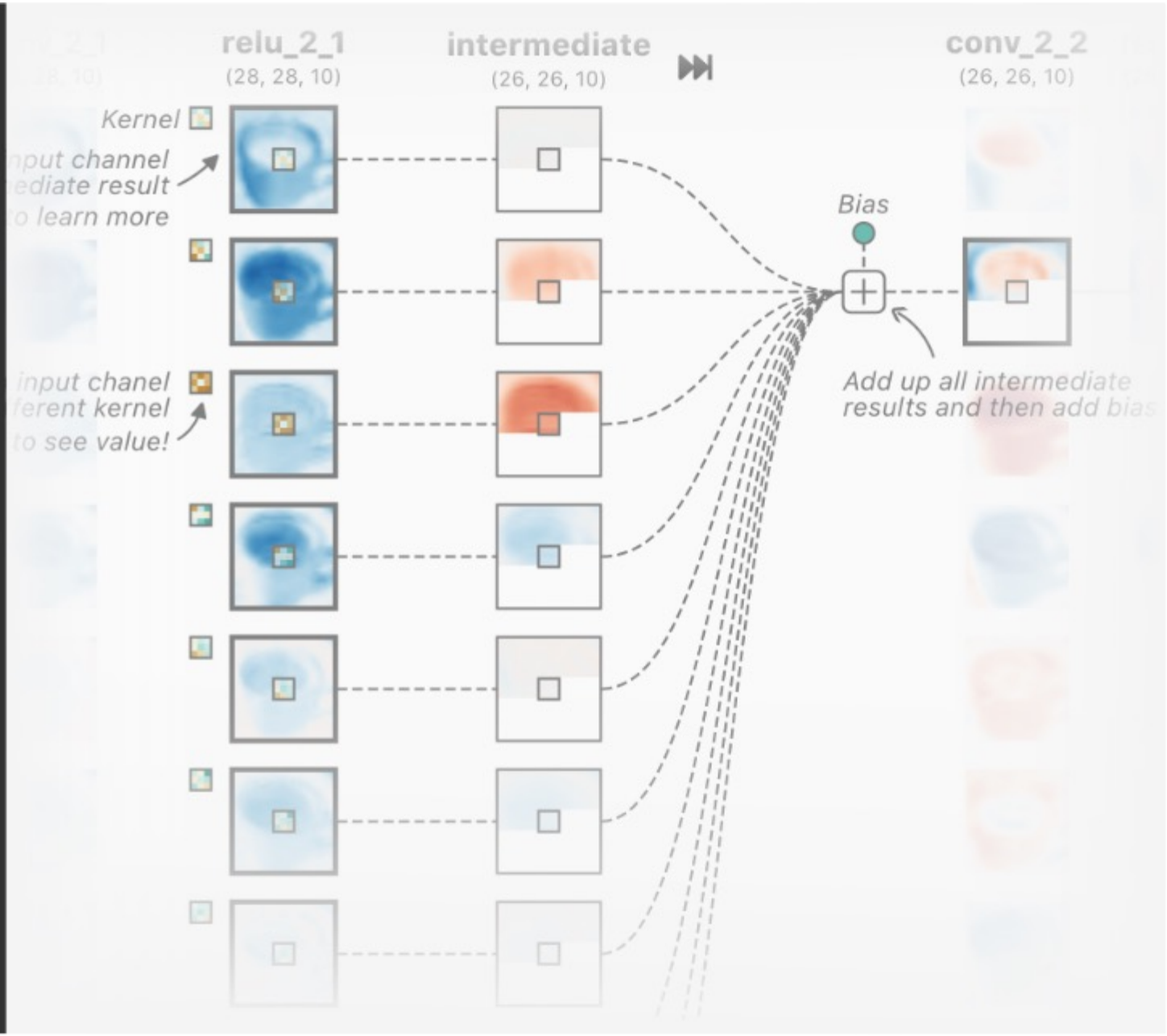
two more layers to go: POOL/FC



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

CNN Explainer

Learn Convolutional Neural Networks in your browser!



https://poloclub.github.io/cnn-explainer/

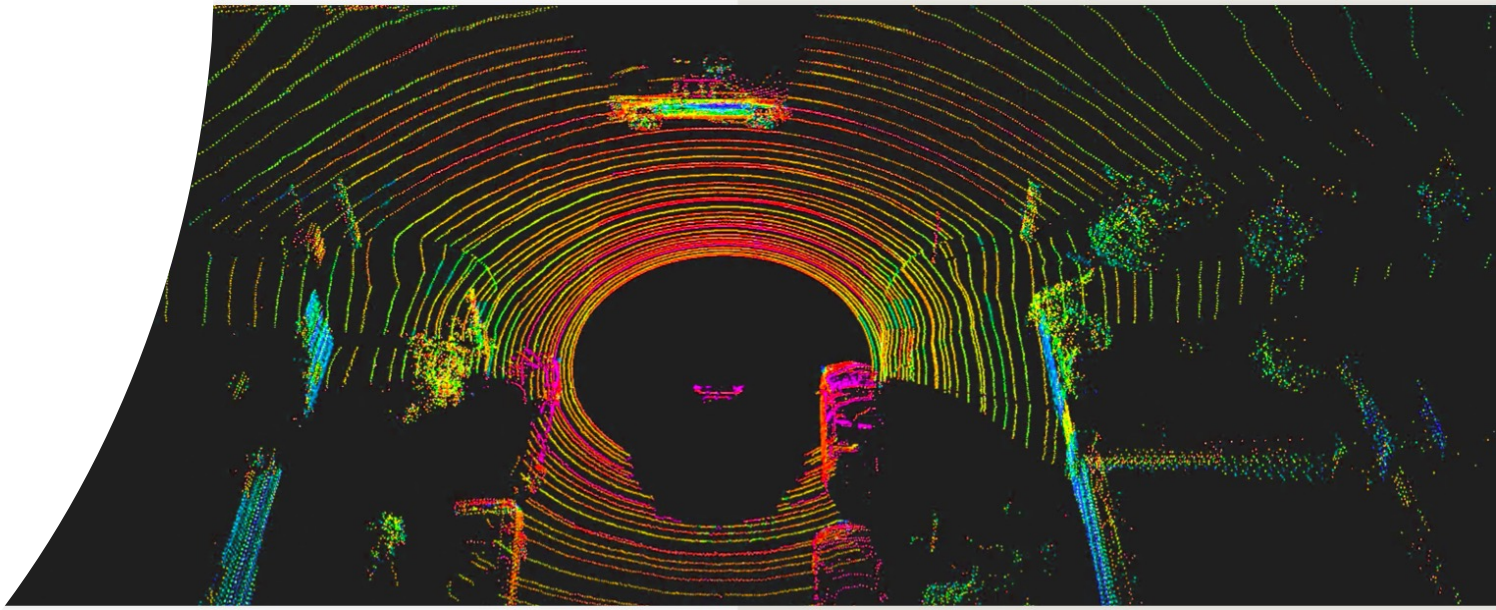
Sensing: LIDAR

If we want to do real-time planning in dynamic environments, we need to be able to sense and understand the environment.

- Cameras provide enormous amounts of data, but computer vision remains a difficult problem.
- In general, the computation cost and the error rates for computer vision systems have prevented them from being adopted as the sensor of choice for self-driving cars.
- LIDAR is a fast and effective sensor that can be used to build 3D point clouds in real time.
- LIDAR is the sensor of choice for most robotics systems (including self-driving cars) that navigate in the real world.

LIDAR

- Superpowers:
  - 360 Visibility
  - Accurate depth!
- Almost all AV prototypes have them



[Images and exposition take from Voyage Blog post]

https://news.voyage.auto/an-introduction-to-lidar-the-key-self-driving-car-sensor-a7e405590cff

Some Robots that Rely on LIDAR

Images from the internets

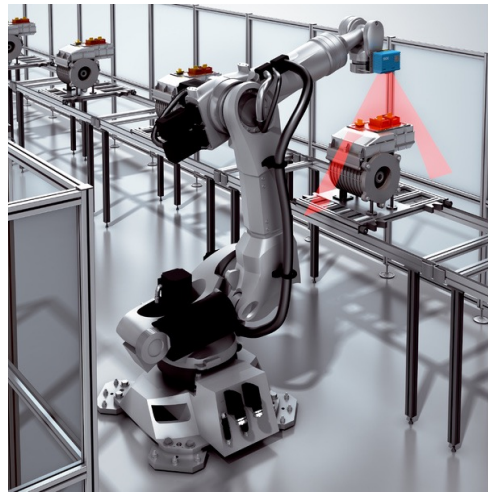
Digit from Agility Robotics

Quadruped at GT (Ye Zhao's lab)

Jackal from Clearpath Robotics

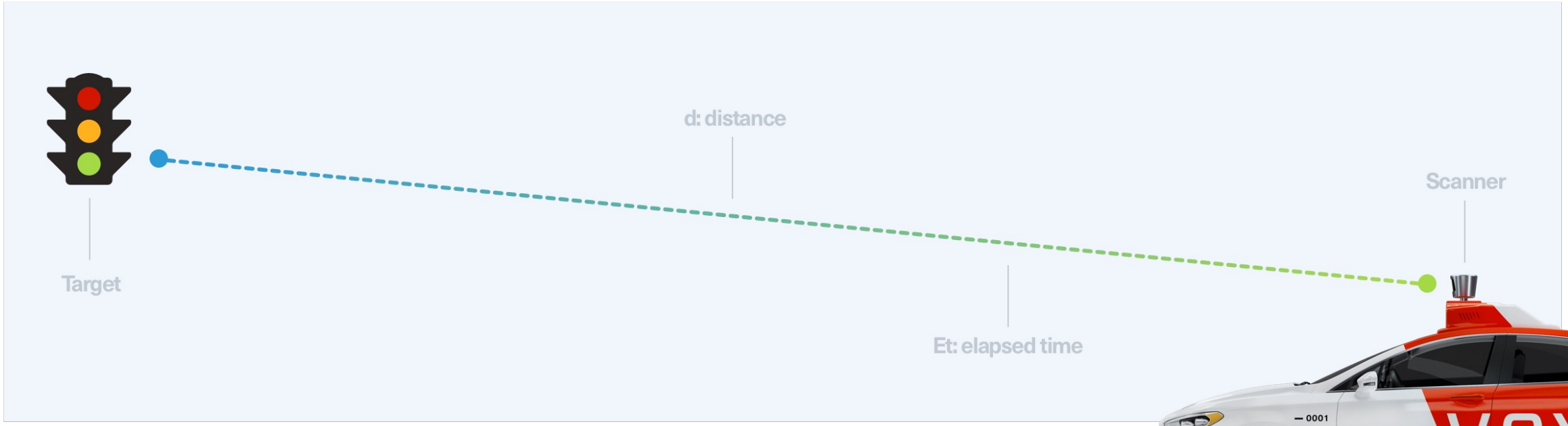
Spot from Boston Dynamics

Yellowscan sensor



SIC sensor

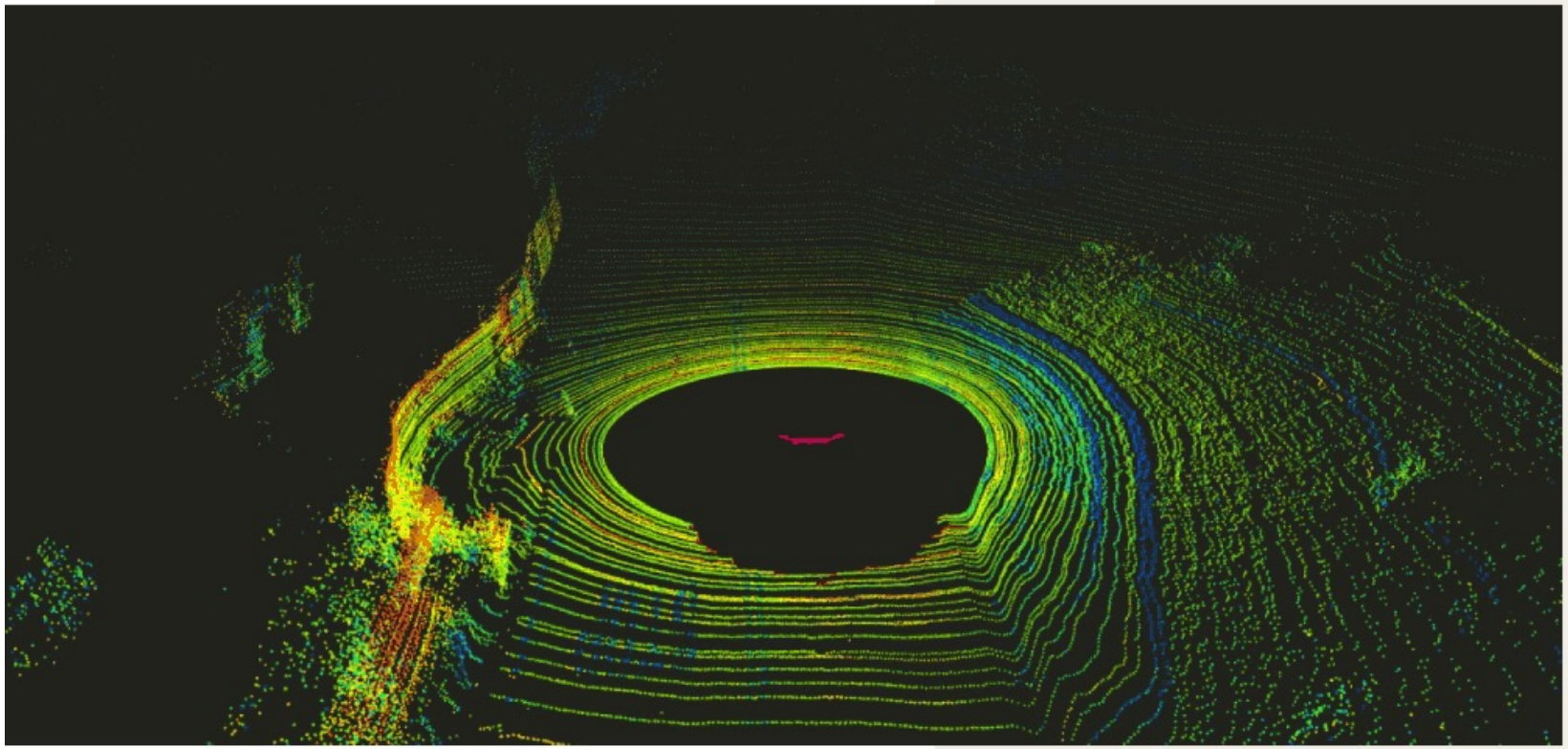
LIDAR Basic Principle



- Send a light pulse
- Measure elapsed time
- Infer distance

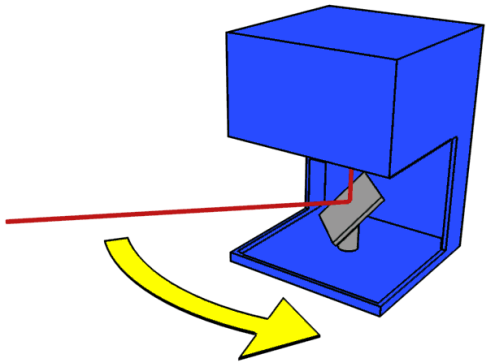
[Images and exposition take from excellent Voyage Blog post]

Example



Images and exposition take from excellent Voyage Blog post

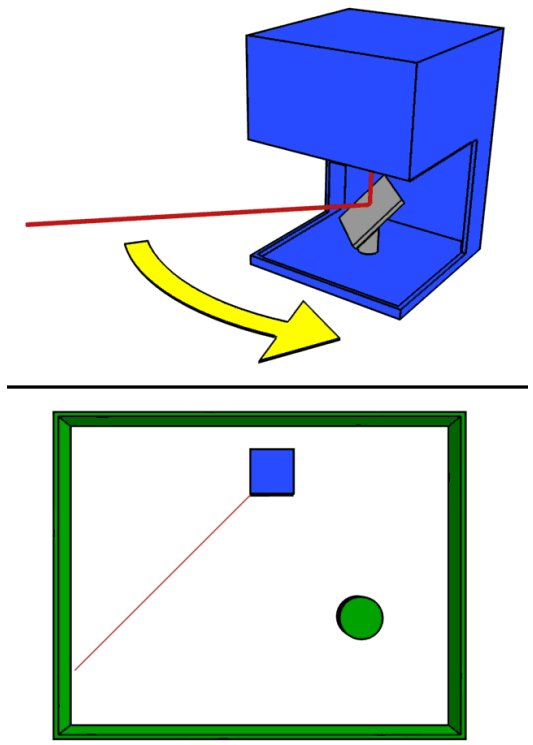
Basic Idea



Sweep a laser beam in a circle.
The beam always lies in a single plane

GIF from Wikipedia

Basic Idea

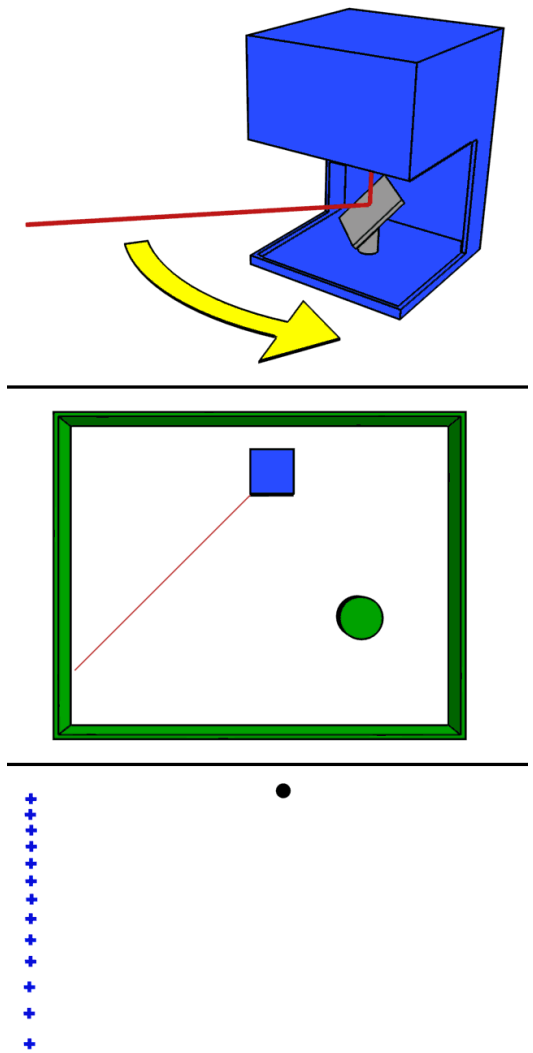


Sweep a laser beam in a circle.
The beam always lies in a single plane

Time of flight of the beam can be measured.
Time of flight is proportional to distance.

GIF from Wikipedia

Basic Idea



Sweep a laser beam in a circle.
The beam always lies in a single plane

Time of flight of the beam can be measured.
Time of flight is proportional to distance.

Compute distance at a discrete set of beam angles.
Once the beam “hits” an object, anything behind the object is occluded.

GIF from Wikipedia

Time of Flight

Emitted laser pulse

Reflected laser pulse

d

This assumes that the motion of the pedestrian is slow relative to the speed of light…

Standard equation: speed × time = distance

$$ct = 2d \rightarrow \boldsymbol{d = 0.5ct}$$

In our case:

- c is the speed of light.
- t is the time from emission to measured return.
- Distance is $2d$, since the light travels to the target, and then returns.

Example:

For an object 15 meters from the detector:

$$0.015 = 300000 \times \frac{t}{2}$$

$$t = \frac{0.030}{300000} = 1 \times 10^{-7} = 0.1\ \mu s$$

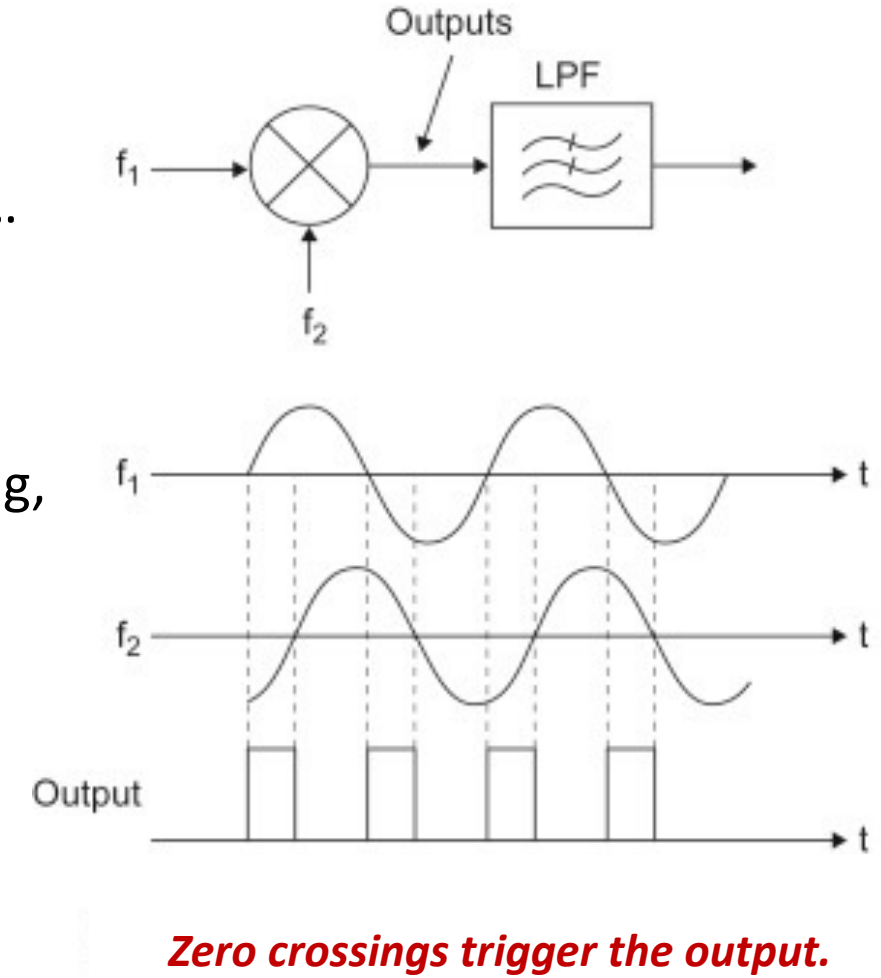
➢ ***We typically don't apply this method literally, because measuring elapsed time between pulses is difficult.***

Coherent Detection

- Instead of measuring time of flight directly for a single pulse…
- ➢ *Emit a periodic signal (e.g., a sine wave), and measure the phase shift between the emitted and received signals.*

- There are plenty of clever circuits that can do this sort of thing, e.g., using a phase locked loop to “lock onto” the received signal.

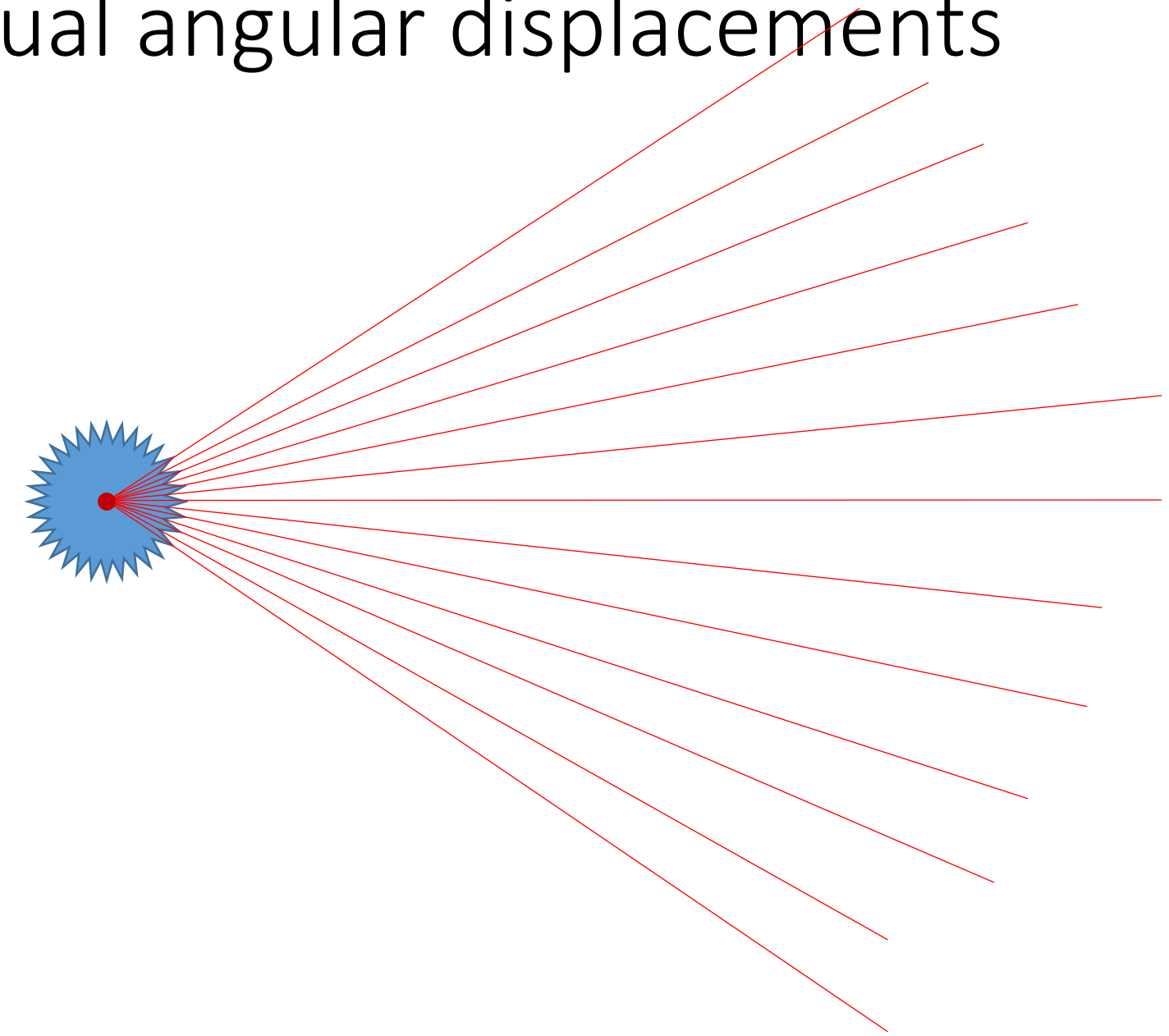
Here, f_1 is the source signal, f_2 is the received signal, and the width of the pulse is the phase shift, which is equal to the elapsed time of flight.



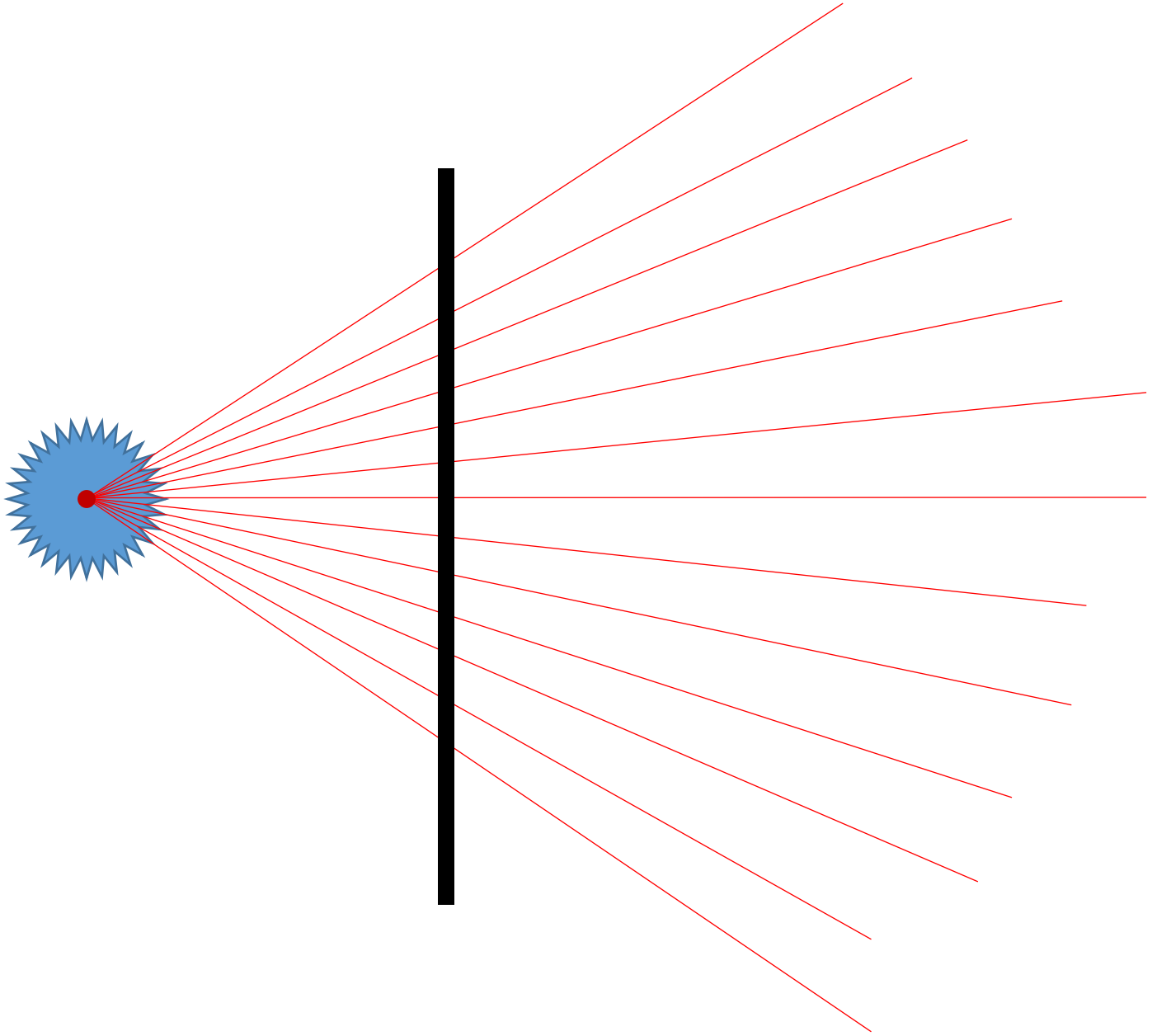
Zero crossings trigger the output.

Figure from: Yongheng Yang, Keliang Zhou, Chapter 4 - Modeling and Control of Single-Phase AC/DC Converters, Editor(s): Frede Blaabjerg, Control of Power Electronic Converters and Systems, Academic Press, 2018, Pages 93-115.

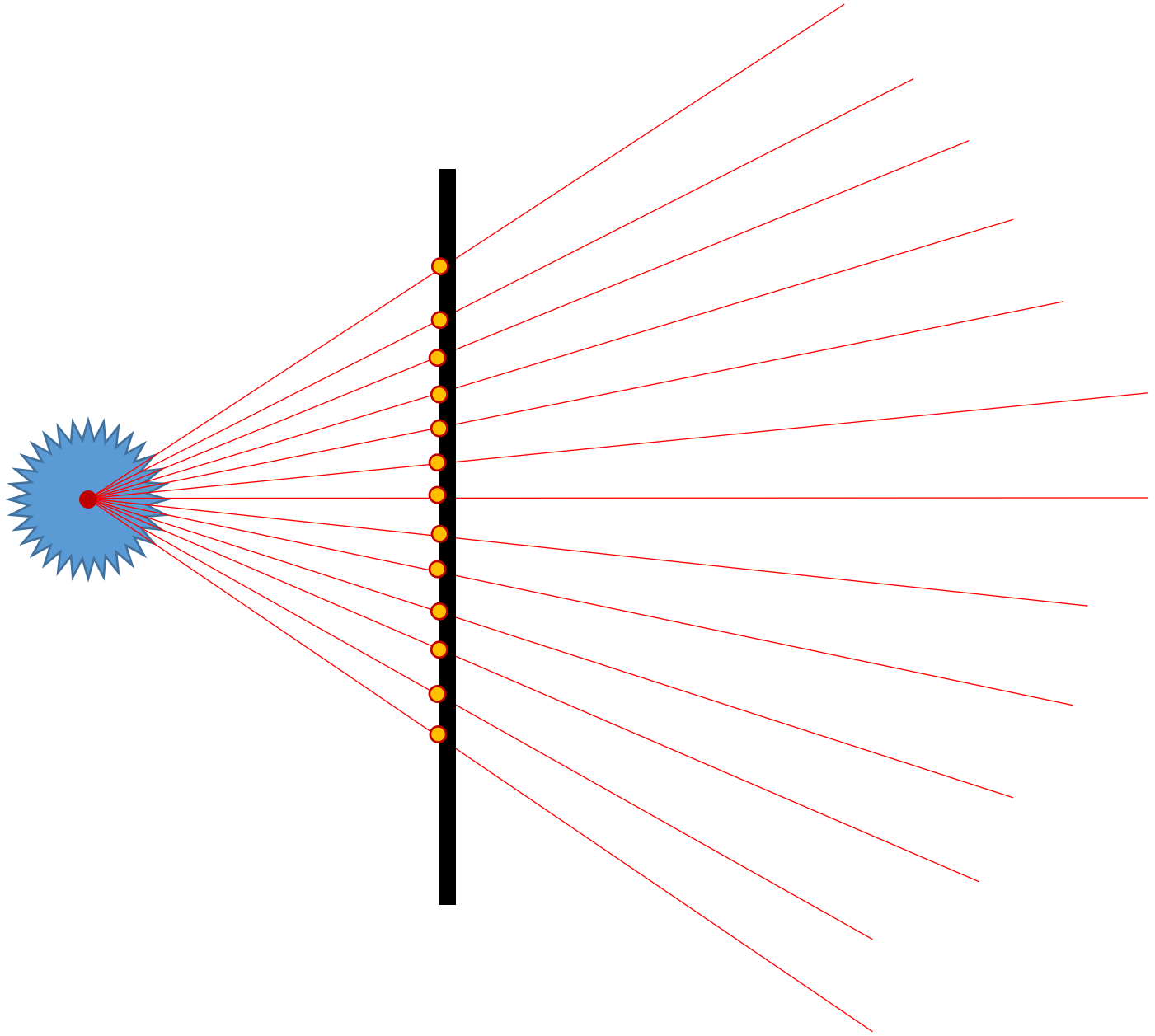
Equal angular displacements



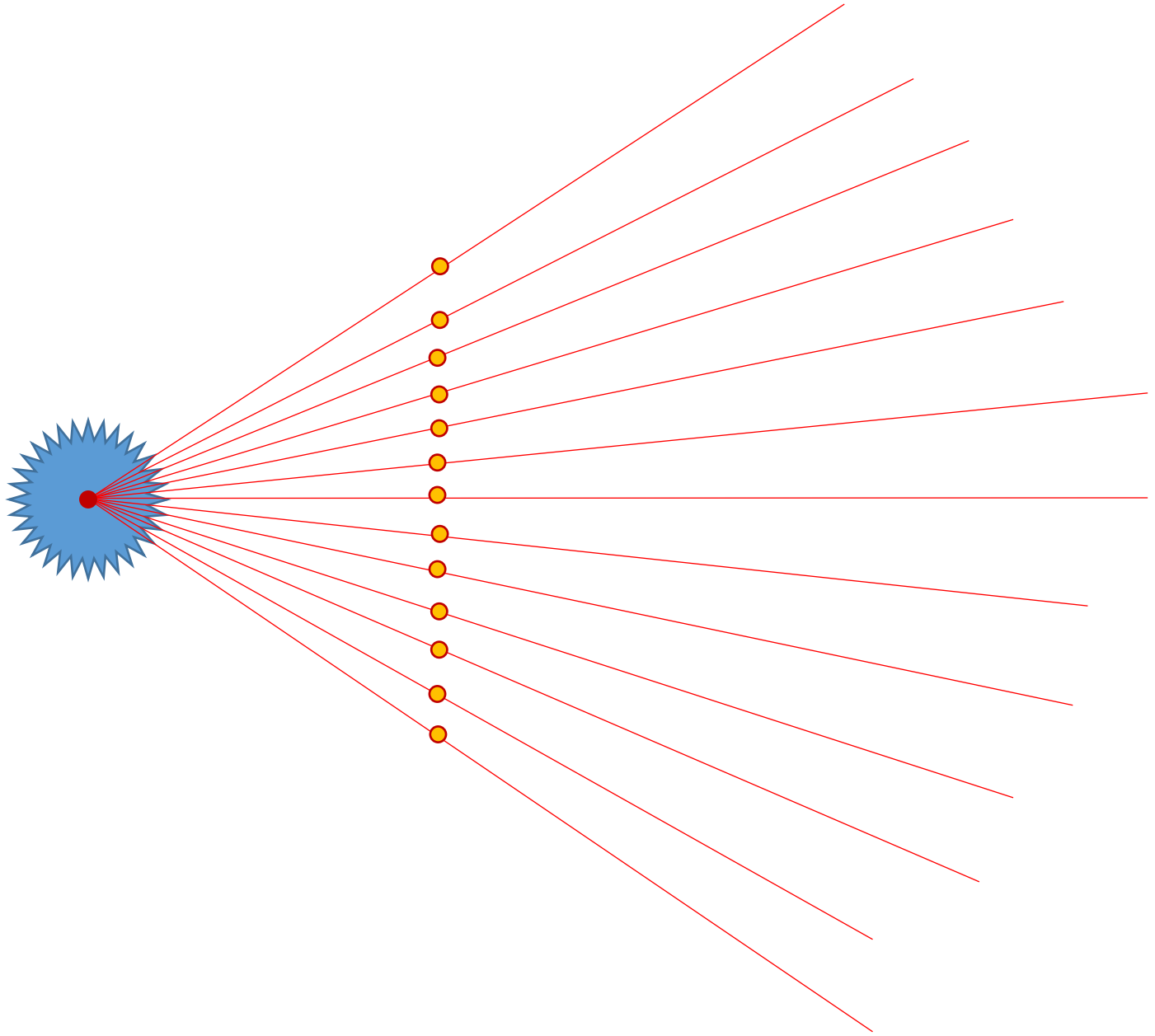
The fact that the measurements are taken at equally spaced orientations does not imply that the measured data (i.e., the point cloud) will be uniformly distributed in space.



Consider a single long wall. The orientation of the wall relative to the sensor determines the distribution of data in the point cloud.

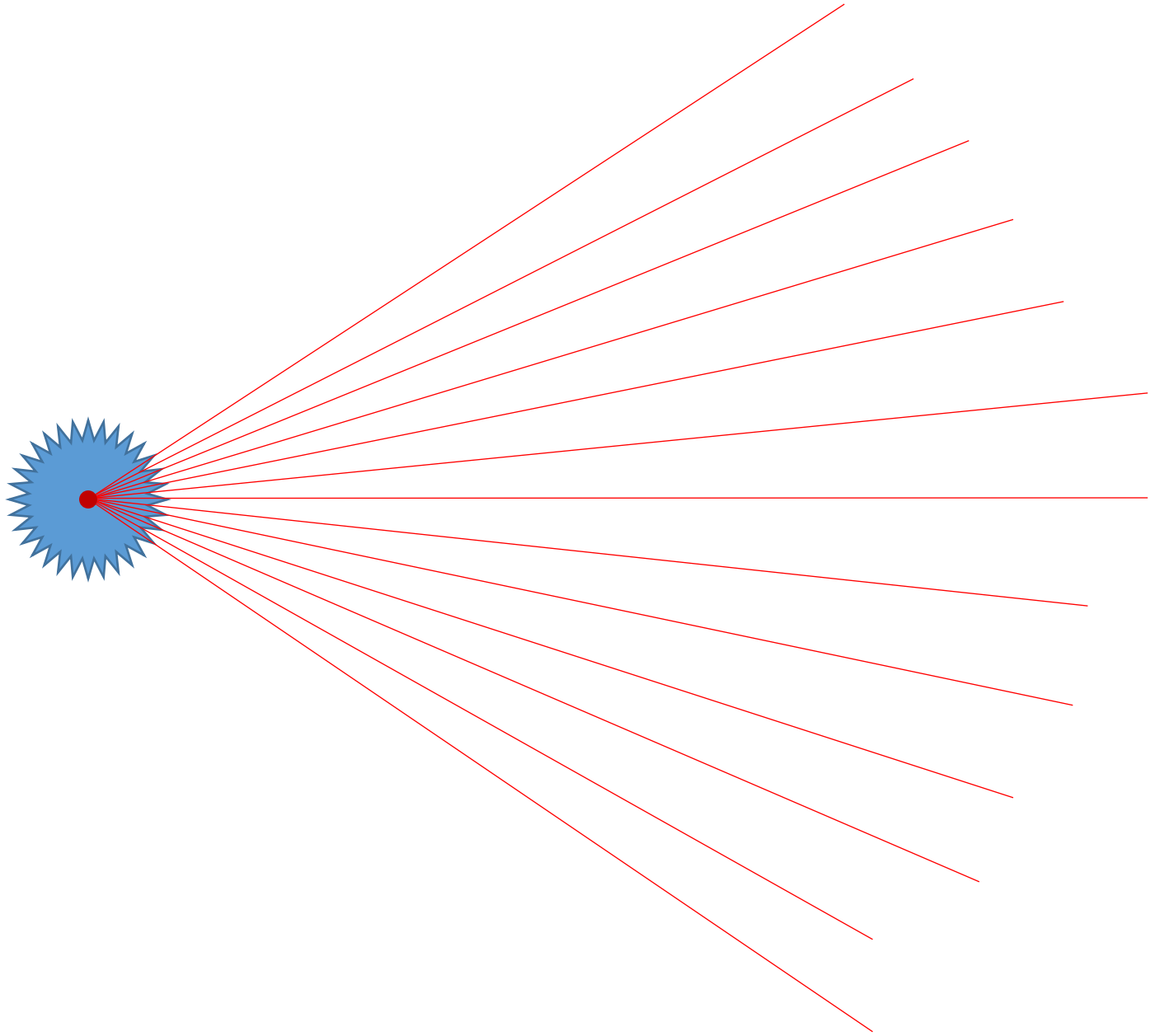


Consider a single long wall. The orientation of the wall relative to the sensor determines the distribution of data in the point cloud.

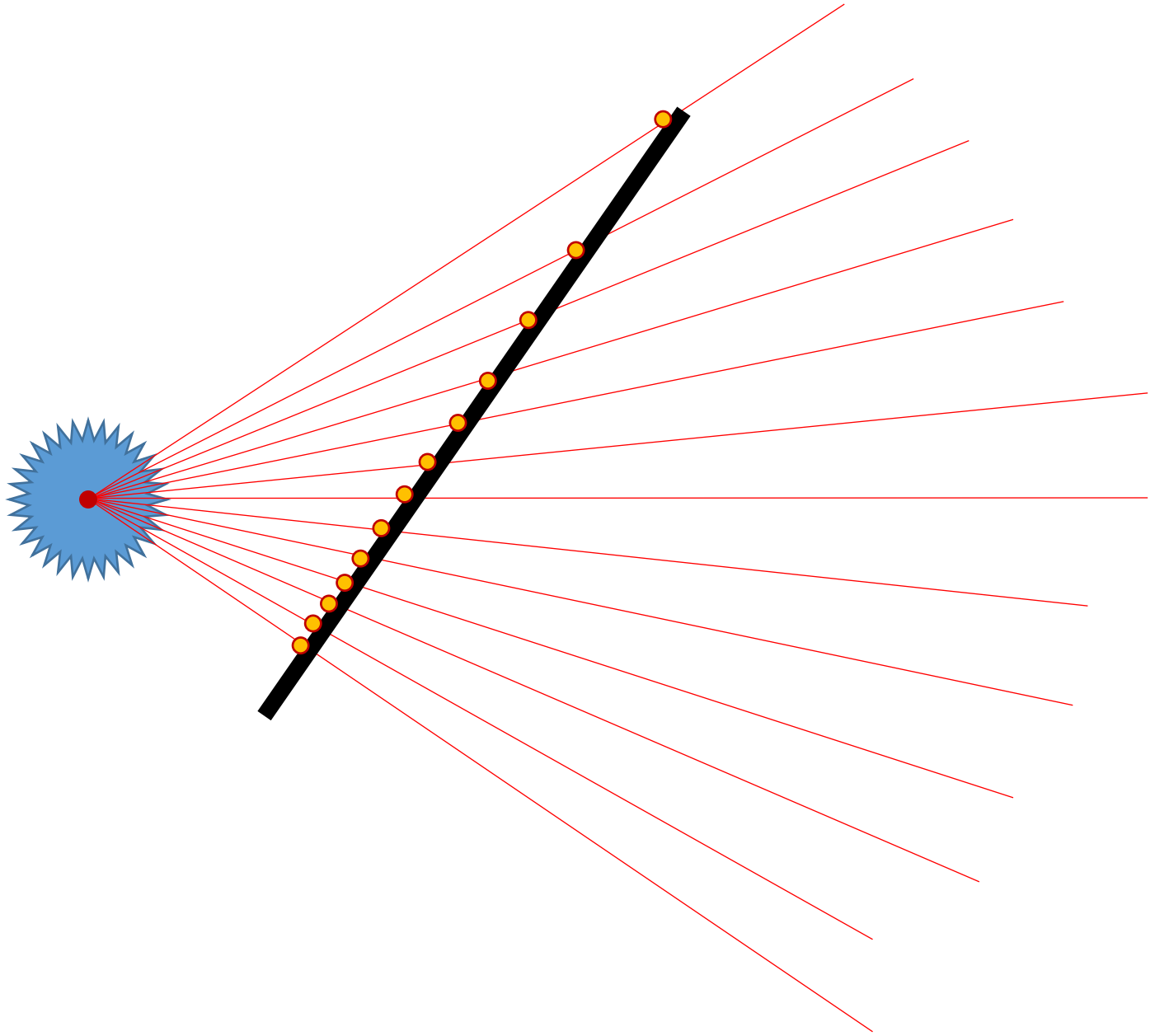


Consider a single long wall. The orientation of the wall relative to the sensor determines the distribution of data in the point cloud.

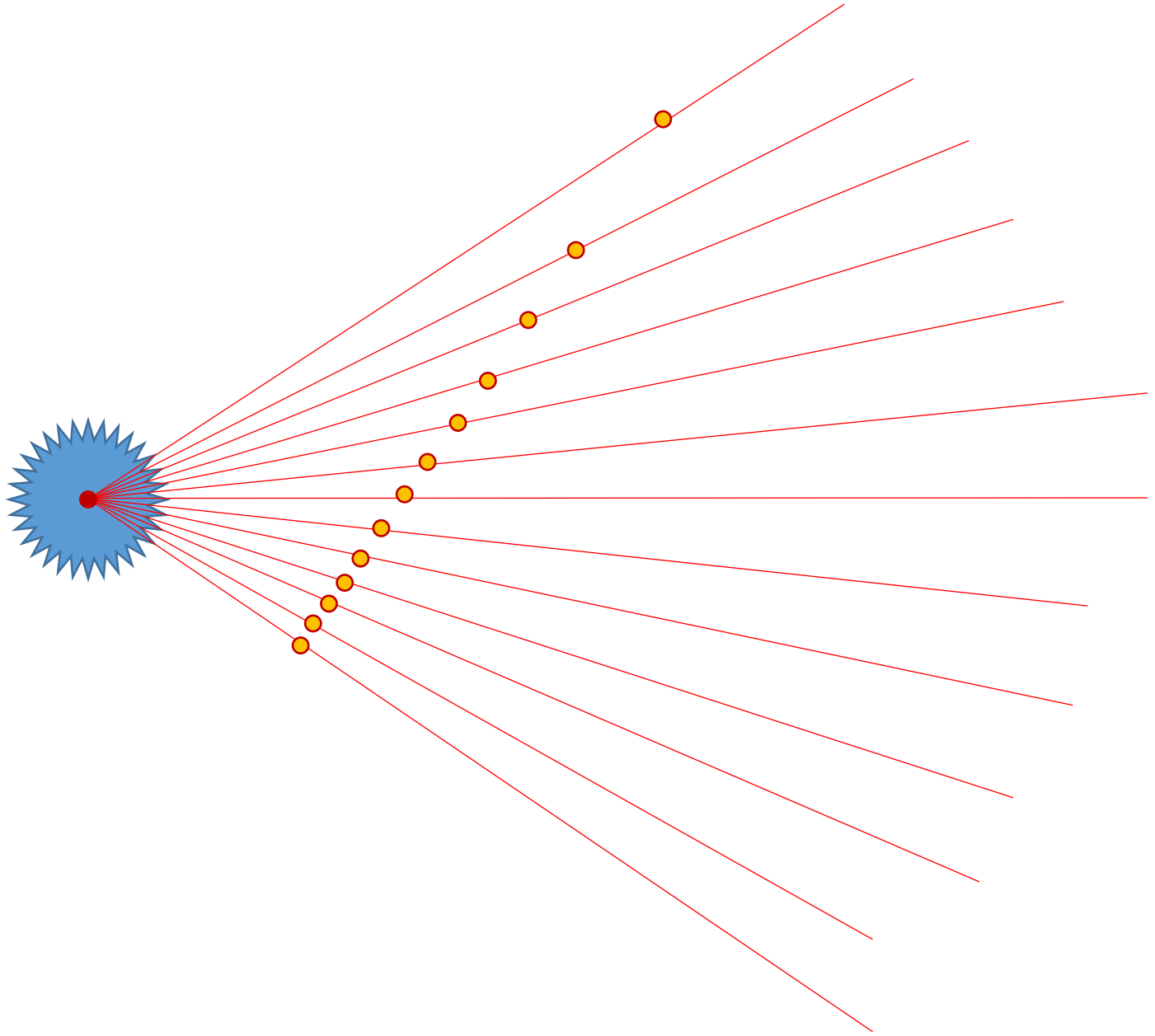
In this example, the data are spaced uniformly in space (approximately).



Suppose, however, that the wall is at an oblique angle.



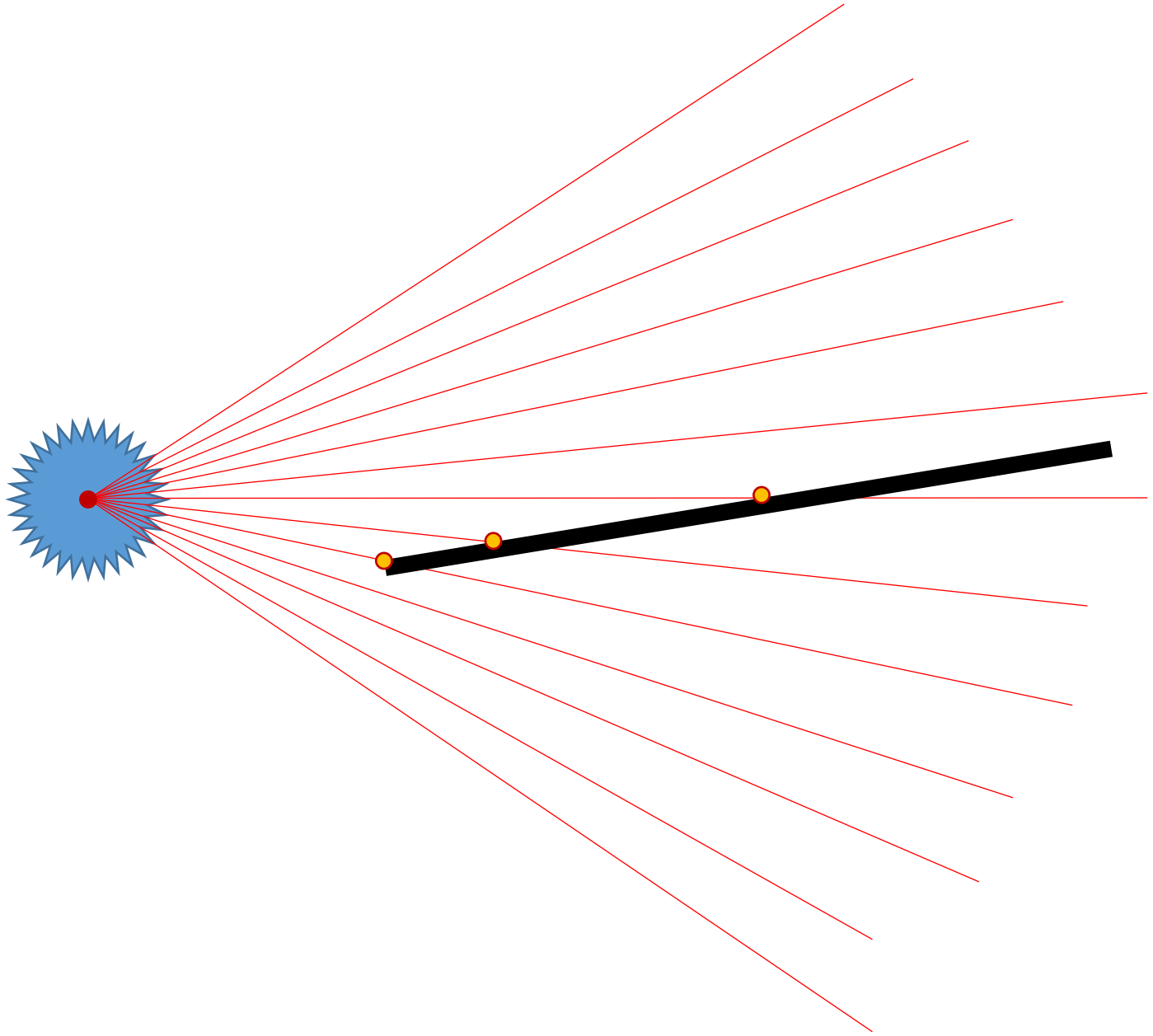
Suppose, however, that the wall is at an oblique angle.



Suppose, however, that the wall is at an oblique angle.

In this case, that data are not at all uniformly spaced.

Sampling uniformity can result in parts of the scene with poor resolution.



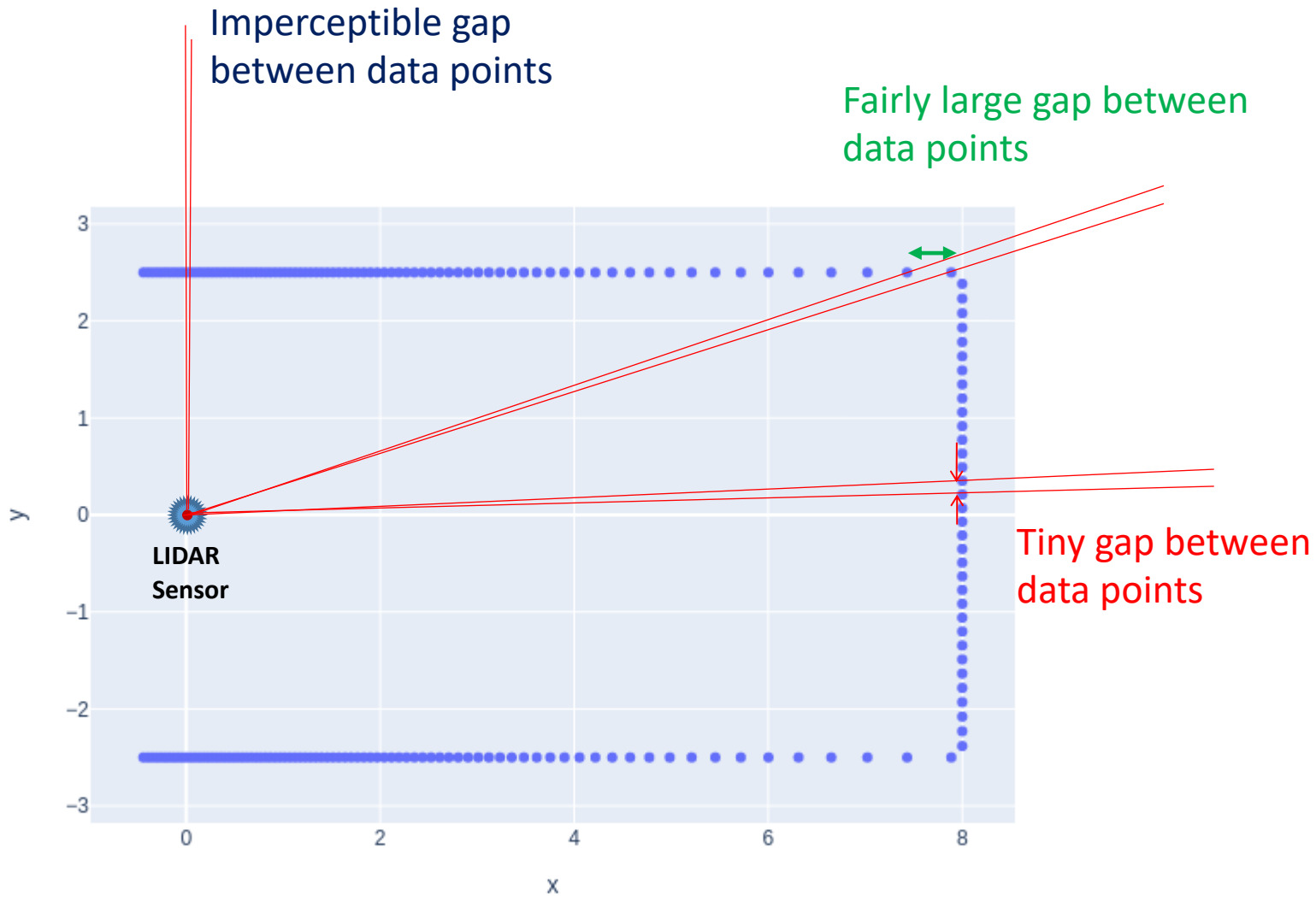
In this case, a fairly large obstacle in the world is represented by only three data points!

You can see these effects in real LIDAR data images.

An Example from the Book

A robot enters a large, rectangular room, and uses its LIDAR sensor to build a model of the room.

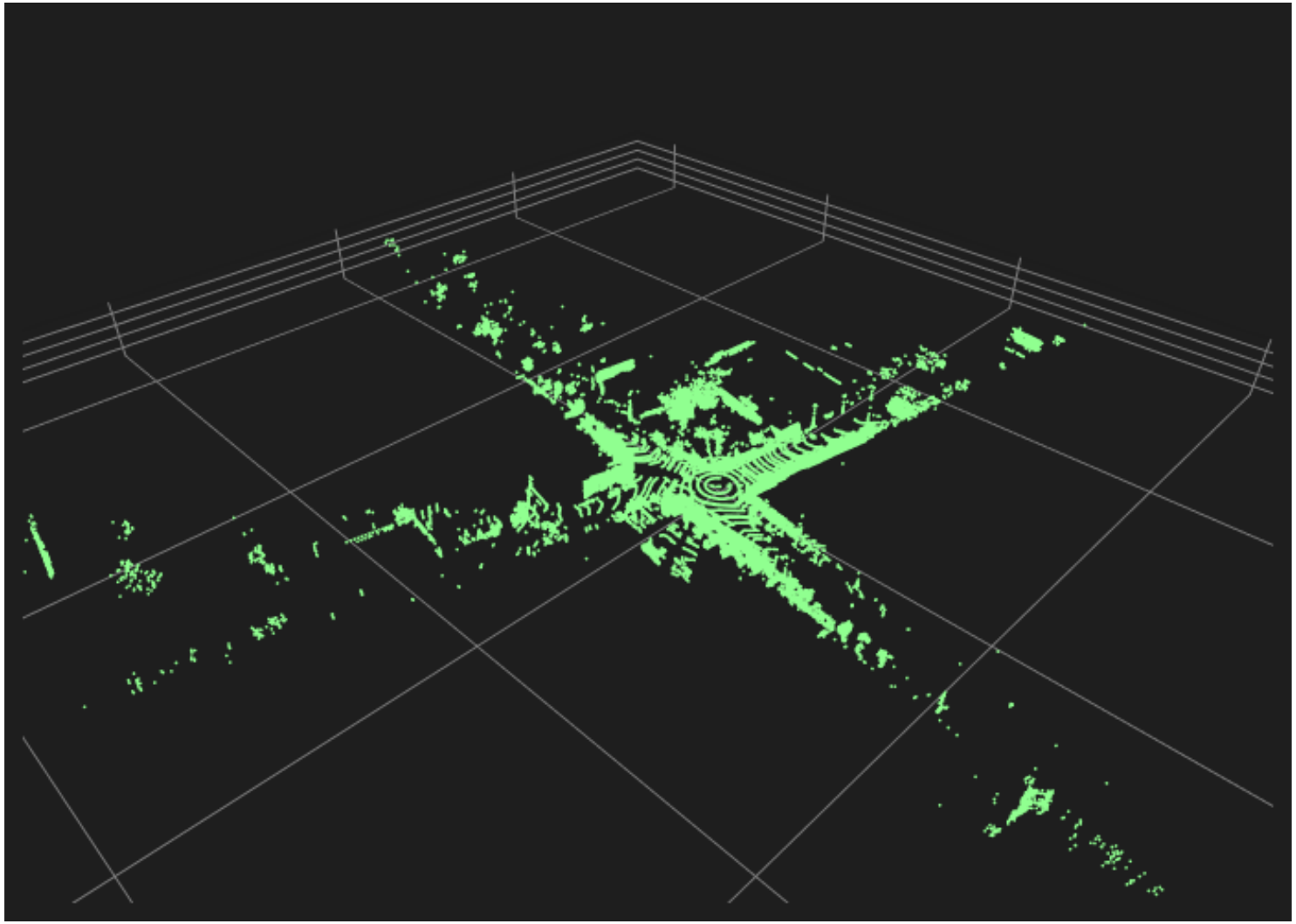
- Points that are nearby and on a surface perpendicular to rays give dense data.
- Points that are distant and on walls that are not parallel to rays give sparse data.
- Points that are distant, but on walls that are perpendicular to rays give moderately dense data.



Real Data (from the book)

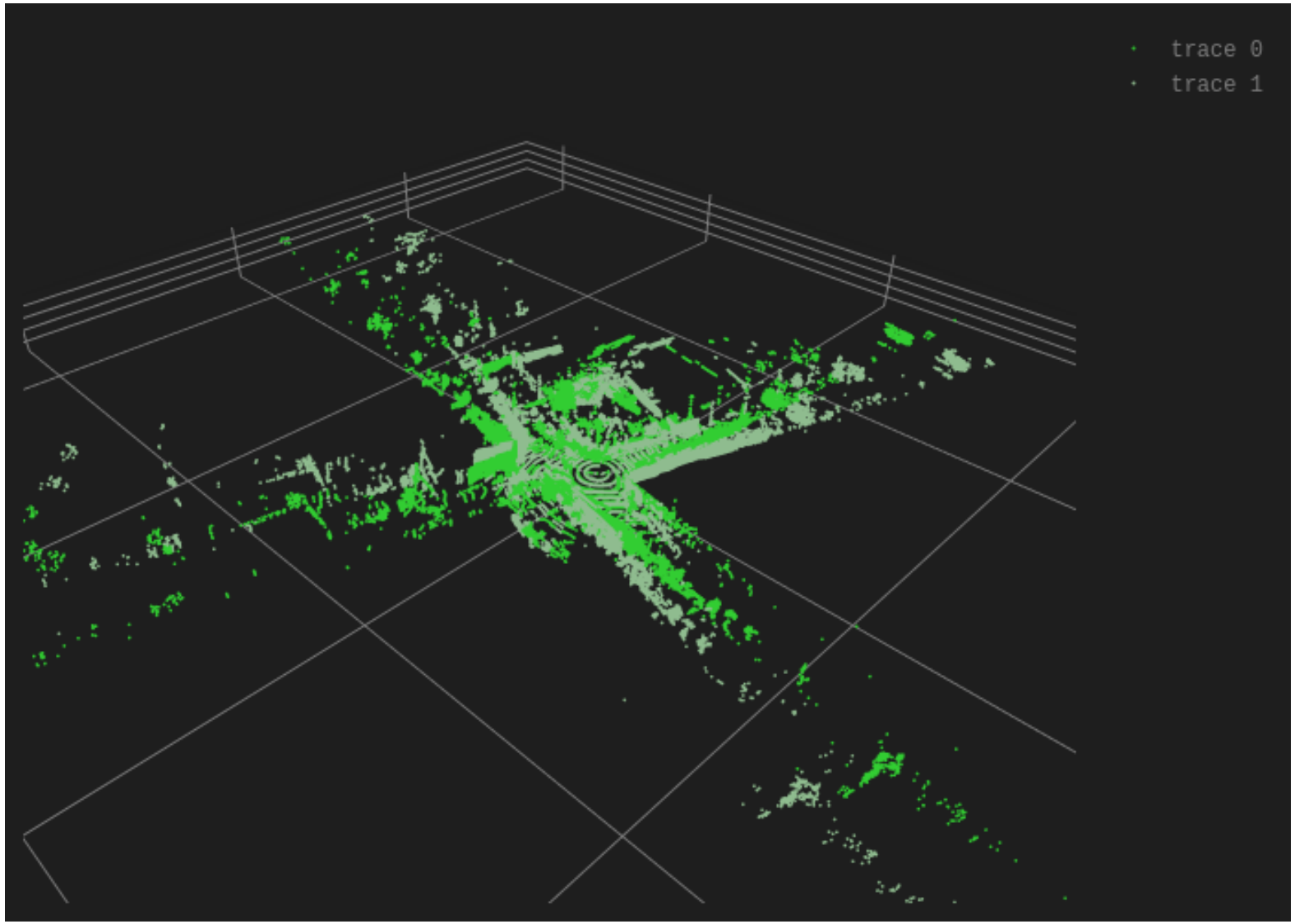
Scan taken when the Argo test car is turning at an intersection:

- The location of the car (at the origin) is marked by concentric circles formed by the lowest beams.
- The range is approximately 200m, so we can see fairly far down the cross streets.
- Occlusion is significant: objects close to the car throw “occlusion shadows”.
- Everything is in *body coordinates*, and the fact that the streets appear rotated betrays that the car is actually turning.



Real Data (from the book)

- This image shows two scans
  - two slightly different colors for the data points.
- Both scans are shown w.r.t. the body coordinate frame of the car.
- These aren't successive scans; eight scans have been omitted.
- Notice the difference in orientation of the data: The car is turning!
- Notice also a slight translation: The car is also moving forward.



Building a Point Cloud

- As the sensor moves through its environment, it performs a sequence of scans.
- Each scan acquires a set of points in the world, and the coordinates of these points (either (x, y) for planar data or (x, y, z) for 3D data) are ***expressed with respect to the coordinate frame of the sensor***.
- To make life easier, we will assume that the body frame of the robot is coincident with the coordinate frame of the sensor.
- In order to build a single map of the environment, it is necessary to express these points in a single coordinate system.
- For most applications, we map points to a single, world coordinate frame.
- This frame could be defined in terms of the local environment (e.g., the origin of the fame could be in Klaus), or in terms of GPS coordinates.
- We will learn how to do this using coordinate transformations!