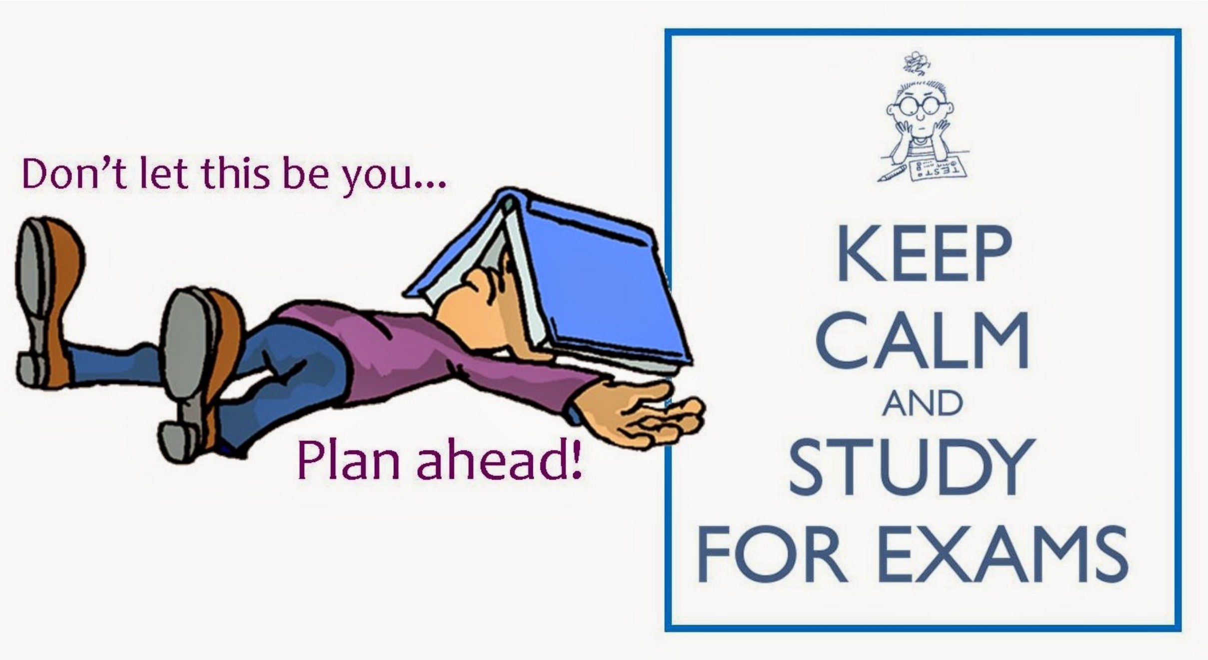
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 24: Review

Final Exam Details

2.5 hours

Closed book, **no** calculators

— Equations will be given

Format similar to midterm exam

— Part A: Multiple-part true/false

— Part B: Short answer

No coding questions

How to study?

— Look at the Lectures Notes and Assignment and think critically if you **truly** understand the material

— Look at each algorithm, concept,

- — what are properties of the algorithm / concept?
- — what does each step do?
- — is this step important? can you imagine doing it another way?
- — what are parameters? what would be the effect of changing those?

piazza

Course Review: Cameras and Lenses

Pinhole **camera**

Projections (and projection equations)

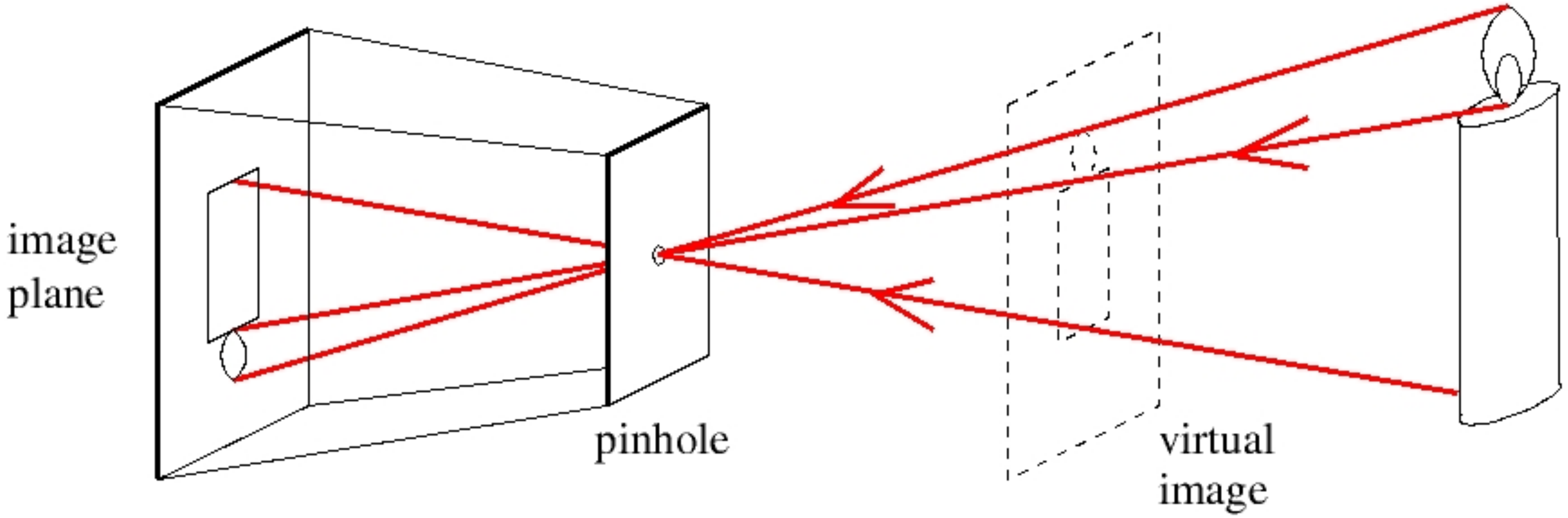
— perspective, weak perspective, orthographic

Lenses

Human **eye**

Pinhole Camera

A pinhole camera is a box with a small hall (**aperture**) in it



Forsyth & Ponce (2nd ed.) Figure 1.2

Summary of **Projection Equations**

3D object point $P = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ projects to 2D image point $P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ where

Perspective	$x' = f' \frac{x}{z}$ $y' = f' \frac{y}{z}$	
Weak Perspective	$x' = m\,x$ $y' = m\,y$	$m = \frac{f'}{z_0}$
Orthographic	$x' = x$ $y' = y$	

Why **Not** a Pinhole Camera?

— If pinhole is **too big** then many directions are averaged, blurring the image

— If pinhole is **too small** then diffraction becomes a factor, also blurring the image

— Generally, pinhole cameras are **dark**, because only a very small set of rays from a particular scene point hits the image plane

— Pinhole cameras are **slow**, because only a very small amount of light from a particular scene point hits the image plane per unit time

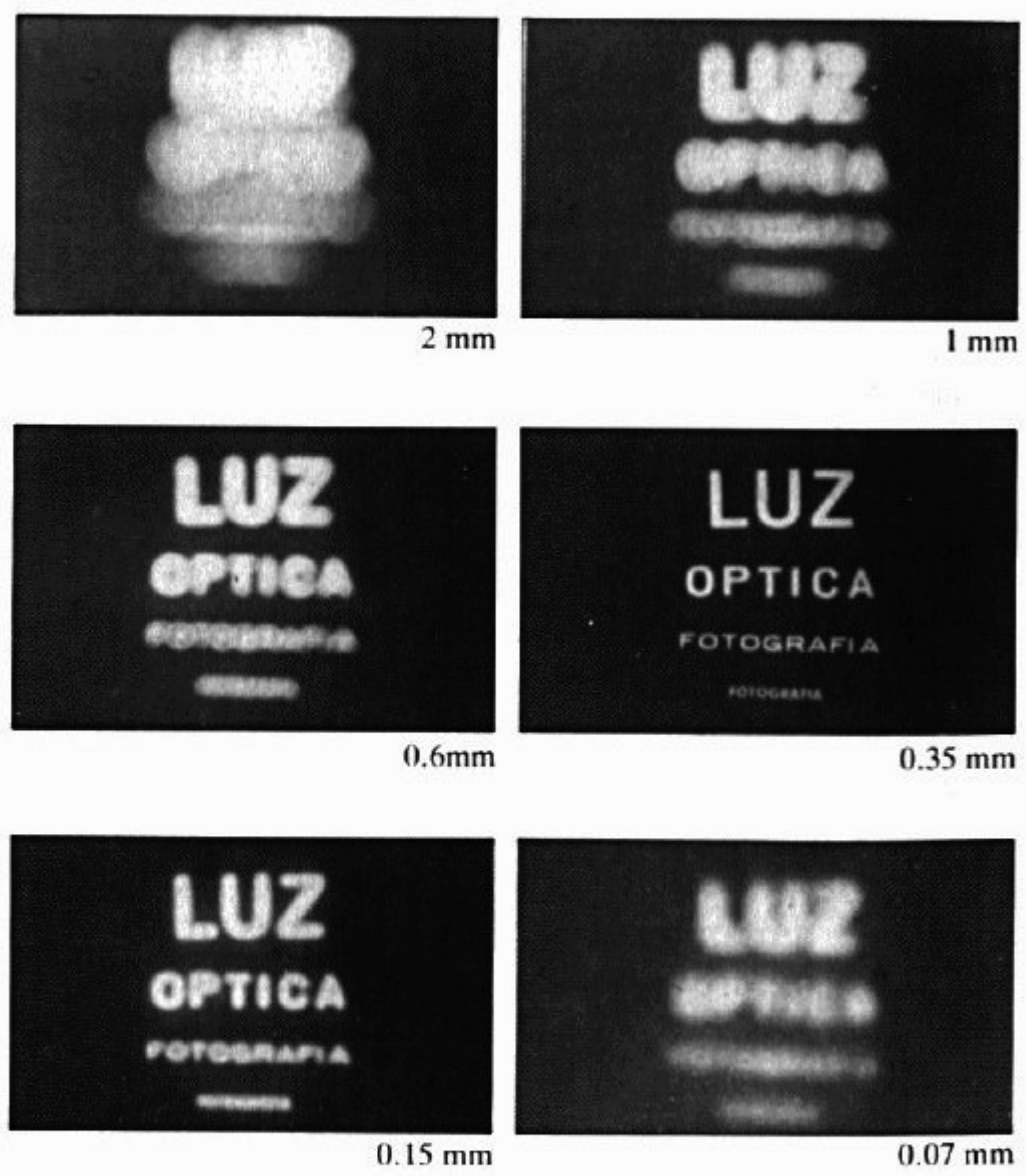
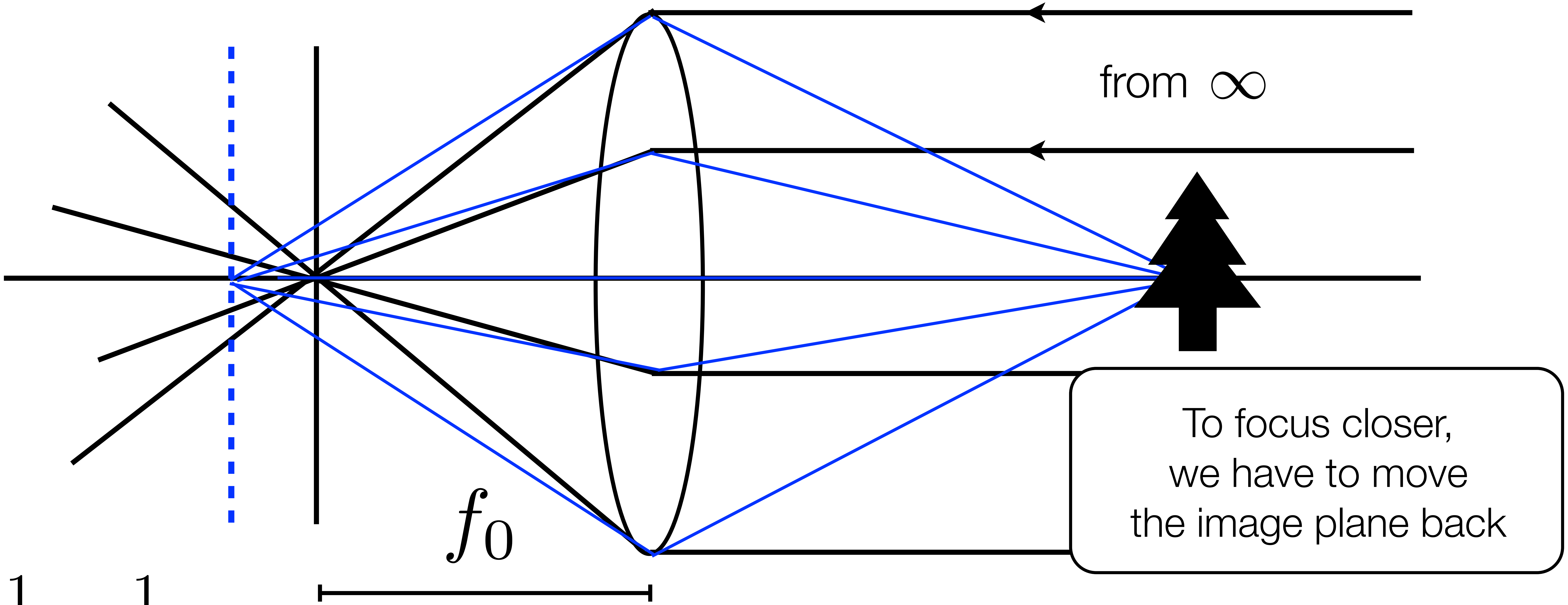


Image Credit: Credit: E. Hecht. "Optics," Addison-Wesley, 1987

Lens Basics

A lens focuses parallel rays (from points at infinity) at focal length of the lens

Rays passing through the center of the lens are not bent



$$\frac{1}{z'} - \frac{1}{z} = \frac{1}{f}$$

Course Review: Filters

Correlation and **convolution**

Box, pillbox, Gaussian filters

Separability

Non-linear filters: median, bilateral

Template matching

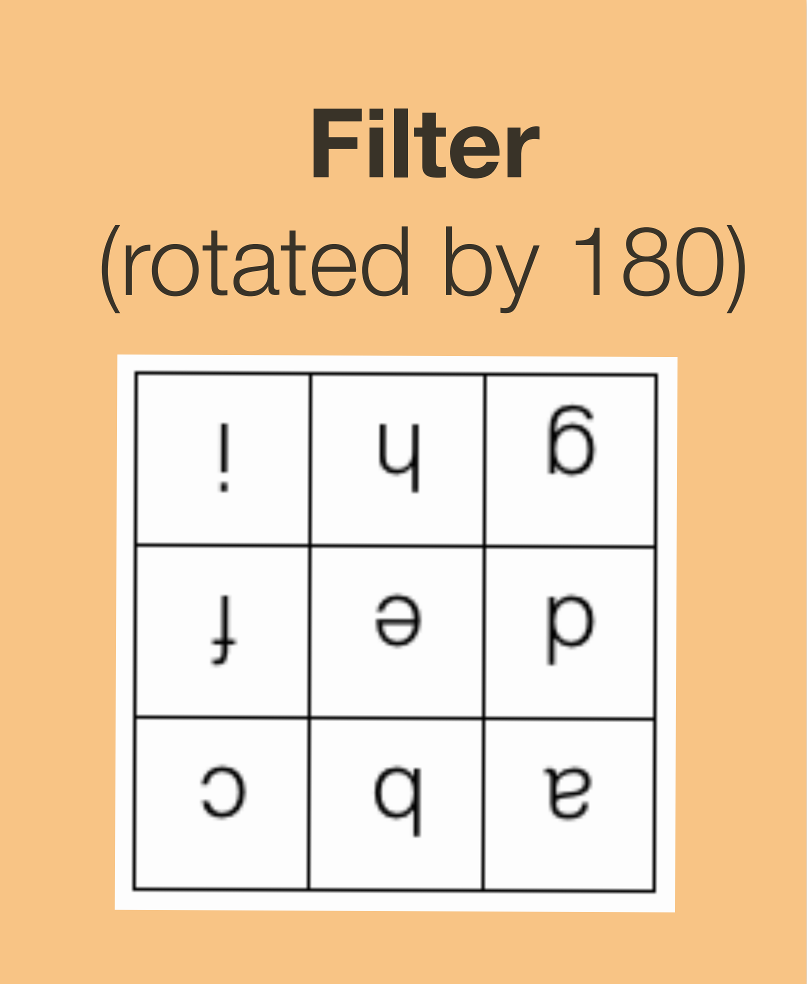
Linear Filters: Correlation vs. Convolution

Definition: **Correlation**

$$I'(X,Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(i,j) I(X+i, Y+j)$$

Definition: **Convolution**

$$I'(X,Y) = \sum_{j=-k}^{k} \sum_{i=-k}^{k} F(i,j) I(X-i, Y-j)$$

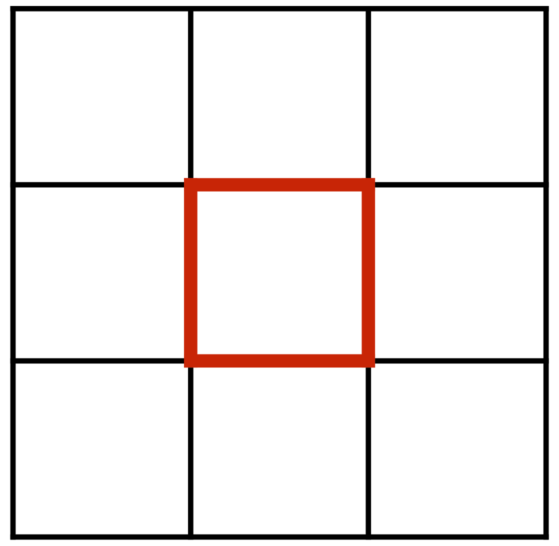


a	b	c
d	e	f
g	h	i

Filter

1	2	3
4	5	6
7	8	9

Image



Output

= 9a + 8b + 7c
+ 6d + 5e + 4f
+ 3g + 2h + 1i

Separability: Box Filter Example

Standard (3x3)

image $I(X,Y)$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$F(X,Y) = F(X)F(Y)$$

filter

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

output $I'(X,Y)$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	0	10	20	30	30	30	20	10	
	10	10	10	10	0	0	0	0	
	10	30	10	10	0	0	0	0	

Separable

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F(X)$

filter

$\frac{1}{3}$

1	1	1

	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	
	0	30	60	90	90	90	60	30	
	0	30	60	90	90	90	60	30	
	0	30	30	60	60	90	60	30	
	0	30	60	90	90	90	60	30	
	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	
	30	30	30	30	0	0	0	0	
	0	0	0	0	0	0	0	0	

$F(Y)$

filter

$\frac{1}{3}$

1
1
1

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	0	10	20	30	30	30	20	10	
	10	10	10	10	0	0	0	0	
	10	30	10	10	0	0	0	0	

Efficient Implementation: **Separability**

Naive implementation of 2D **Gaussian**:

At each pixel, (X, Y), there are $m \times m$ multiplications

There are $n \times n$ pixels in (X, Y)

Total: $m^2 \times n^2$ multiplications

Separable 2D **Gaussian**:

At each pixel, (X, Y), there are $2m$ multiplications

There are $n \times n$ pixels in (X, Y)

Total: $2m \times n^2$ multiplications

Speeding Up **Convolution** (The Convolution Theorem)

General implementation of **convolution**:

At each pixel, (X, Y), there are $m \times m$ multiplications

There are $n \times n$ pixels in (X, Y)

Total: $m^2 \times n^2$ multiplications

Convolution if FFT space:

Cost of FFT/IFFT for image: $\mathcal{O}(n^2 \log n)$

Cost of FFT/IFFT for filter: $\mathcal{O}(m^2 \log m)$

Cost of convolution: $\mathcal{O}(n^2)$

Note: not a function of filter size !!!

Bilateral Filter

image $I(X,Y)$

25	0	25	255	255	255
0	0	0	230	255	255
0	25	25	255	230	255
0	0	25	255	255	255

→

image $I(X,Y)$

0.1	0	0.1	1	1	1
0	0	0	0.9	1	1
0	0.1	0.1	1	0.9	1
0	0	0.1	1	1	1

Domain Kernel

$\sigma_d = 0.45$

$$\sum \begin{bmatrix} 0.08 & 0.12 & 0.08 \\ 0.12 & 0.20 & 0.12 \\ 0.08 & 0.12 & 0.08 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0.9 \\ 0.1 & 0.1 & 1 \\ 0 & 0.1 & 1 \end{bmatrix} = 0.3$$

Gaussian Filter (only)

Range Kernel

$\sigma_r = 0.45$

0.98	0.98	0.2
1	1	0.1
0.98	1	0.1

(this is different for each locations in the image)

multiply →

Range * **Domain** Kernel

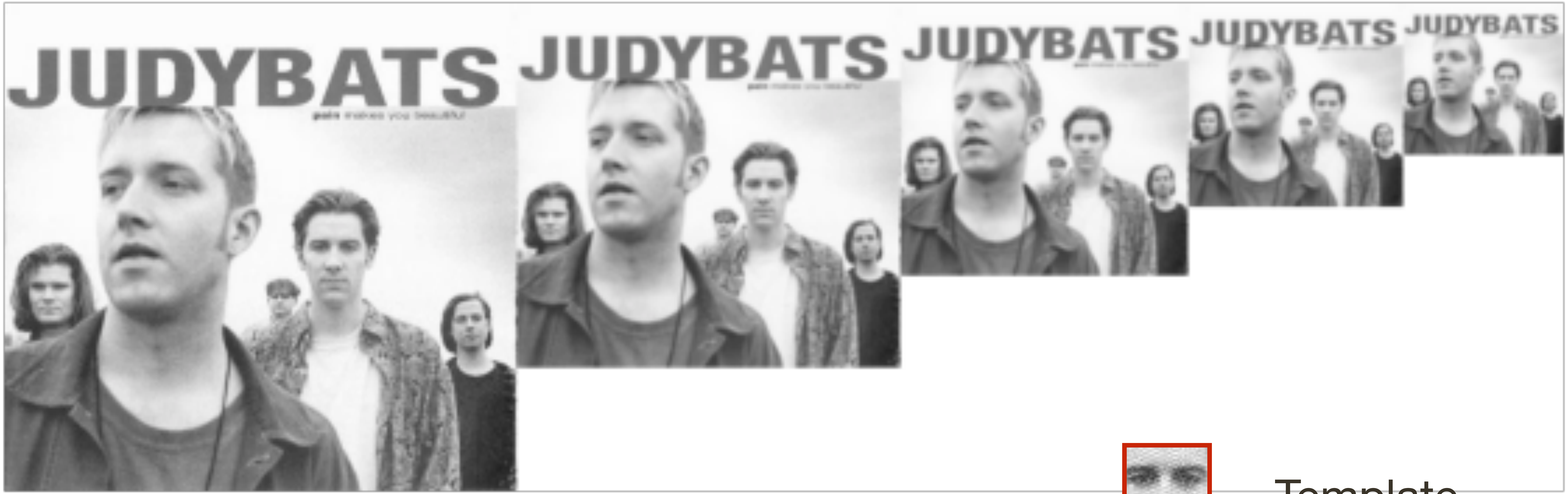
0.08	0.12	0.02
0.12	0.20	0.01
0.08	0.12	0.01

$$\sum \begin{bmatrix} 0.11 & 0.16 & 0.03 \\ 0.16 & 0.26 & 0.01 \\ 0.11 & 0.16 & 0.01 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0.9 \\ 0.1 & 0.1 & 1 \\ 0 & 0.1 & 1 \end{bmatrix} = 0.1$$

Bilateral Filter

Multi-Scale Template Matching

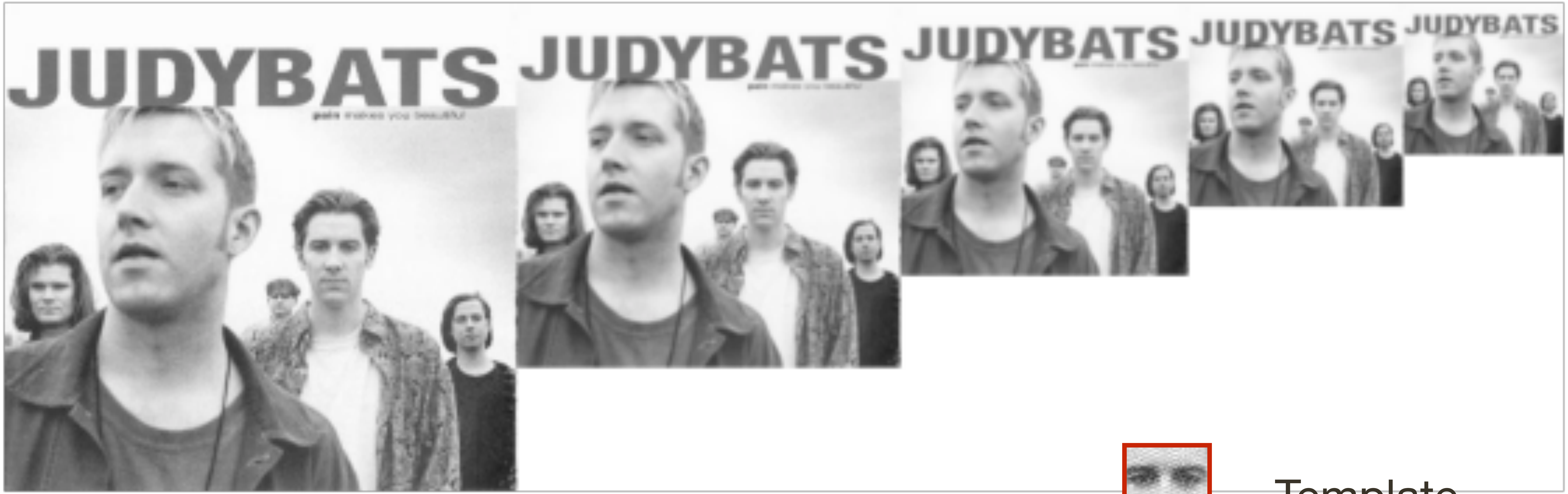
Correlation with a **fixed-sized template** only detects faces at **specific scales**



= Template

Multi-Scale Template Matching

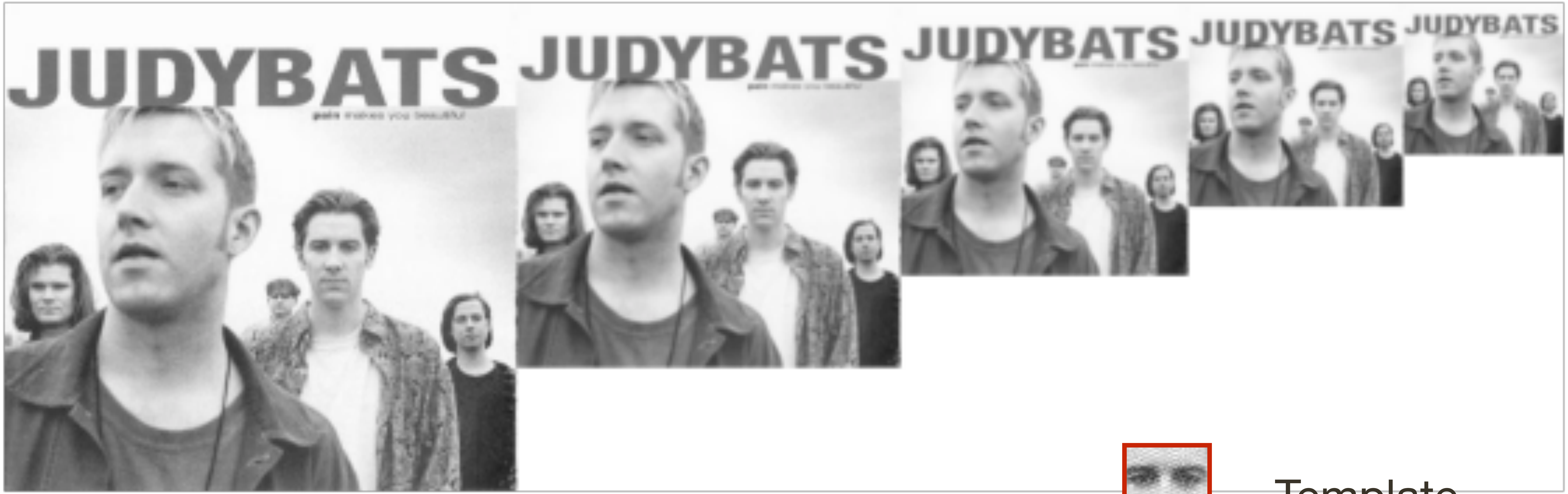
Correlation with a **fixed-sized template** only detects faces at **specific scales**



= Template

Multi-Scale Template Matching

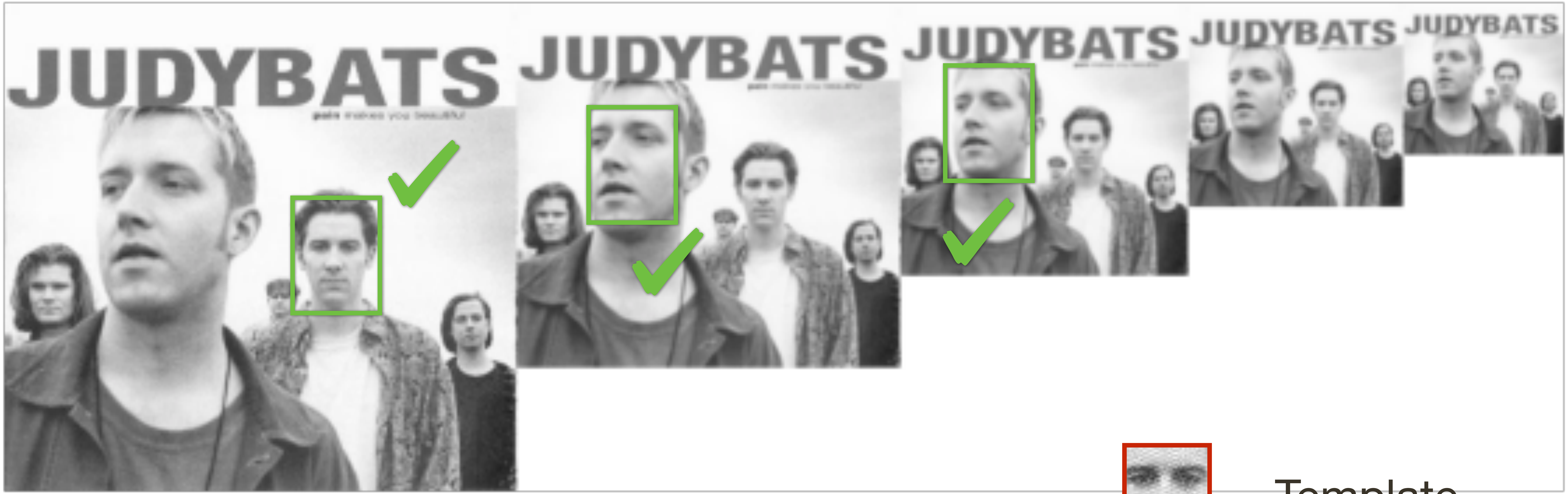
Correlation with a **fixed-sized template** only detects faces at **specific scales**



= Template

Multi-Scale Template Matching

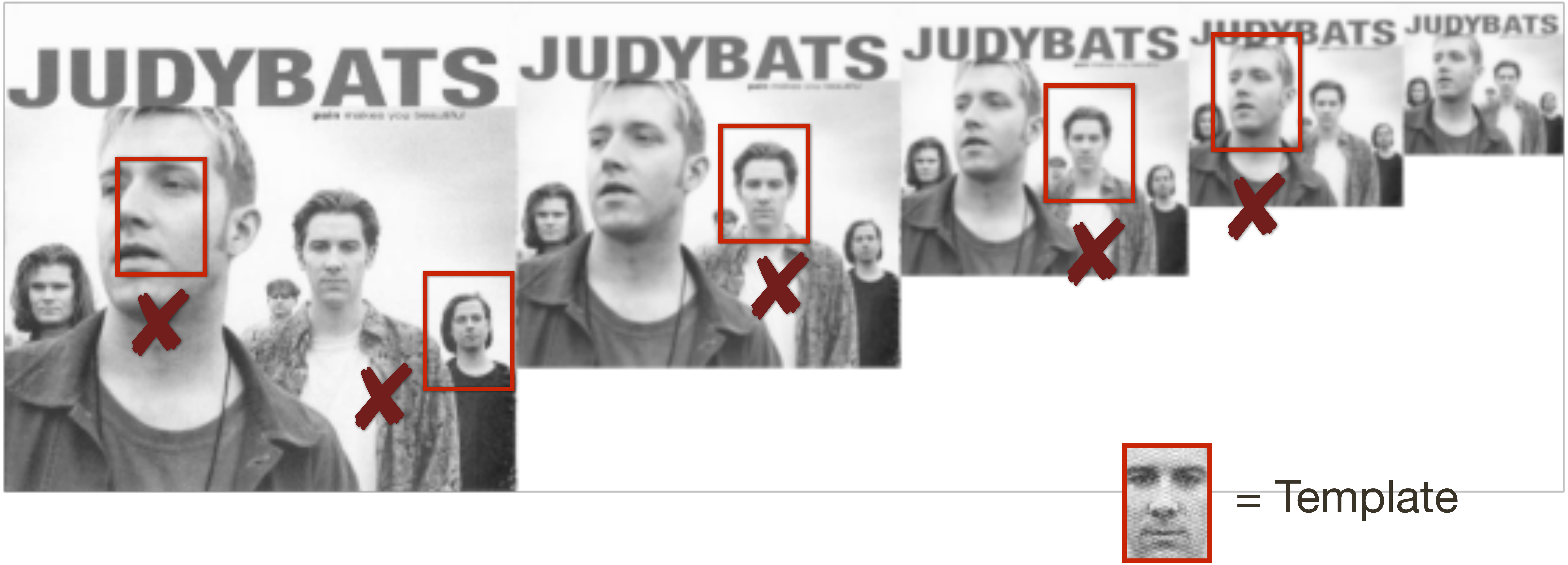
Correlation with a **fixed-sized template** only detects faces at **specific scales**



= Template

Multi-Scale Template Matching

Correlation with a **fixed-sized template** only detects faces at **specific scales**



Course Review: Edge and Corners

Estimating the **image gradient**

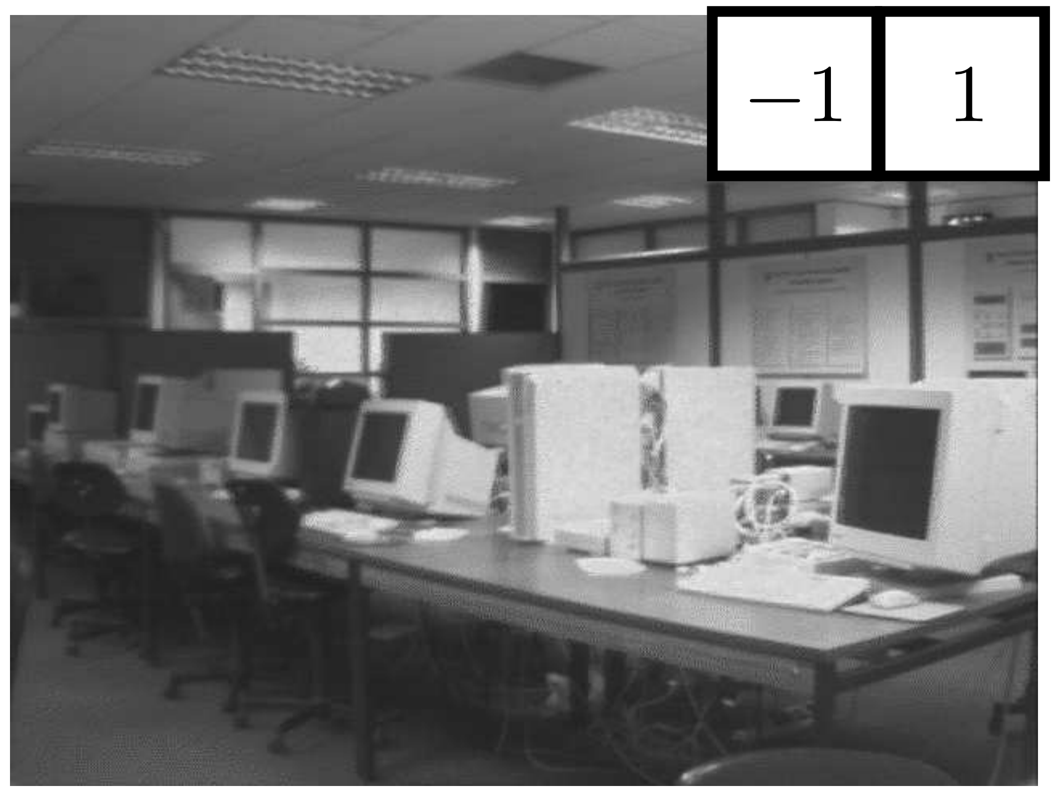
Canny edge detection

Marr/Hildreth edge detection

Boundary detection

Harris corner detection

Estimating **Derivatives**



"**forward** difference" implemented as

correlation

-1	1

from **left**

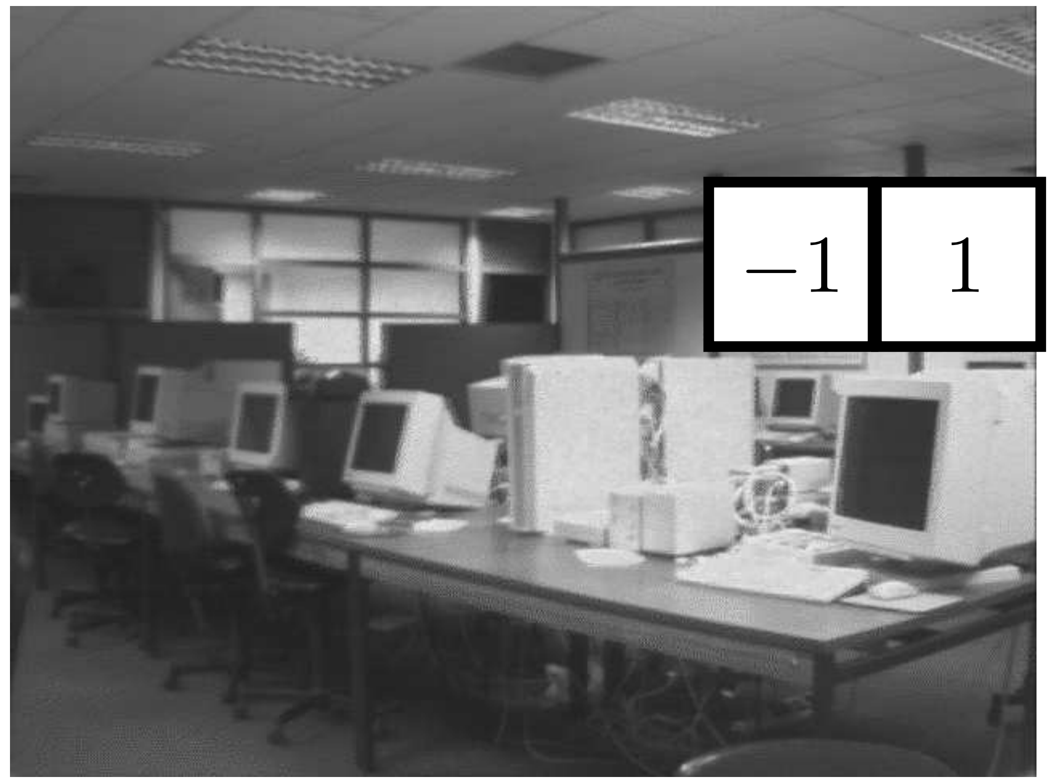
"**backward** difference" implemented as

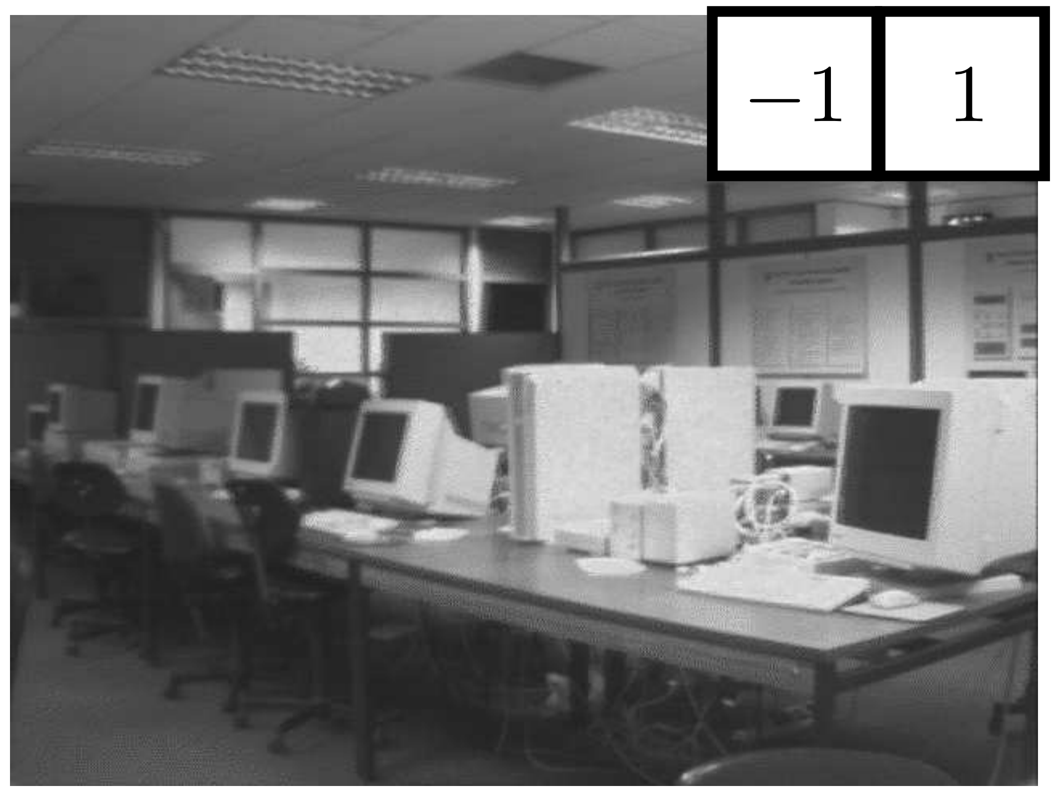
correlation

-1	1

from **right**

Estimating **Derivatives**





"**forward** difference" implemented as

correlation

-1	1

from **left**

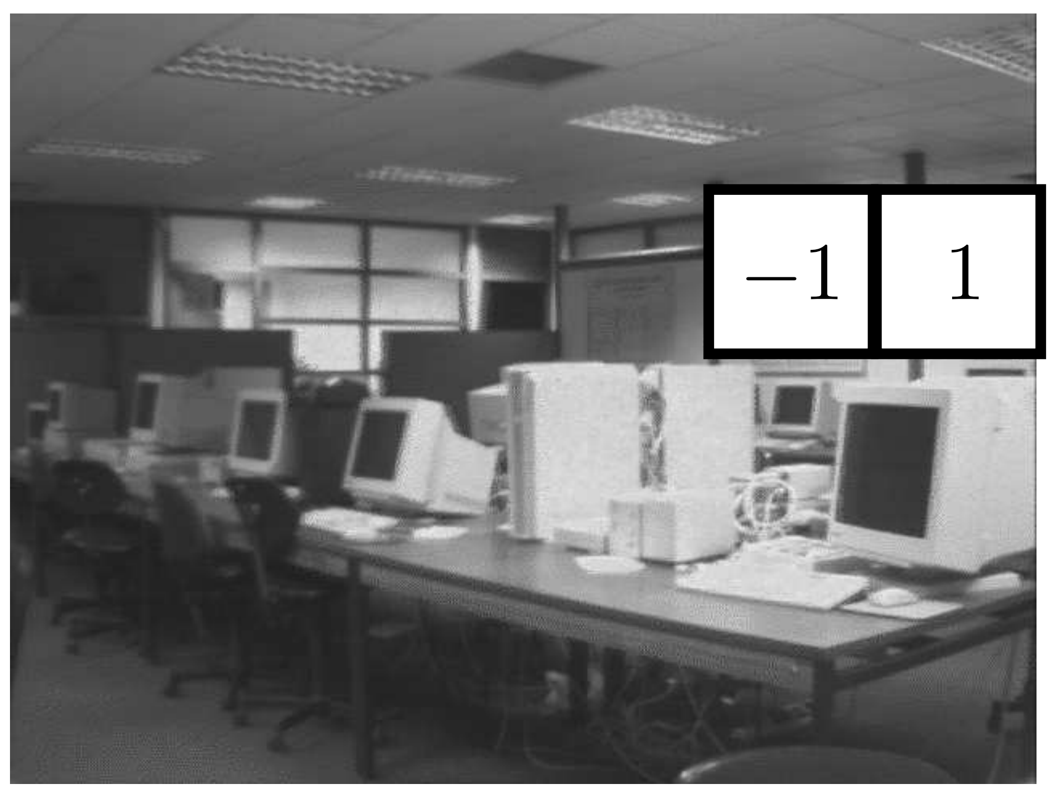
"**backward** difference" implemented as

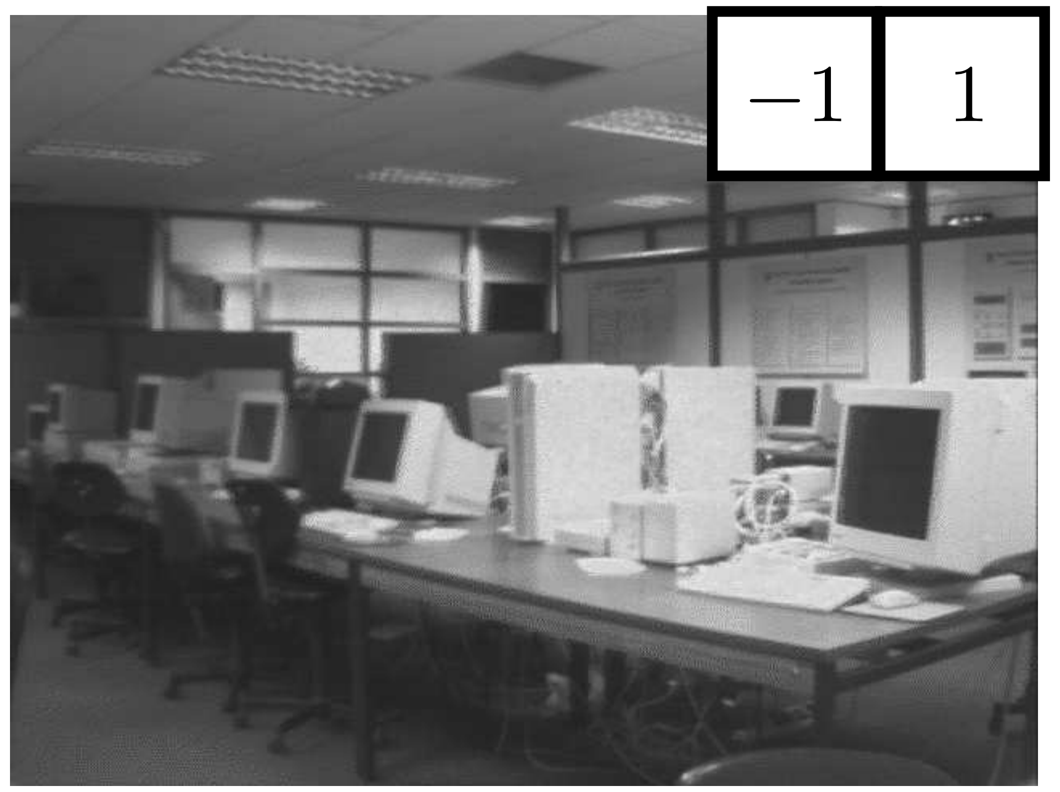
correlation

-1	1

from **right**

Estimating **Derivatives**





"**forward** difference" implemented as

correlation

-1	1

from **left**

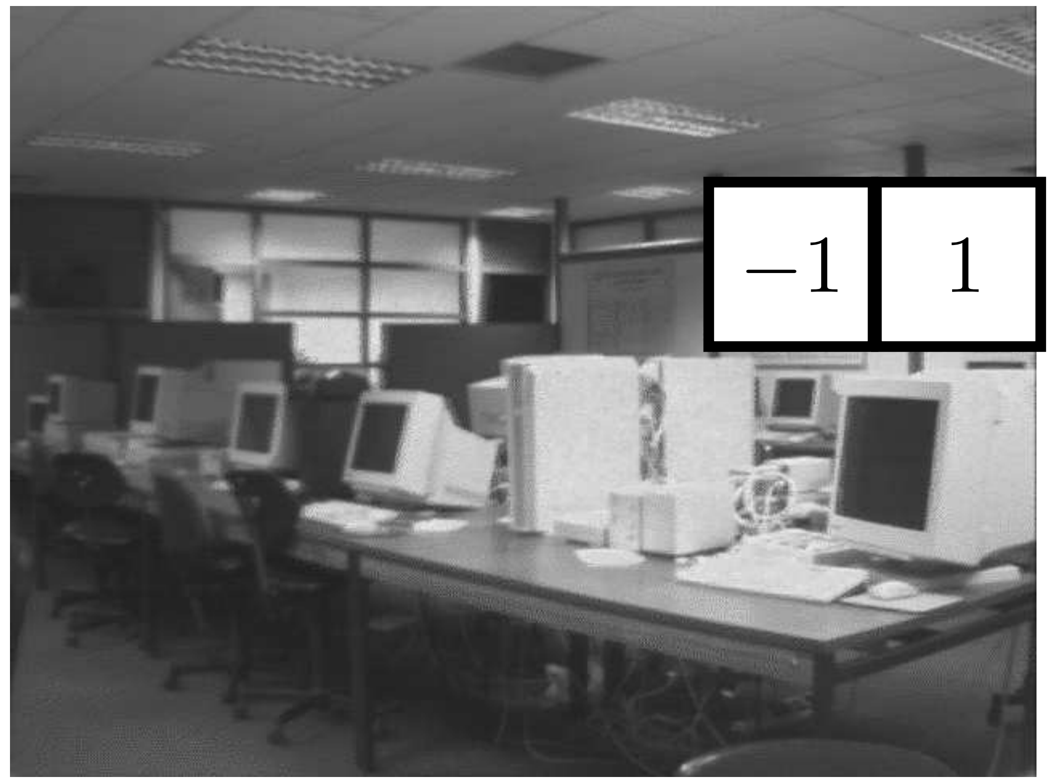
"**backward** difference" implemented as

correlation

-1	1

from **right**

Estimating **Derivatives**



"**forward** difference" implemented as

correlation

-1	1

from **left**

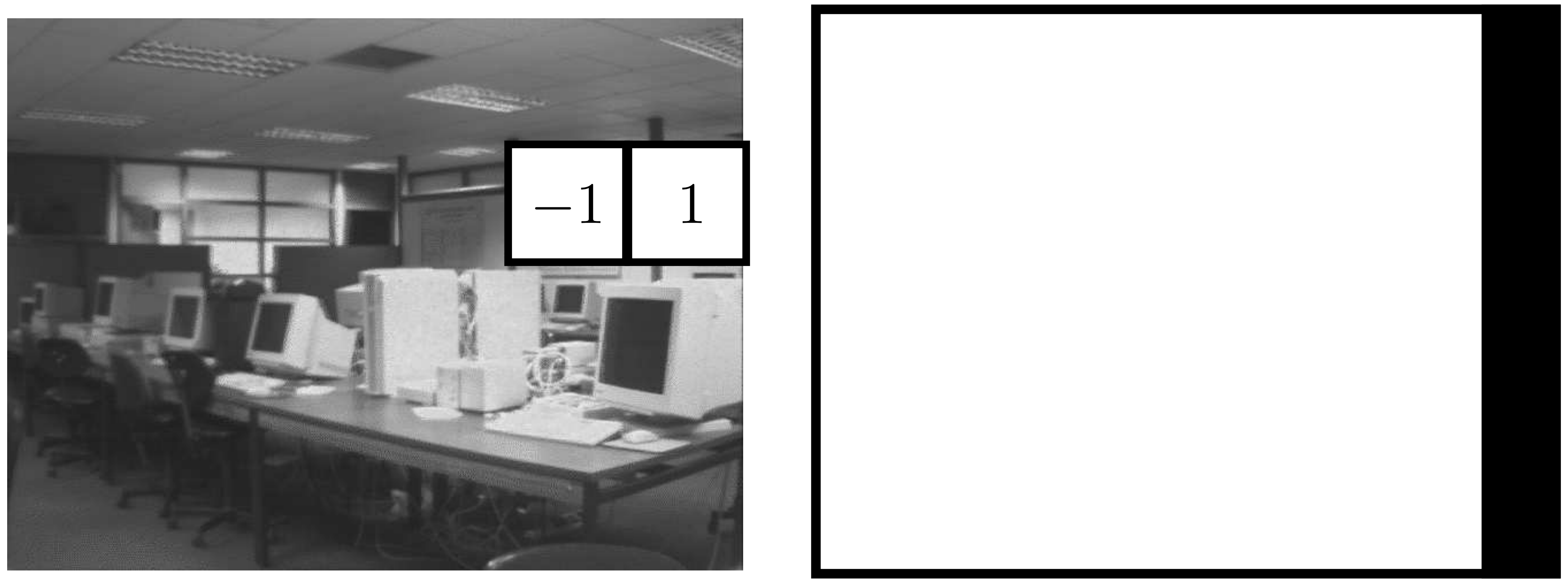
"**backward** difference" implemented as

correlation

-1	1

from **right**

Estimating **Derivatives**



"**forward** difference" implemented as

correlation

-1	1

from **left**

"**backward** difference" implemented as

correlation

-1	1

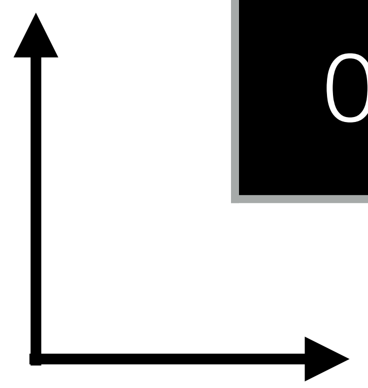
from **right**

A Sort **Exercise**: Derivative in X Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

1	1	0.6	0.3	0	0
1	1	0.6	0.3	0	0
0	0	0	0	0	0
0	0	0	0	0	0



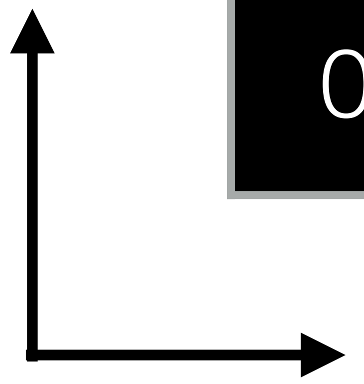
0	-0.4	-0.3	-0.3	0	
0	-0.4	-0.3	-0.3	0	
0	0	0	0	0	
0	0	0	0	0	

A Sort **Exercise**: Derivative in Y Direction

Use the “first forward difference" to compute the image derivatives in X and Y directions.

(Compute two arrays, one of $\frac{\partial f}{\partial x}$ values and one of $\frac{\partial f}{\partial y}$ values.)

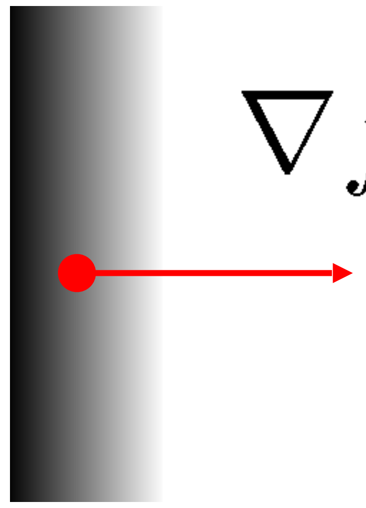
1	1	0.6	0.3	0	0
1	1	0.6	0.3	0	0
0	0	0	0	0	0
0	0	0	0	0	0



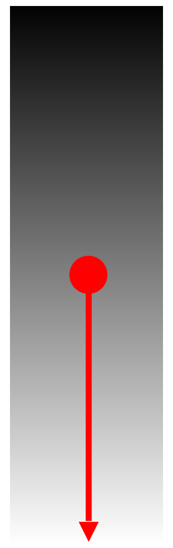
0	0	0	0	0	0
1	1	0.6	0.3	0	0
0	0	0	0	0	0

Image **Gradient**

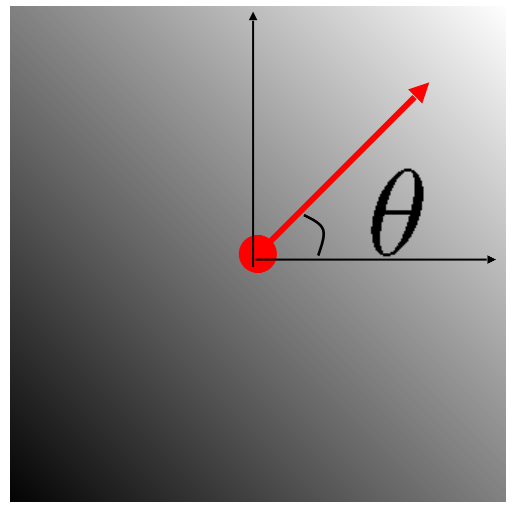
The gradient of an image: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of most rapid **increase of intensity**:

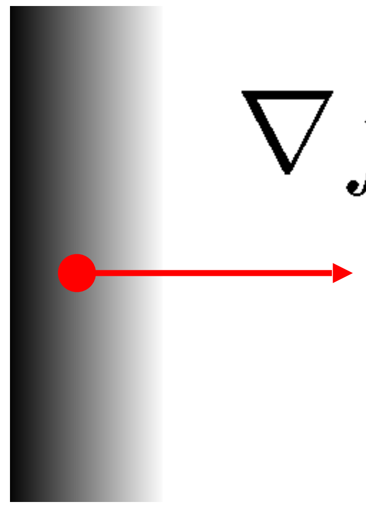
The **gradient direction** is given by:

(how is this related to the direction of the edge?)

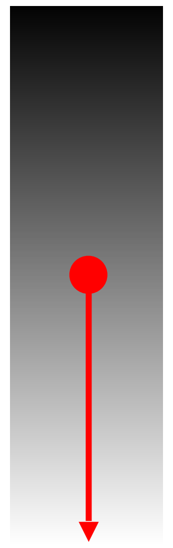
The edge strength is given by the **gradient magnitude**:

Image **Gradient**

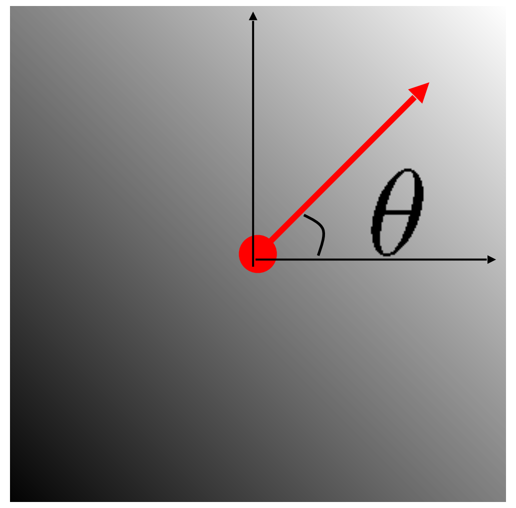
The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$



$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$



$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$



$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

The gradient points in the direction of most rapid **increase of intensity**:

The **gradient direction** is given by: $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$

(how is this related to the direction of the edge?)

The edge strength is given by the **gradient magnitude**: $\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

Marr / Hildreth **Laplacian of Gaussian**

A "**zero crossings** of a second derivative operator" approach

Steps:

1. Gaussian for smoothing

2. Laplacian (∇^2) for differentiation where

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

3. Locate zero-crossings in the Laplacian of the Gaussian ($\nabla^2 G$) where

$$\nabla^2 G(x,y) = \frac{-1}{2\pi\sigma^4}\left[2 - \frac{x^2+y^2}{\sigma^2}\right]\exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

Canny Edge Detector

Steps:

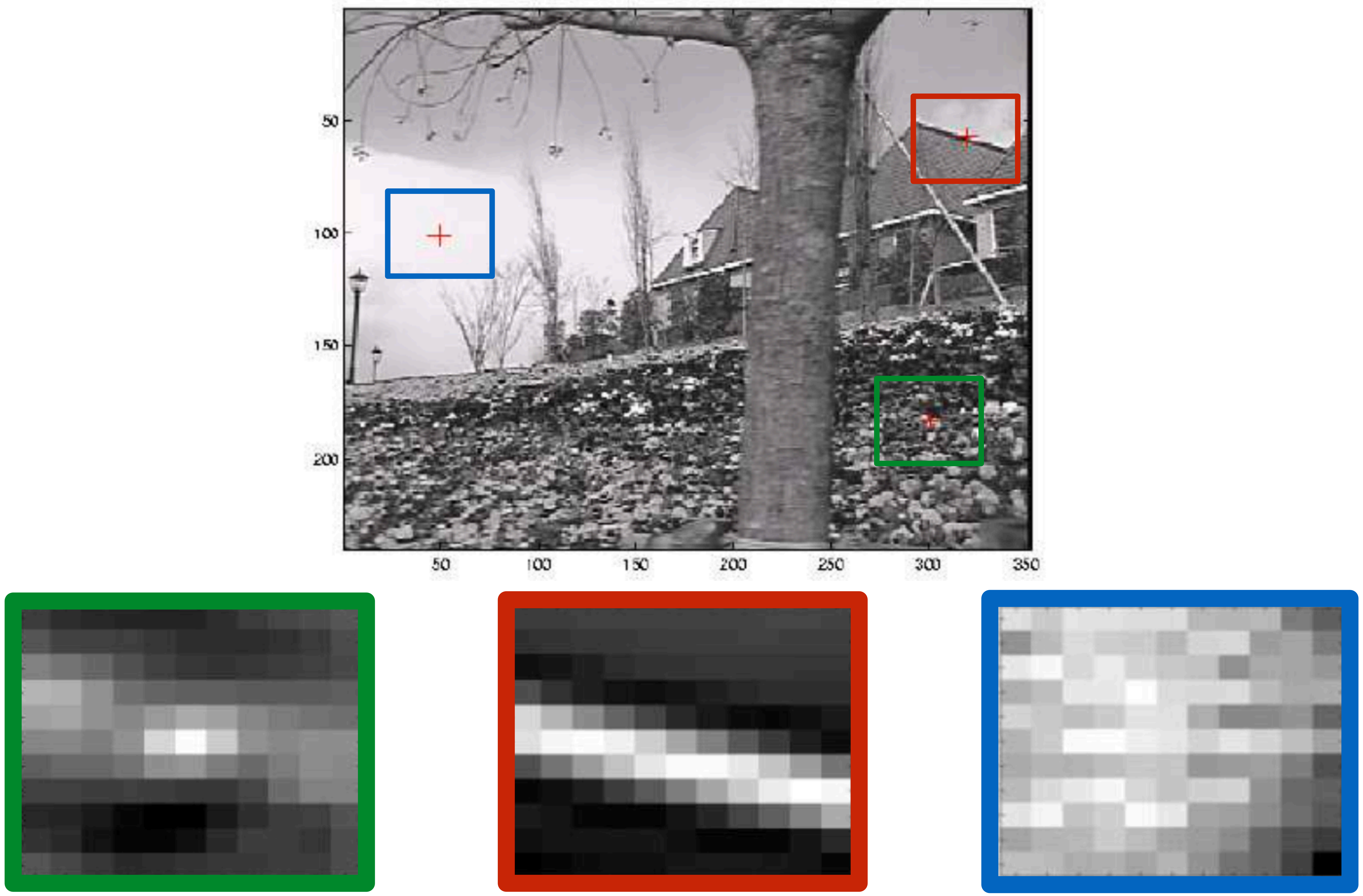
1. Apply **directional derivatives** of Gaussian

2. Compute **gradient magnitude** and **gradient direction**

3. **Non-maximum** suppression
   — thin multi-pixel wide “ridges” down to single pixel width

4. **Linking** and thresholding
   — Low, high edge-strength thresholds
   — Accept all edges over low threshold that are connected to edge over high threshold

Autocorrelation



Szeliski, Figure 4.5

Harris Corner Detection Review

— Filter image with **Gaussian**

— Compute magnitude of the x and y **gradients** at each pixel

— Construct C in a window around each pixel

— Harris uses a **Gaussian window**

— Solve for product of the λ's

— If λ's both are big (product reaches local maximum above threshold) then we have a corner

— Harris also checks that ratio of λs is not too high

Harris & Stephens (1988)

$$\det(C) - \kappa \operatorname{trace}^2(C)$$

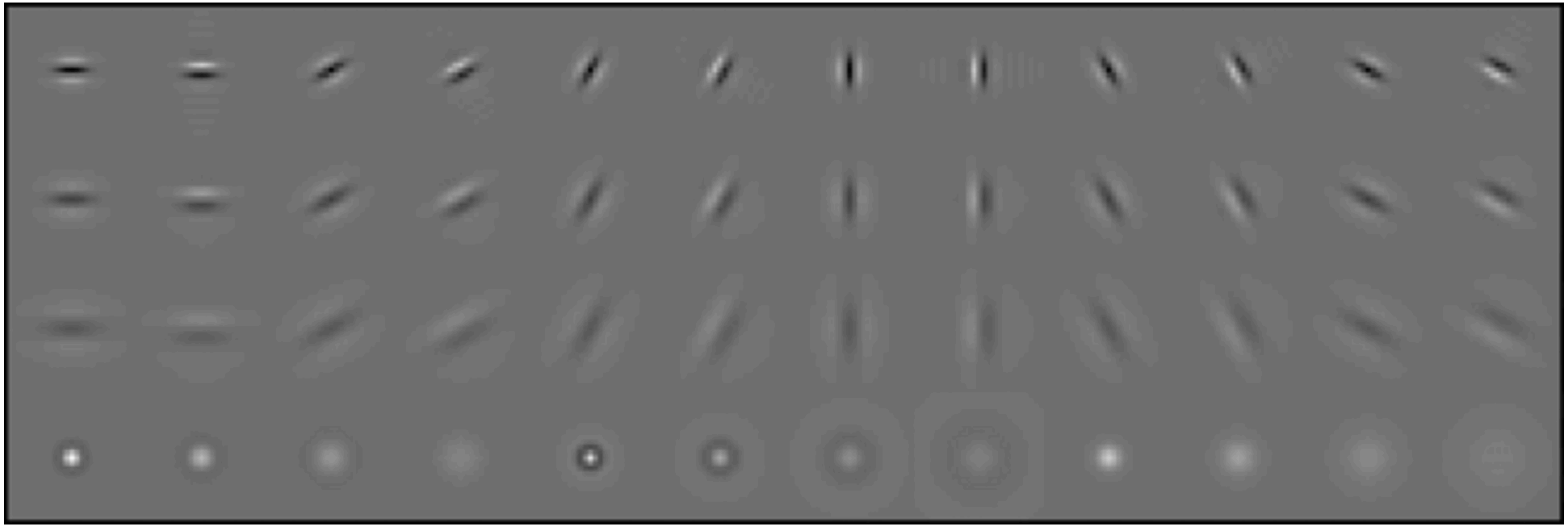
Course Review: Texture

Texture representation

Laplacian pyramid, oriented pyramid

Texture **synthesis** (Efros and Leung paper)

Texture **Representation**

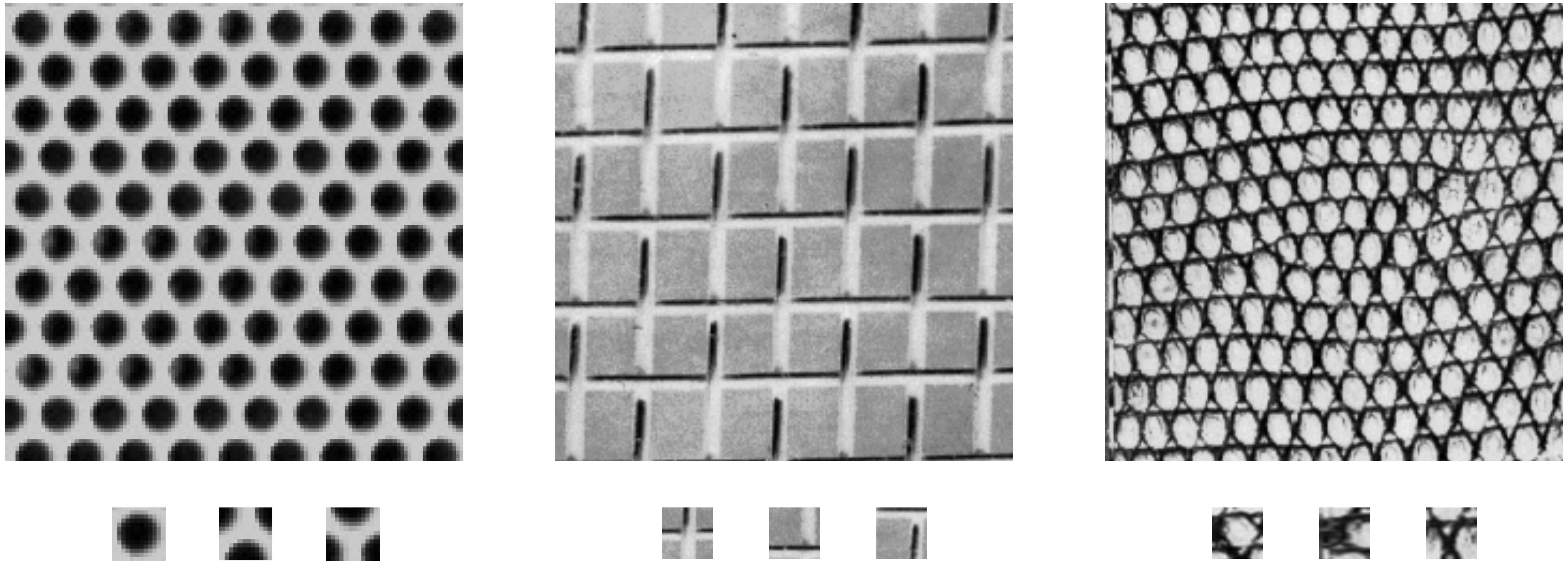


Result: 48-channel “image”

Figure Credit: Leung and Malik, 2001

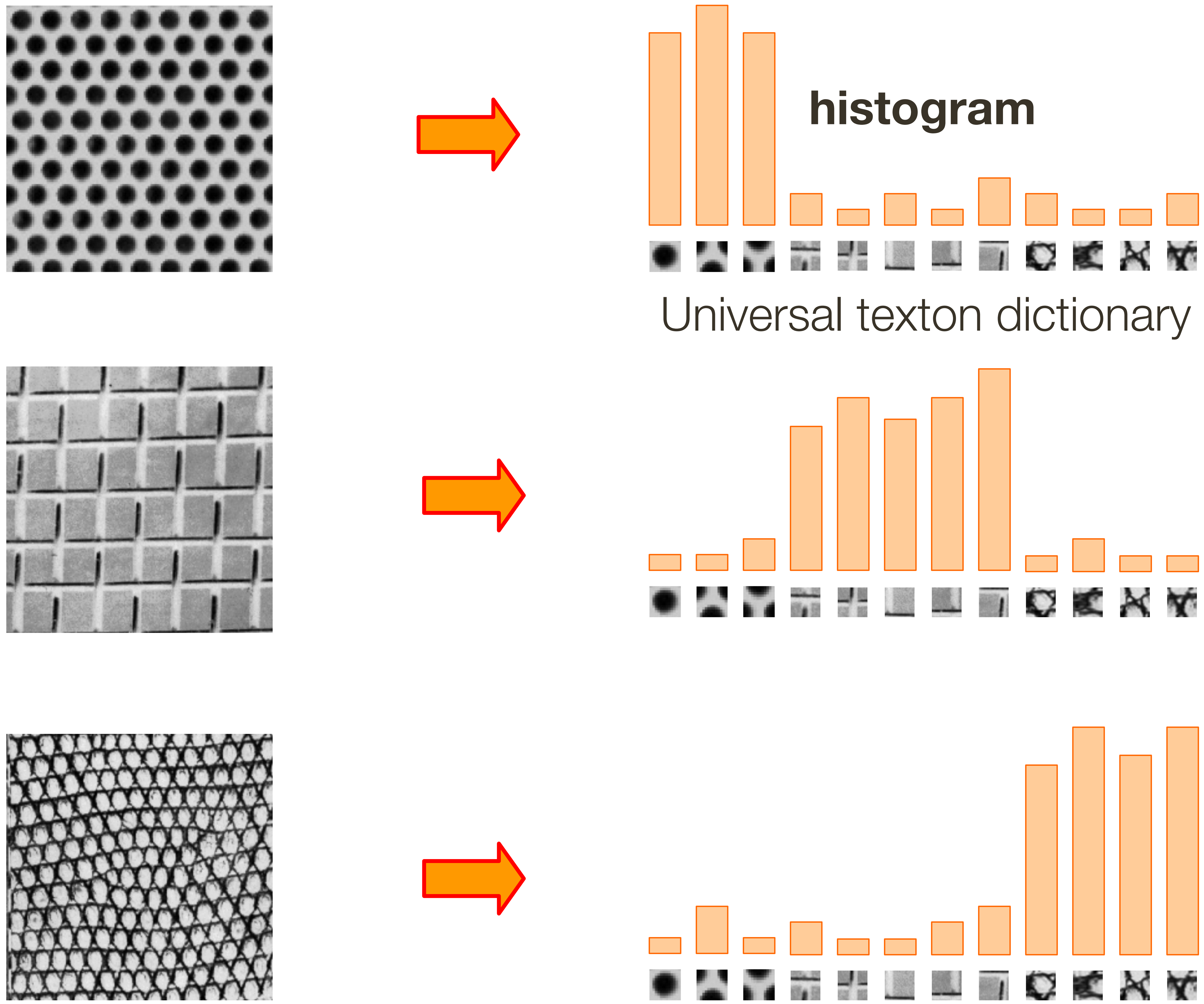
Texture representation and recognition

- Texture is characterized by the repetition of basic elements or **textons**
- For stochastic textures, it is the **identity of the textons**, not their spatial arrangement, that matters

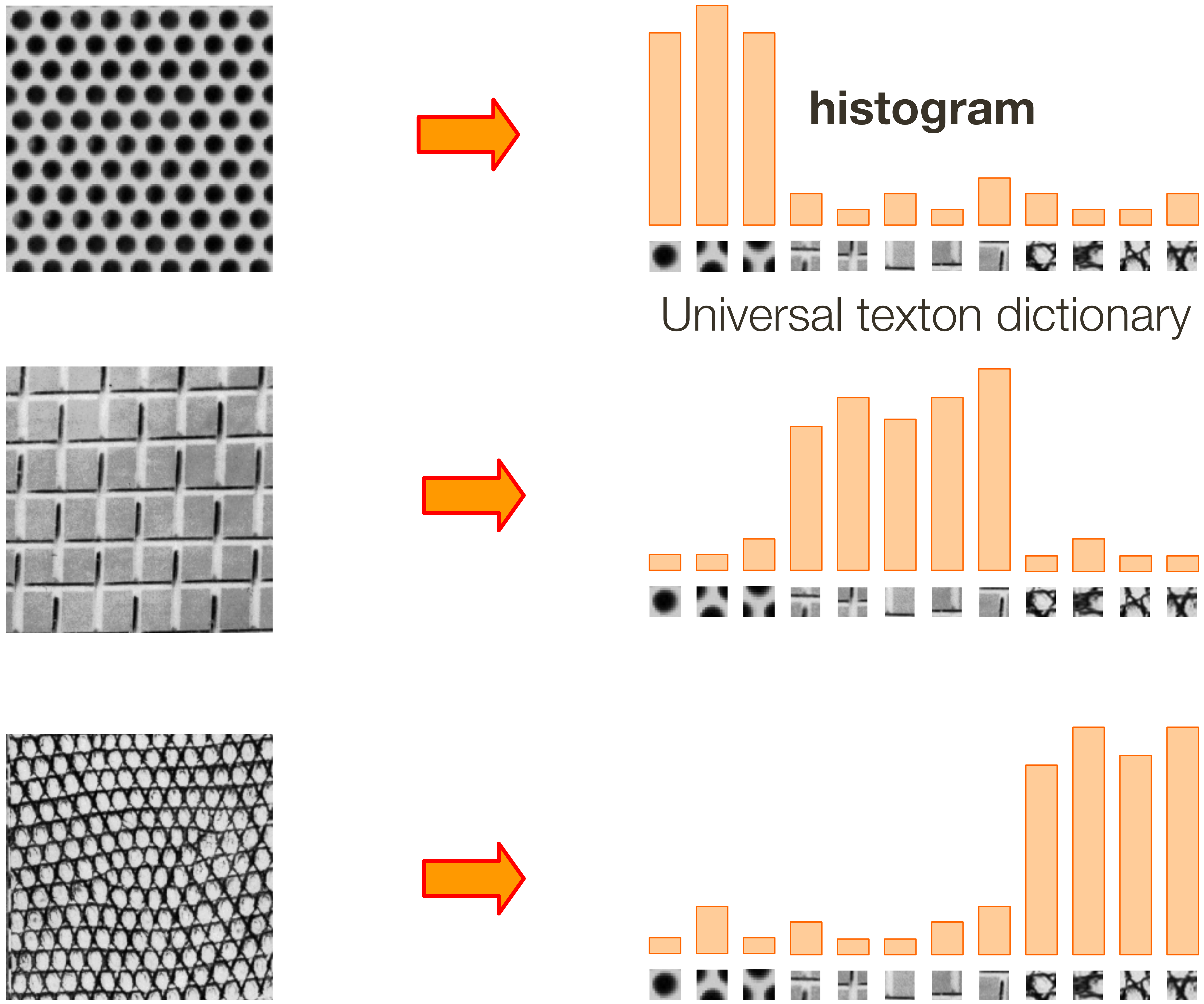


Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

Texture representation and recognition



Texture representation and recognition



Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

Course Review: Local Invariant Features

Keypoint detection using Difference of Gaussian pyramid

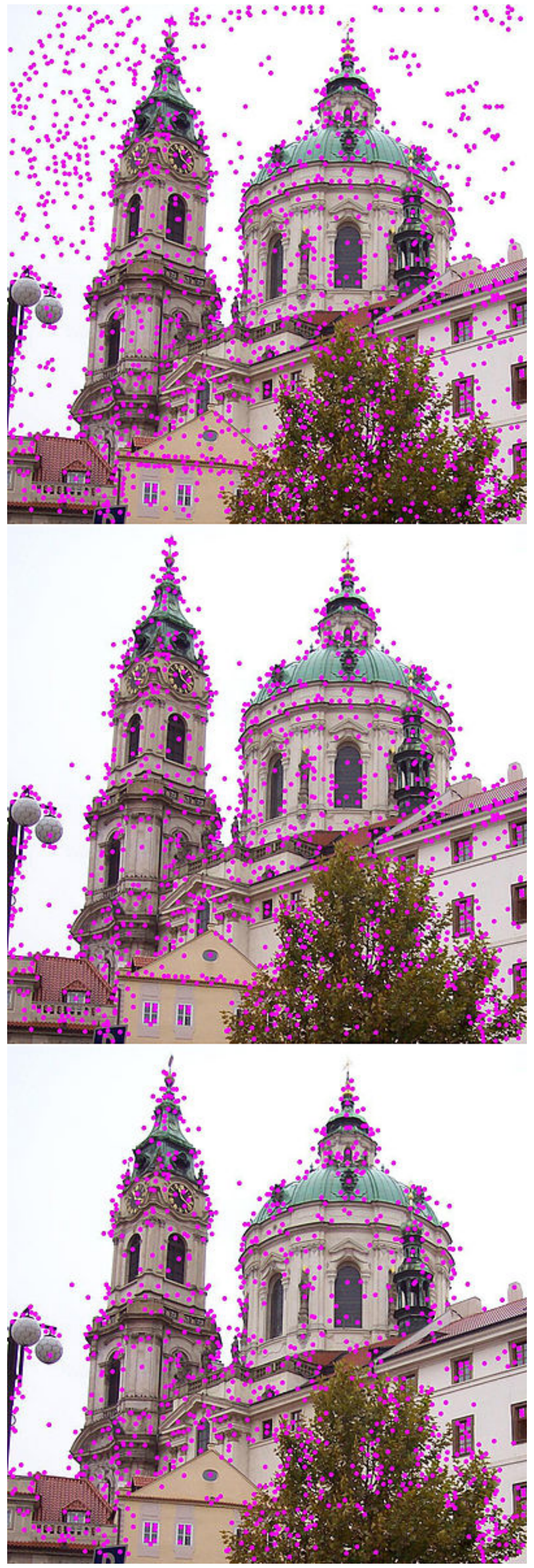
Keypoint **orientation assignment**

Keypoint **descriptor**

Matching with nearest and second-nearest neighbors

SIFT and object recognition

Scale Invariant Feature Transform (**SIFT**)

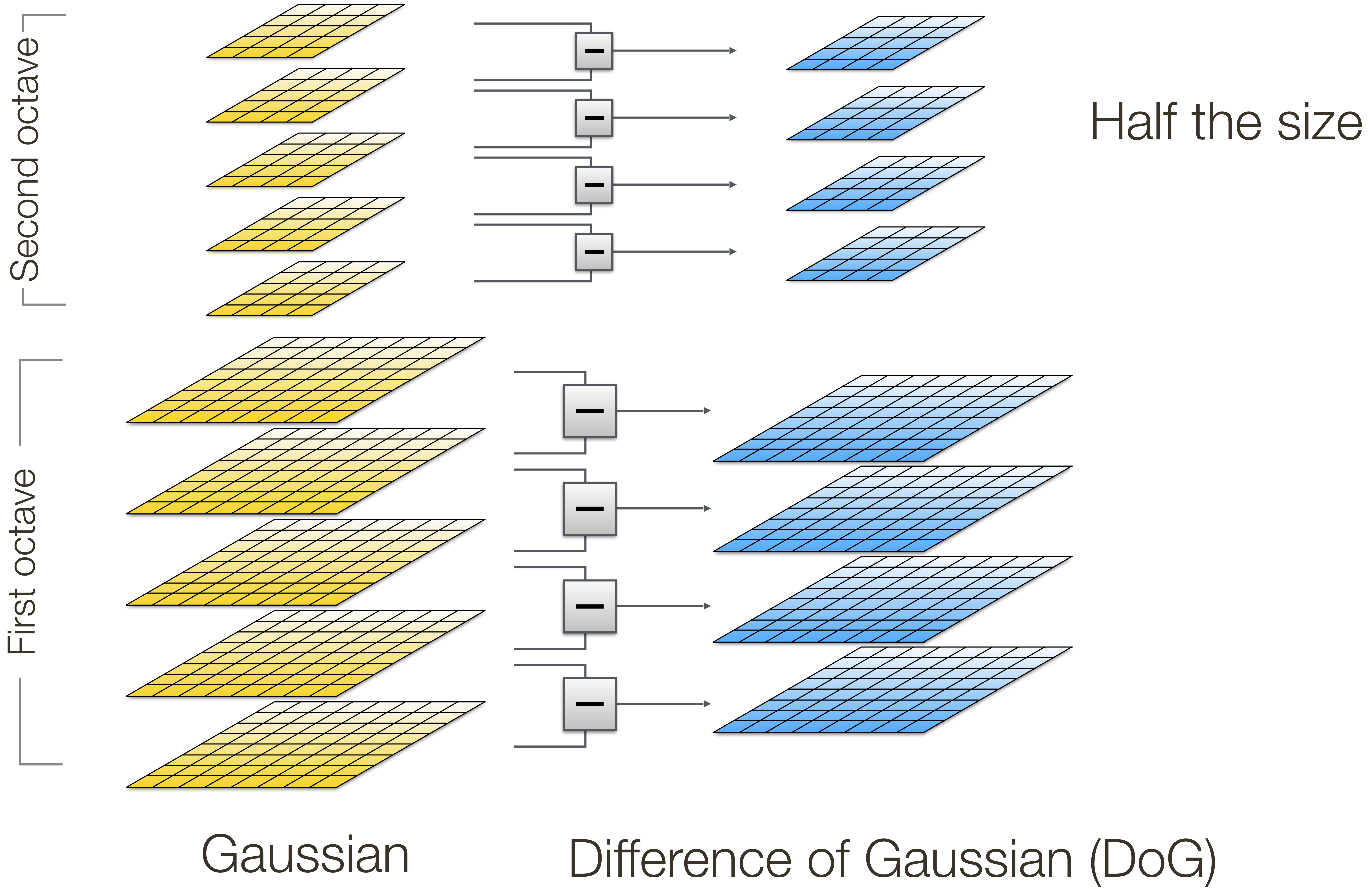


SIFT describes both a **detector** and **descriptor**

1. Multi-scale extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

1. Multi-scale Extrema Detection

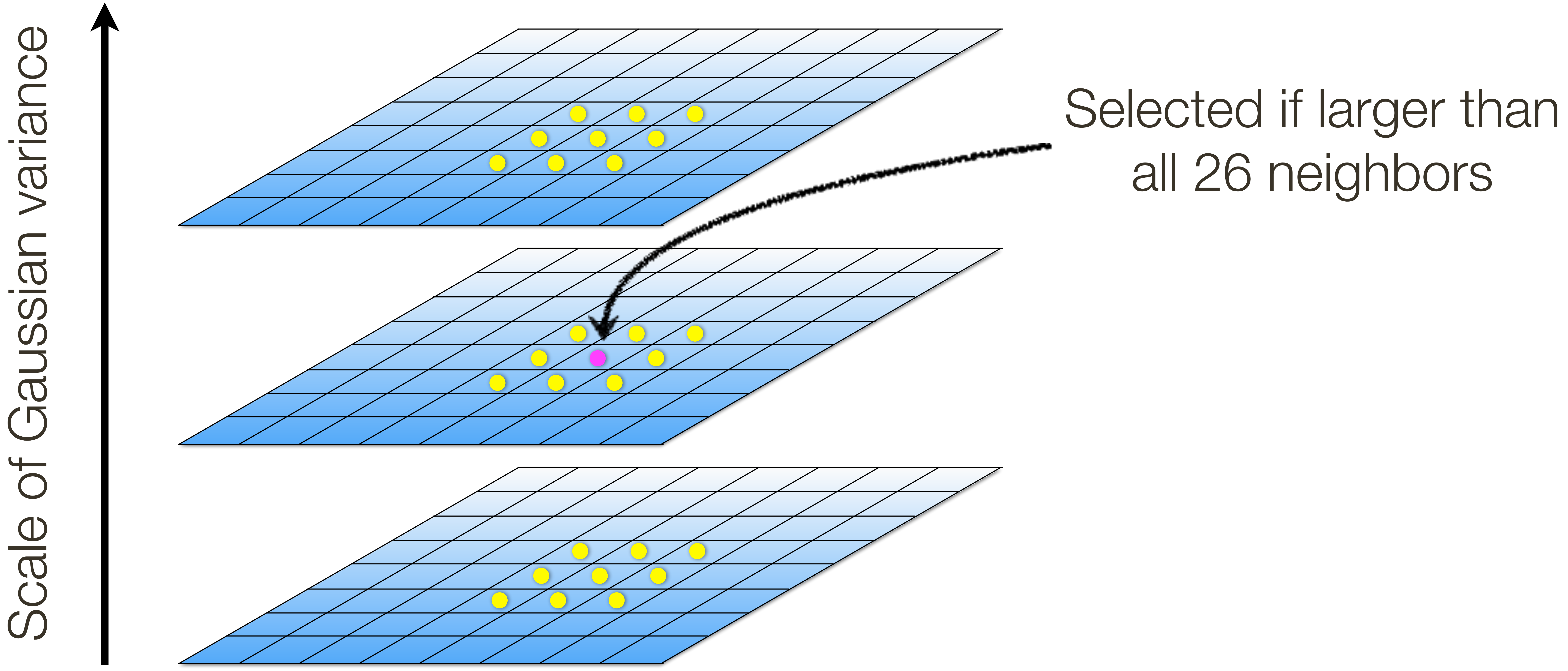


Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

1. Multi-scale Extrema Detection

Detect maxima and minima of Difference of Gaussian in scale space

$$\sigma = 2^{1/s}$$



Difference of Gaussian (DoG)

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

2. Keypoint Localization

— After keypoints are detected, we remove those that have **low contrast** or are **poorly localized** along an edge

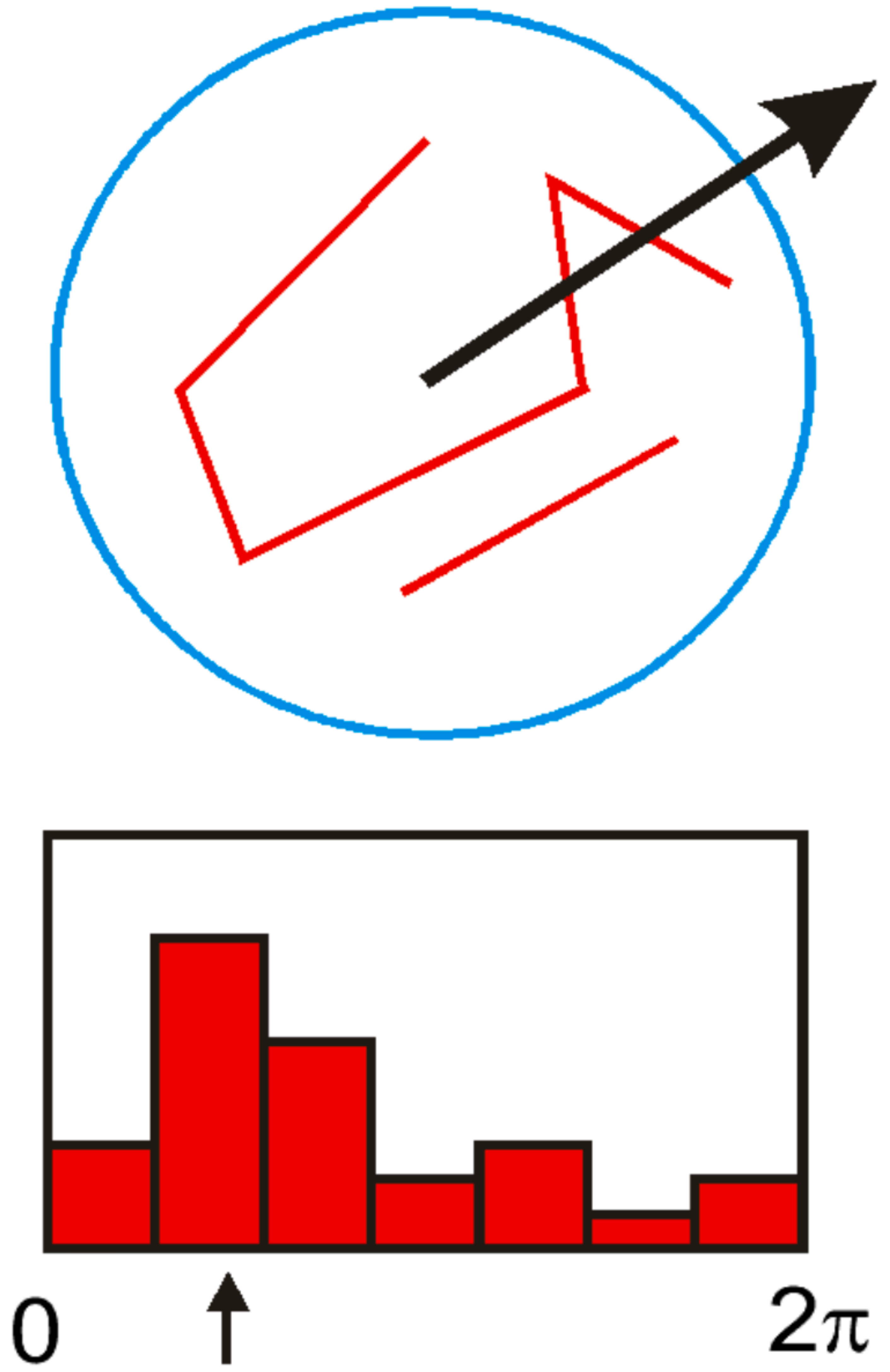
— Lowe suggests computing the ratio of the eigenvalues of **C** (recall Harris corners) and checking if it is greater than a threshold

3. Orientation Assignment

— Create **histogram** of local gradient directions computed at selected scale

— Assign **canonical orientation** at peak of smoothed histogram

— Each key specifies stable 2D coordinates (x , y , scale, orientation)

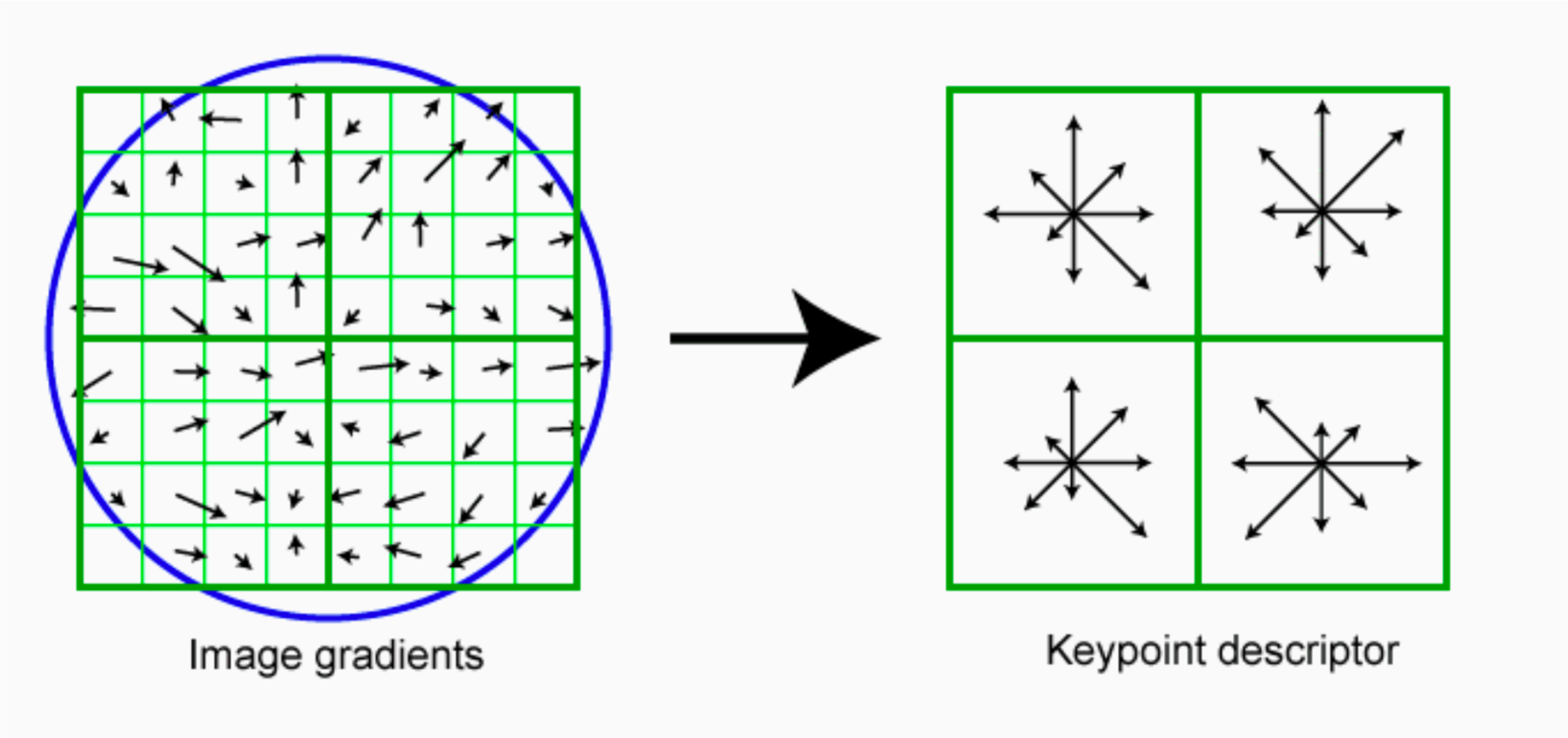


4. SIFT Descriptor

— Thresholded image gradients are sampled over 16 × 16 array of locations in scale space (weighted by a Gaussian with sigma half the size of the window)

— Create array of orientation histograms

— 8 orientations × 4 × 4 histogram array



Alternatives to SIFT

- Histogram of Oriented Gradients (**HoG**) — more detailed, higher dimensional
- **SURF** — faster, lower dimensional

Course Review: Fitting Data to a Model

RANSAC

Hough transform

RANSAC (**RAN**dom **SA**mple **C**onsensus)

1. Randomly choose minimal subset of data points necessary to fit model (a **sample**)

2. Points within some distance threshold, t, of model are a **consensus set**. Size of consensus set is model's **support**

3. Repeat for N samples; model with biggest support is most robust fit
   — Points within distance t of best model are inliers
   — Fit final model to all inliers

Slide Credit: Christopher Rasmussen

RANSAC: How many samples?

Let ω be the fraction of inliers (i.e., points on line)

Let n be the number of points needed to define hypothesis
($n = 2$ for a line in the plane)

Suppose k samples are chosen

The probability that a single sample of n points is correct (all inliers) is

$$\omega^n$$

The probability that all k samples fail is

$$(1 - \omega^n)^k$$

Choose k large enough (to keep this below a target failure rate)

RANSAC: *k* Samples Chosen (p = 0.99)

Sample size	Proportion of outliers						
n	**5%**	**10%**	**20%**	**25%**	**30%**	**40%**	**50%**
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Figure Credit: Hartley & Zisserman

Discussion of RANSAC

Advantages:

— General method suited for a wide range of model fitting problems

— Easy to implement and easy to calculate its failure rate

Disadvantages:

— Only handles a moderate percentage of outliers without cost blowing up

— Many real problems have high rate of outliers (but sometimes selective choice of random subsets can help)

The Hough transform can handle high percentage of outliers

Hough Transform

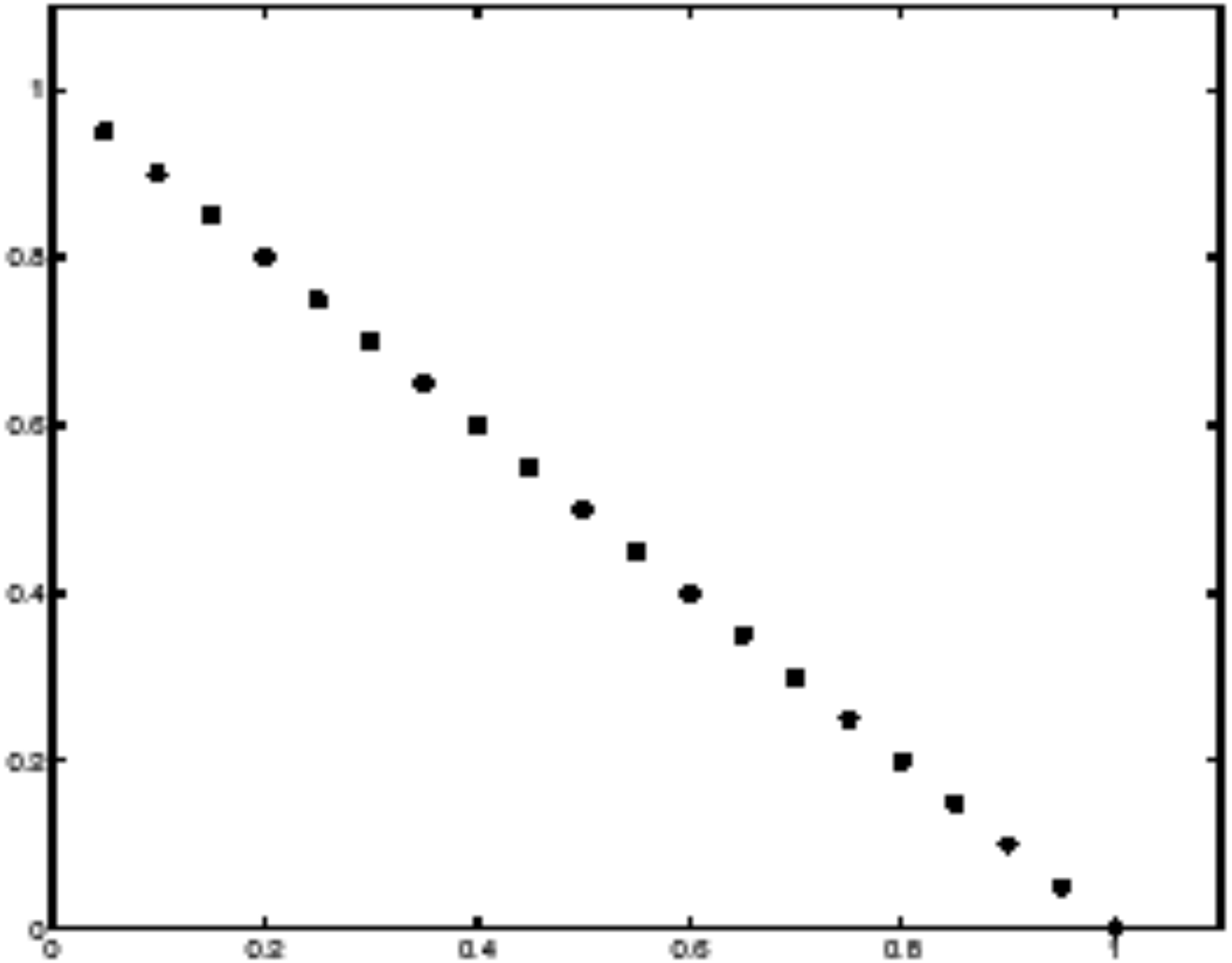
Idea of **Hough transform**:

— For each token vote for all models to which the token could belong

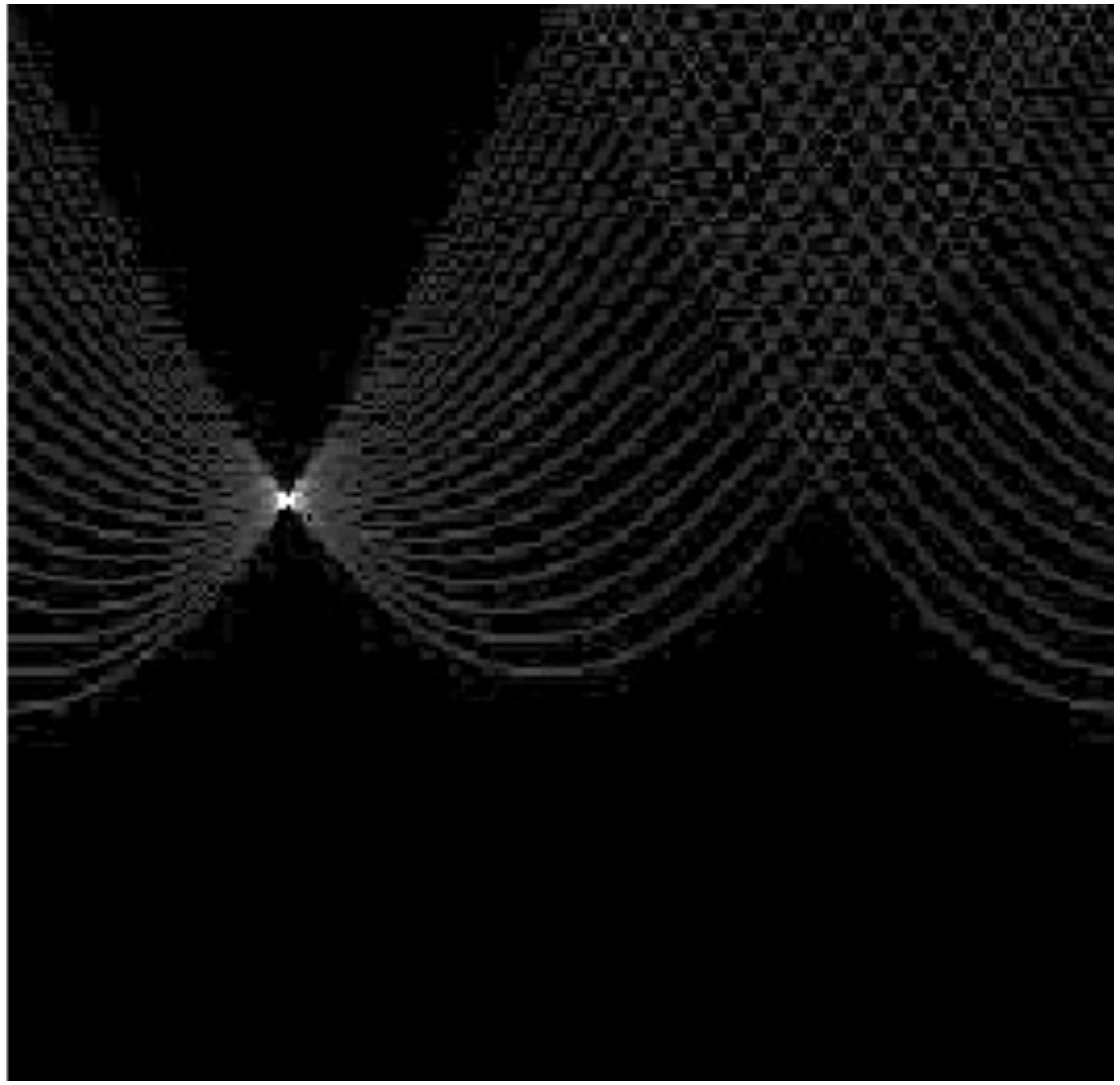
— Return models that get many votes

Example: For each point, vote for all lines that could pass through it; the true lines will pass through many points and so receive many votes

Example: Clean Data



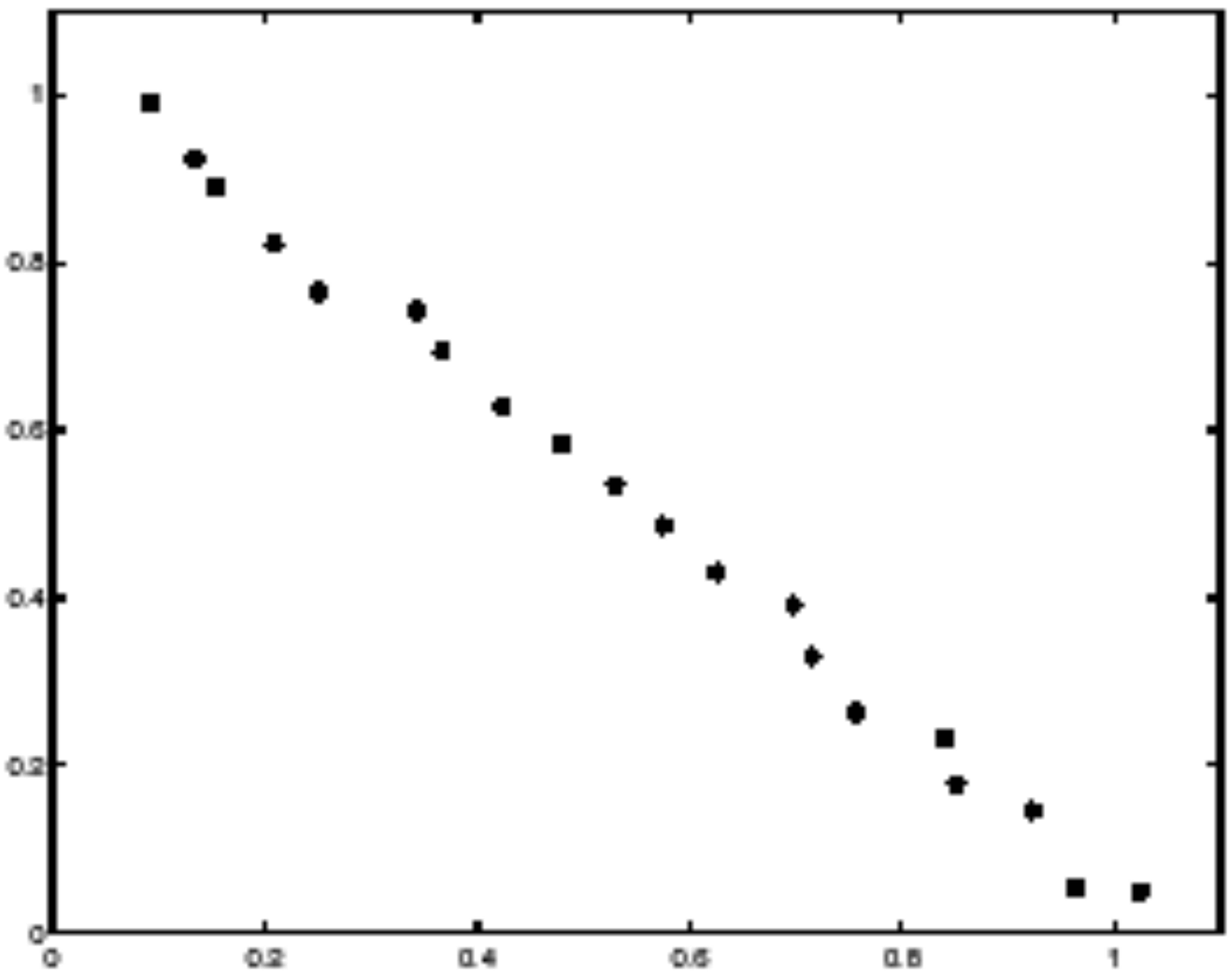
Tokens



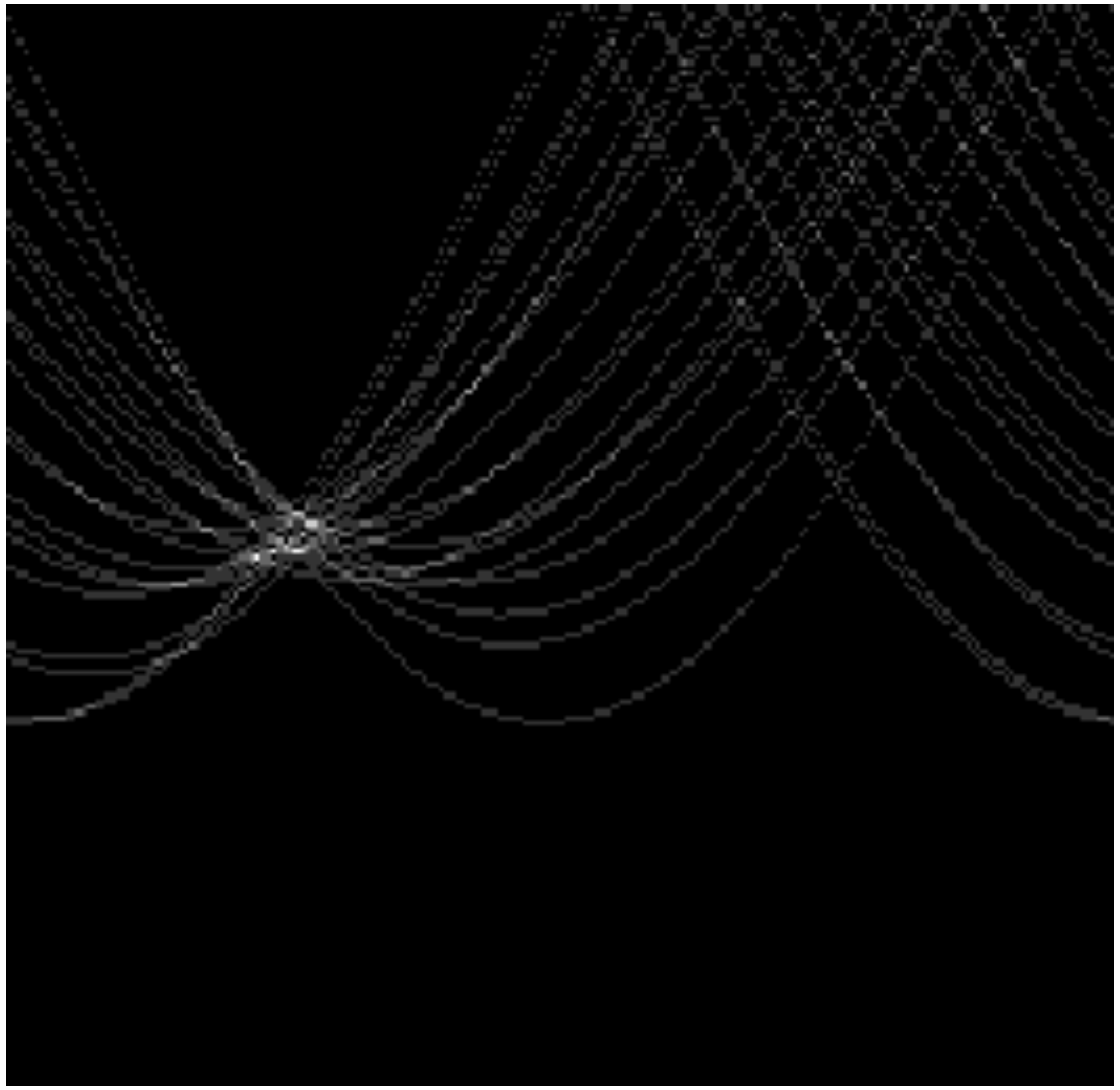
Votes
Horizontal axis is θ
Vertical Axis is r

Forsyth & Ponce (2nd ed.) Figure 10.1 (Top)

Example: Some Noise



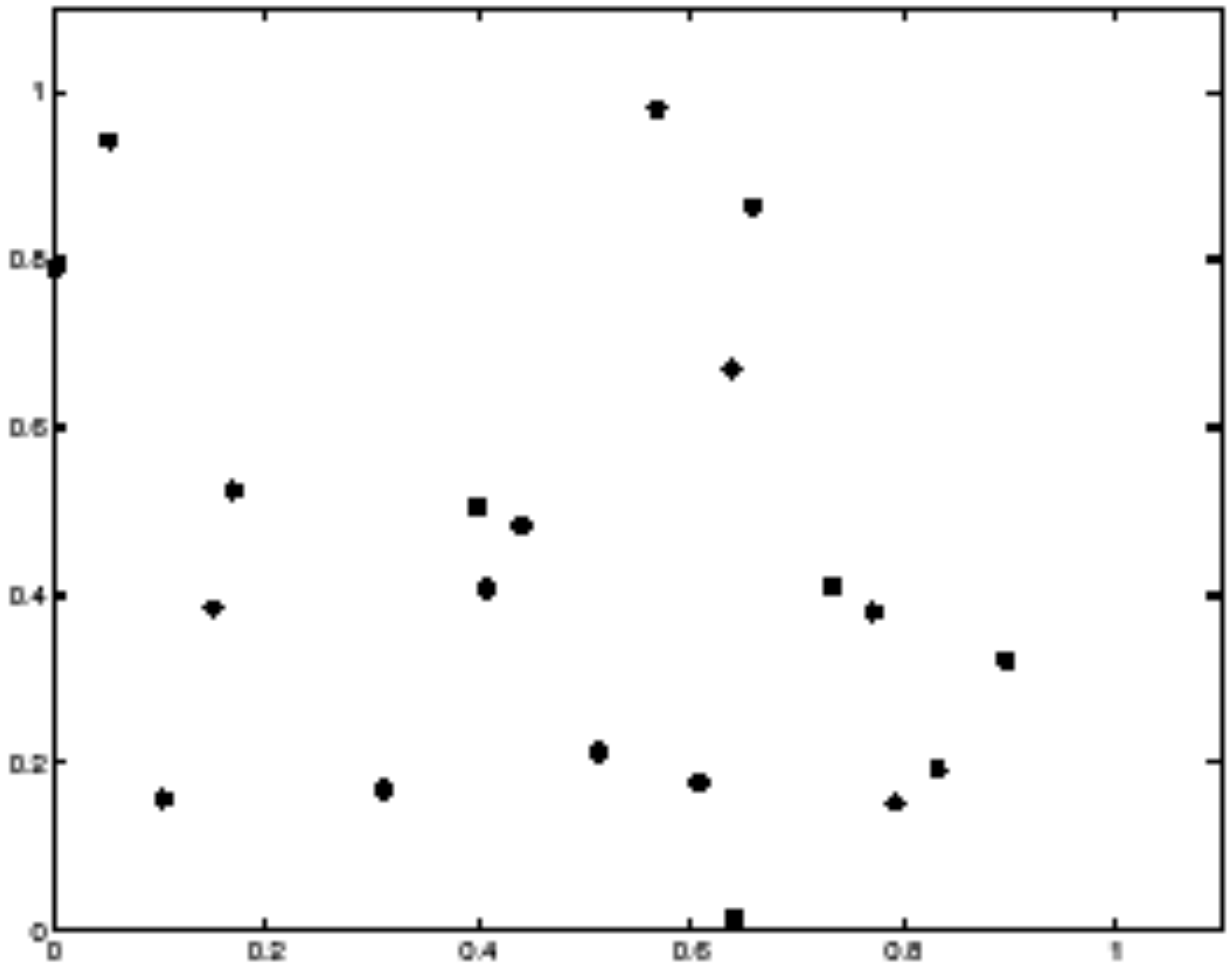
Tokens



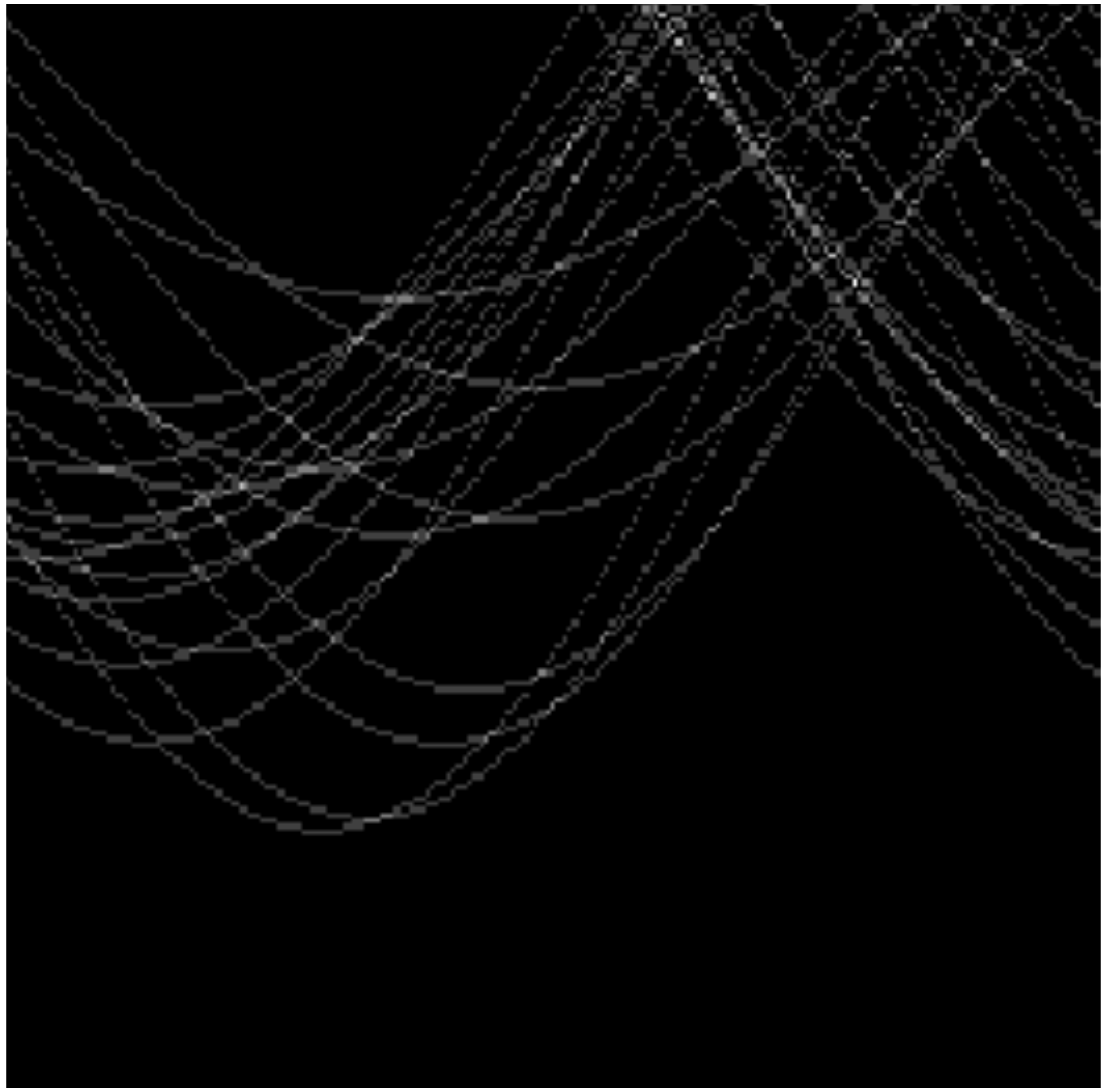
Votes
Horizontal axis is θ
Vertical Axis is r

Forsyth & Ponce (2nd ed.) Figure 10.1 (Bottom)

Example: Too Much Noise



Tokens



Votes

Horizontal axis is θ

Vertical Axis is r

Forsyth & Ponce (2nd ed.) Figure 10.2

Course Review: Stereo

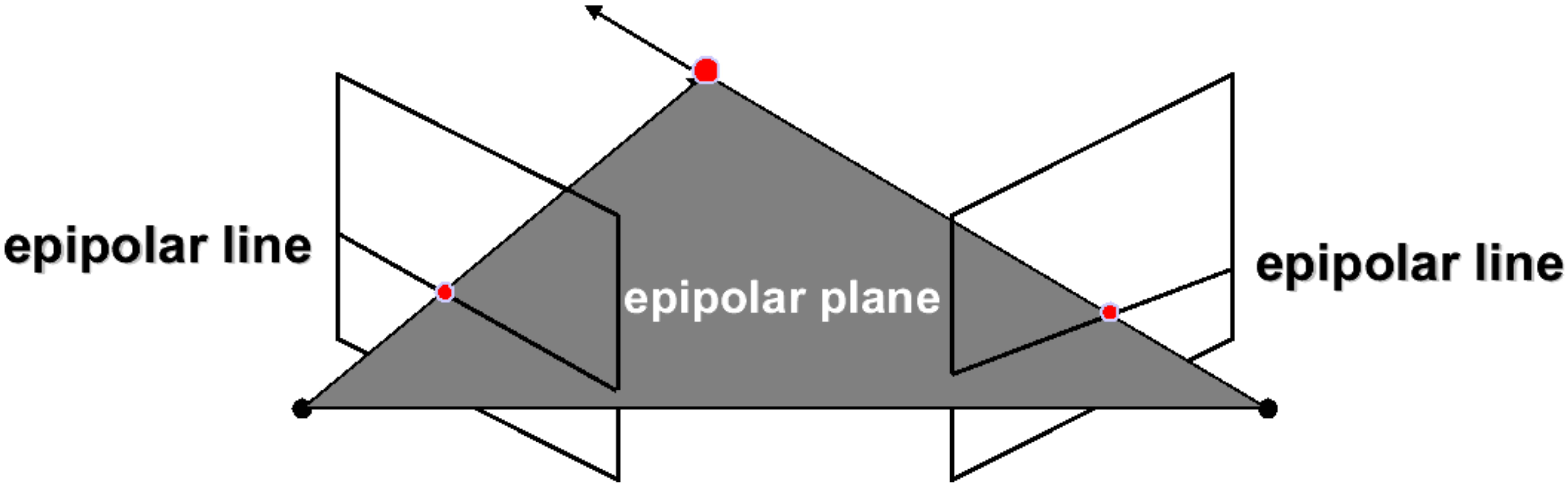
Epipolar constraint

Rectified images

Computing **correspondences**

Ordering constraint

The **Epipolar** Constraint



Matching points lie along corresponding epipolar lines

Reduces correspondence problem to 1D search along conjugate epipolar lines

Greatly reduces cost and ambiguity of matching

Slide credit: Steve Seitz

Simplest Case: **Rectified** Images

Image planes of cameras are **parallel**

Focal **points** are at same height

Focal **lengths** same

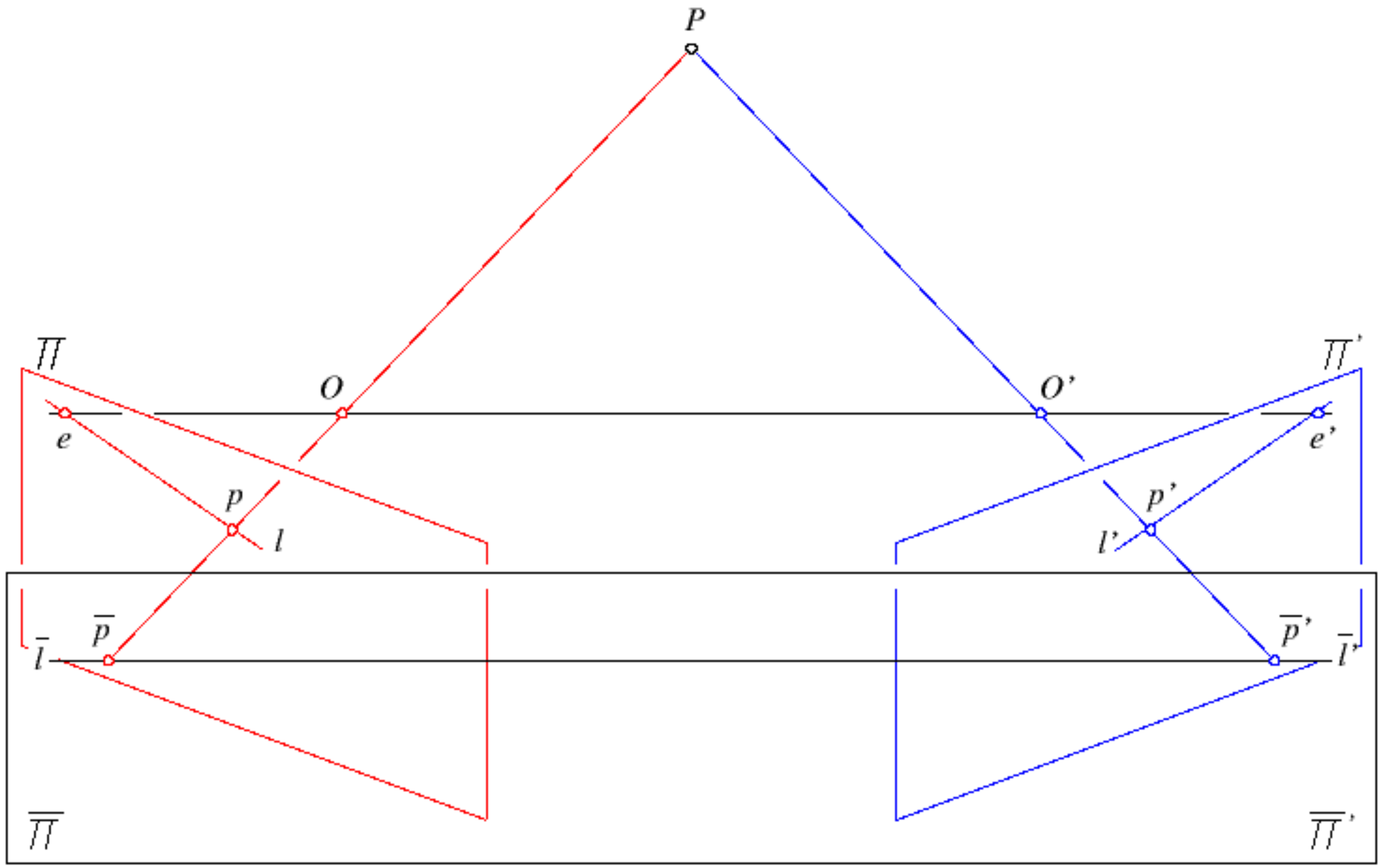
Then, **epipolar lines** fall along the **horizontal scan lines** of the images

We assume images have been **rectified** so that epipolar lines correspond to scan lines

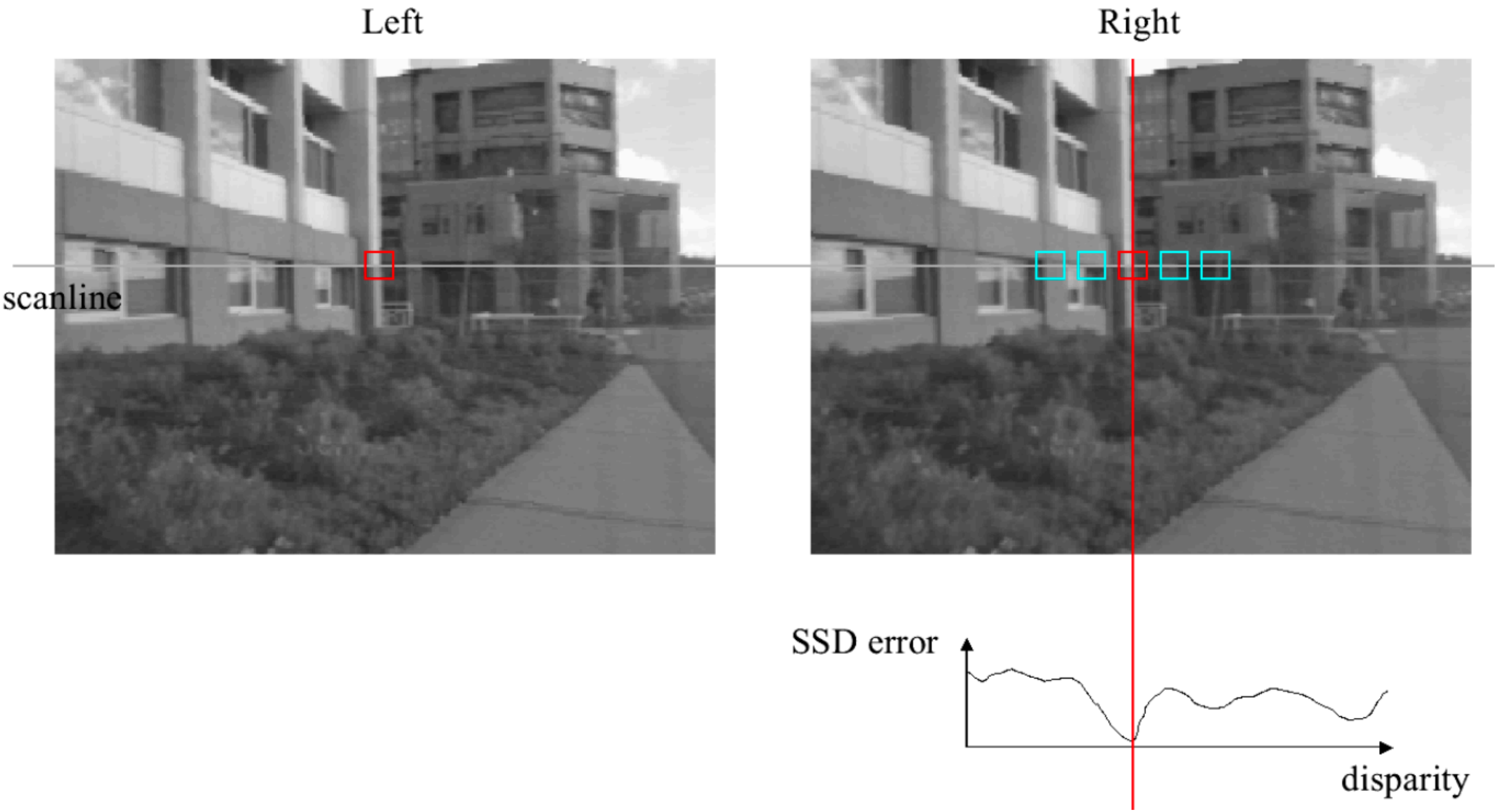
— Simplifies algorithms

— Improves efficiency

Rectified Stereo Pair

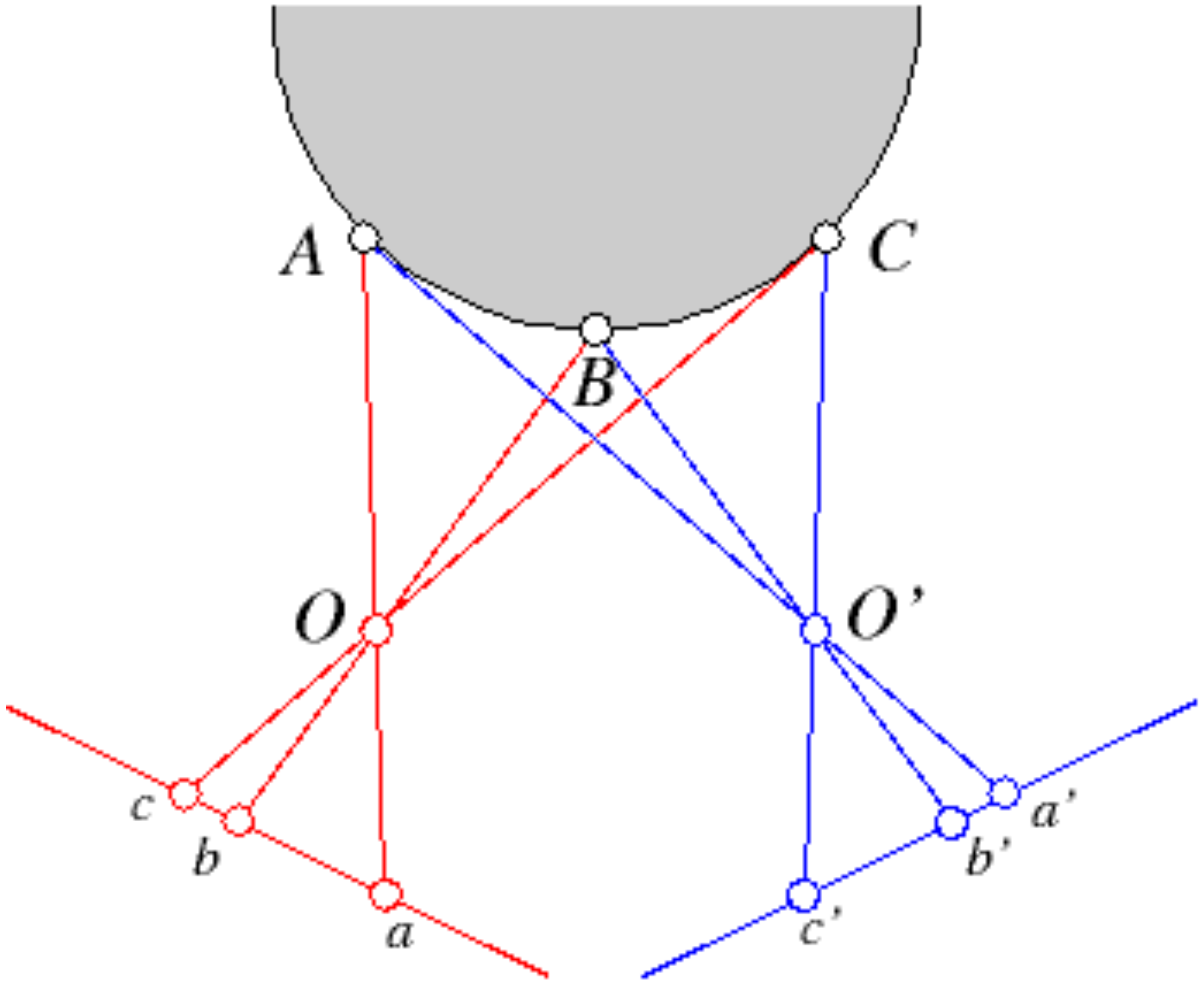


Method: Correlation

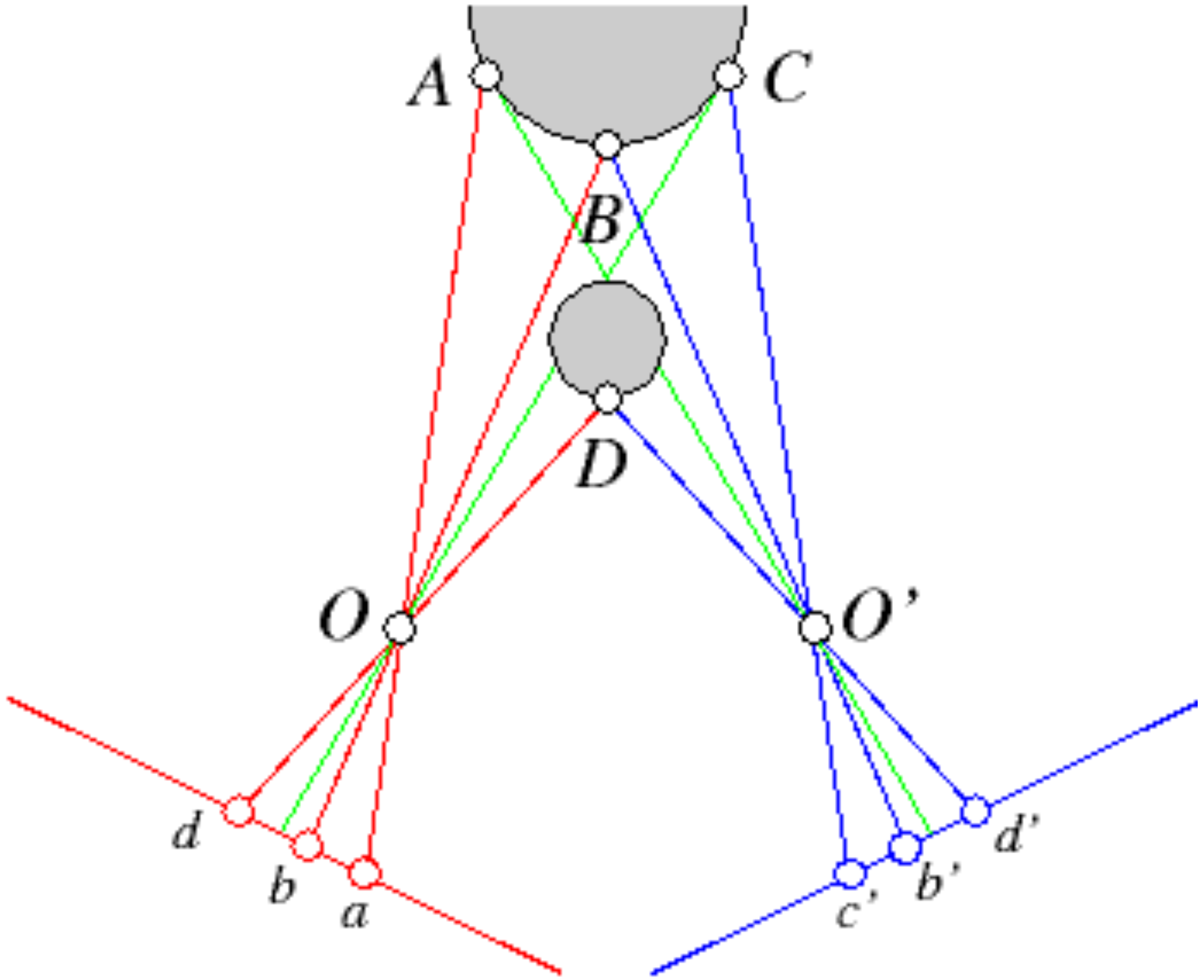


Ordering Constraints

Ordering constraint …



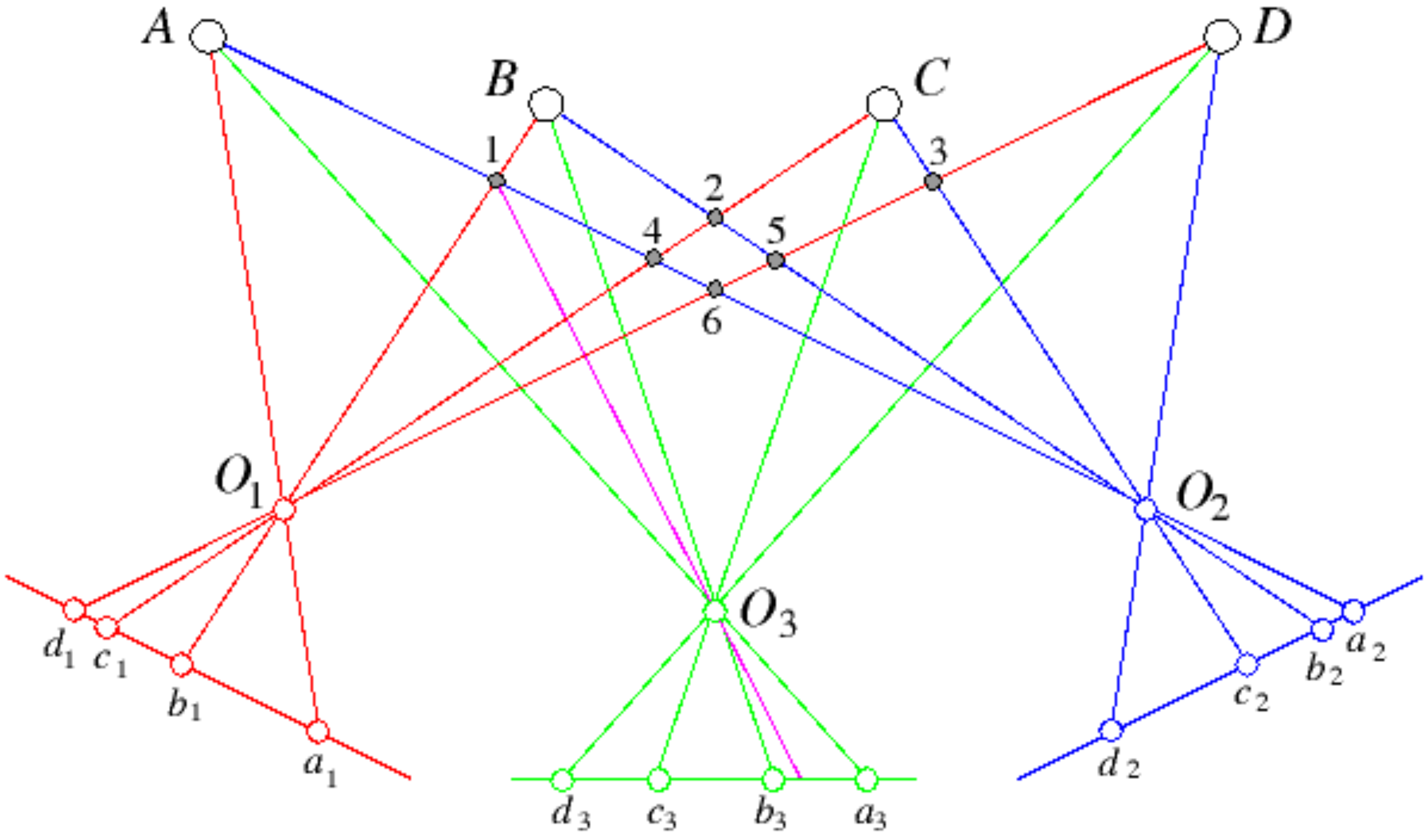
…. and a **failure** case



Forsyth & Ponce (2nd ed.) Figure 7.13

Idea: Use More Cameras

Adding a third camera reduces ambiguity in stereo matching



Forsyth & Ponce (2nd ed.) Figure 7.17

Sample Question

True or **false**: The ordering constraint always holds in stereo vision.

Course Review: Motion and Optical Flow

Motion (geometric), optical flow (radiometric)

Optical flow constraint equation

Lucas-Kanade method

Optical Flow **Constraint Equation**

Consider image intensity also to be a function of time, t. We write

$$I(x, y, t)$$

Applying the **chain rule for differentiation**, we obtain

$$\frac{dI(x, y, t)}{dt} = I_x \frac{dx}{dt} + I_y \frac{dy}{dt} + I_t$$

where subscripts denote partial differentiation

Define $u = \frac{dx}{dt}$ and $v = \frac{dy}{dt}$. Then $[u, v]$ is the 2-D motion and the space of all such u and v is the **2-D velocity space**

Suppose $\frac{dI(x, y, t)}{dt} = 0$. Then we obtain the (classic) **optical flow constraint equation**

$$I_x u + I_y v + I_t = 0$$

How do we **compute** …

$$I_x u + I_y v + I_t = 0$$

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

Forward difference
Sobel filter
Scharr filter
…

$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

Frame differencing

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Frame Differencing: Example

$t+1$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	10	10	10
1	1	10	10	10
1	1	10	10	10

-

t

1	1	1	1	1
1	1	1	1	1
1	10	10	10	10
1	10	10	10	10
1	10	10	10	10
1	10	10	10	10

=

$$I_t = \frac{\partial I}{\partial t}$$

0	0	0	0	0
0	0	0	0	0
0	-9	-9	-9	-9
0	-9	0	0	0
0	-9	0	0	0
0	-9	0	0	0

(example of a forward temporal difference)

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

t

1	1	1	1	1
1	1	1	1	1
1	10	10	10	10
1	10	10	10	10
1	10	10	10	10
1	10	10	10	10

$t+1$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	10	10	10
1	1	10	10	10
1	1	10	10	10

$$I_x = \frac{\partial I}{\partial x}$$

-	0	0	0	-
-	0	0	0	-
-	9	0	0	-
-	9	0	0	-
-	9	0	0	-
-	9	0	0	-

-1 0 1

$$I_y = \frac{\partial I}{\partial y}$$

-	-	-	-	-
0	0	0	0	0
0	9	9	9	9
-	0	0	0	0
0	0	0	0	0
-	-	-	-	-

-1
0
1

$$I_t = \frac{\partial I}{\partial t}$$

0	0	0	0	0
0	0	0	0	0
0	-9	-9	-9	-9
0	-9	0	0	0
0	-9	0	0	0
0	-9	0	0	0

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

How do we **compute** …

$$I_x u + I_y v + I_t = 0$$

$$I_x = \frac{\partial I}{\partial x} \quad I_y = \frac{\partial I}{\partial y}$$

spatial derivative

$$u = \frac{dx}{dt} \quad v = \frac{dy}{dt}$$

optical flow

$$I_t = \frac{\partial I}{\partial t}$$

temporal derivative

Forward difference
Sobel filter
Scharr filter
…

How do you compute this?

Frame differencing

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Lucas-Kanade **Summary**

A dense method to compute motion, $[u, v]$ at every location in an image

Key Assumptions:

1. Motion is slow enough and smooth enough that differential methods apply (i.e., that the partial derivatives, I_x, I_y, I_t, are well-defined)

2. The optical flow constraint equation holds (i.e., $\frac{dI(x,y,t)}{dt} = 0$)

3. A window size is chosen so that motion, $[u, v]$, is constant in the window

4. A window size is chosen so that the rank of $\mathbf{A}^T\mathbf{A}$ is 2 for the window

Sample Question

Describe two examples of imaging situations where motion and optical flow do not coincide.

Course Review: Clustering

K-means clustering

K-Means Clustering

K-means clustering alternates between two steps:

1. Assume the cluster centers are known (fixed). Assign each point to the closest cluster center.

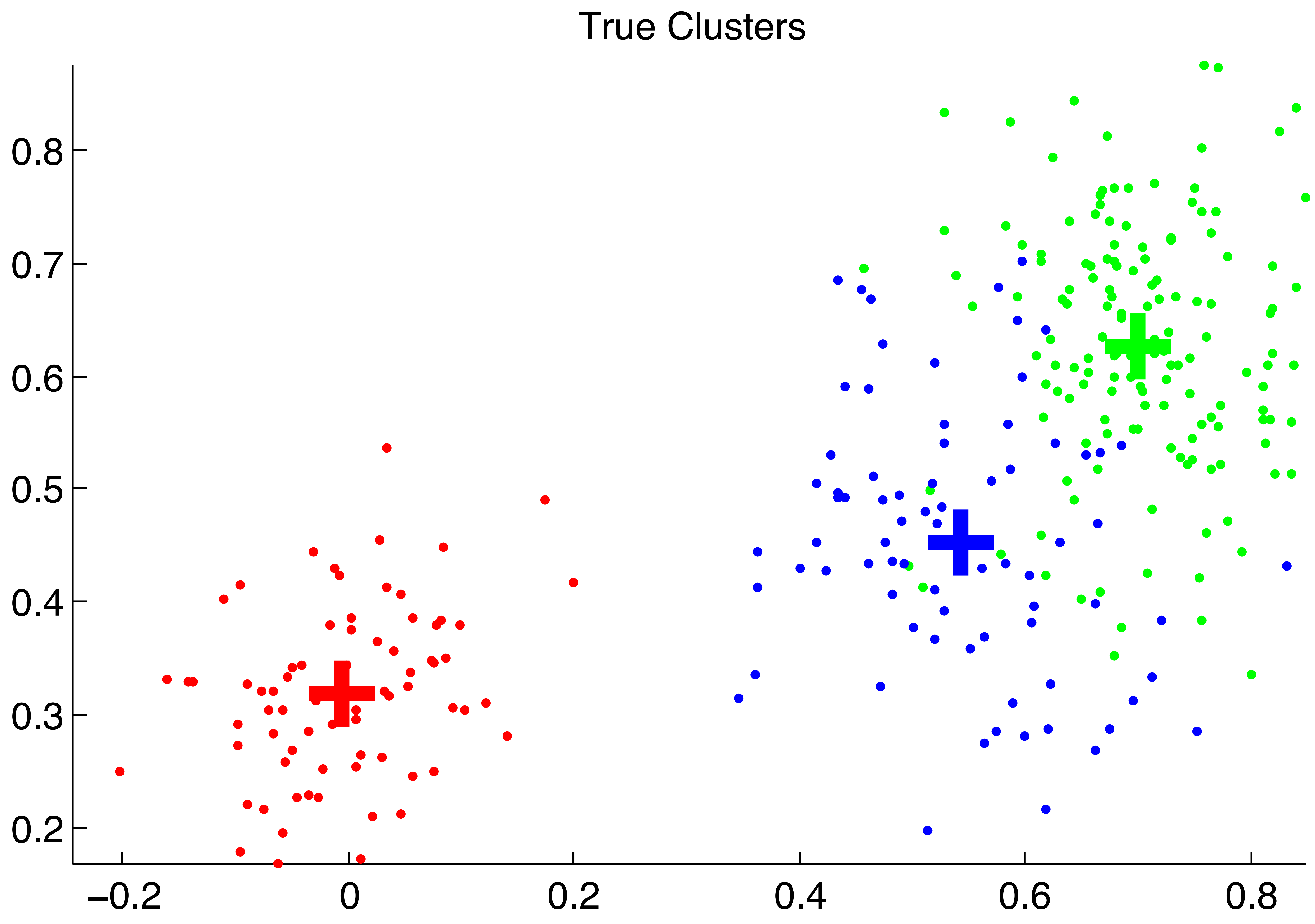
2. Assume the assignment of points to clusters is known (fixed). Compute the best center for each cluster, as the mean of the points assigned to the cluster.

The algorithm is initialized by choosing K random cluster centers

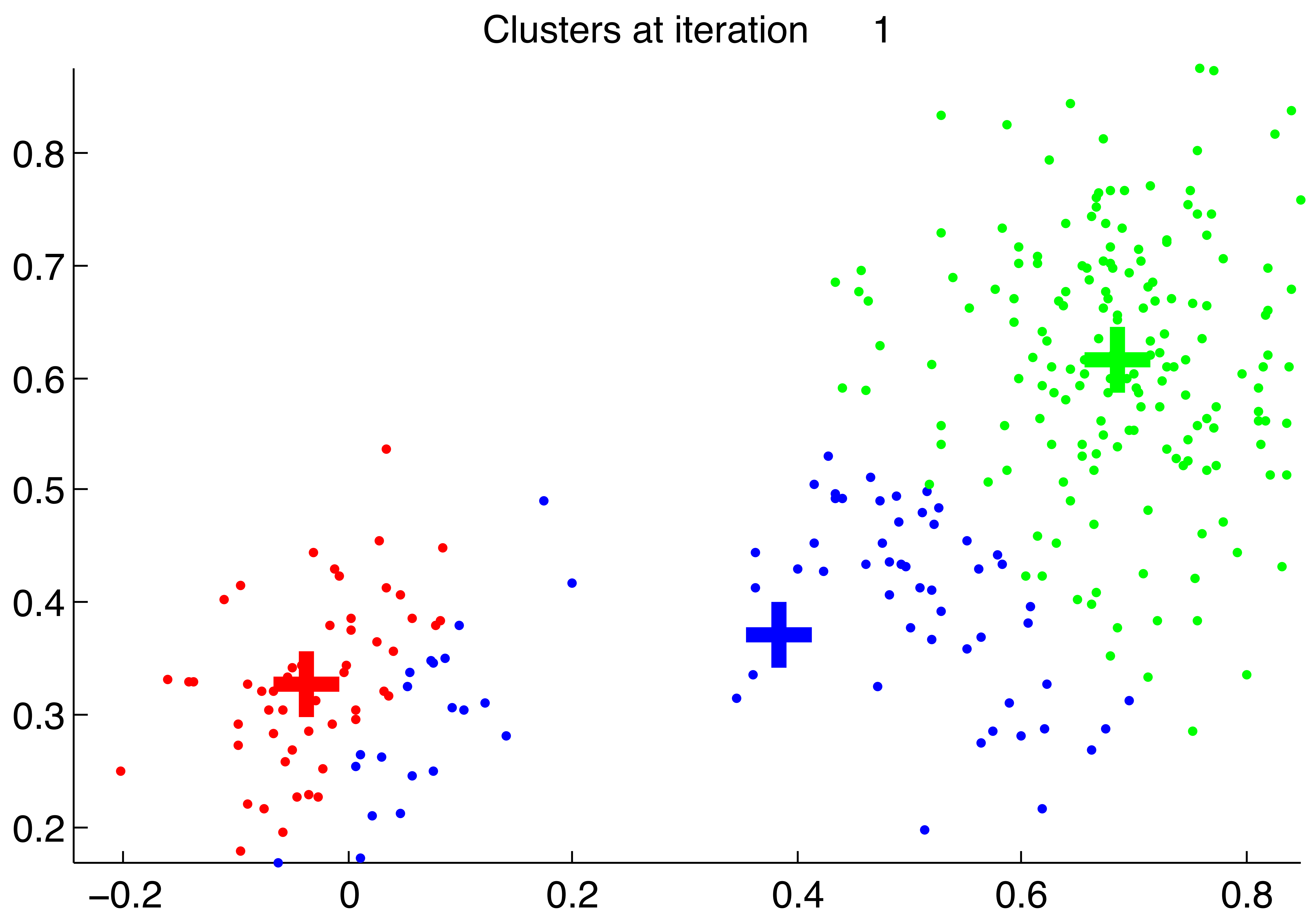
K-means converges to a local minimum of the objective function

— Results are initialization dependent

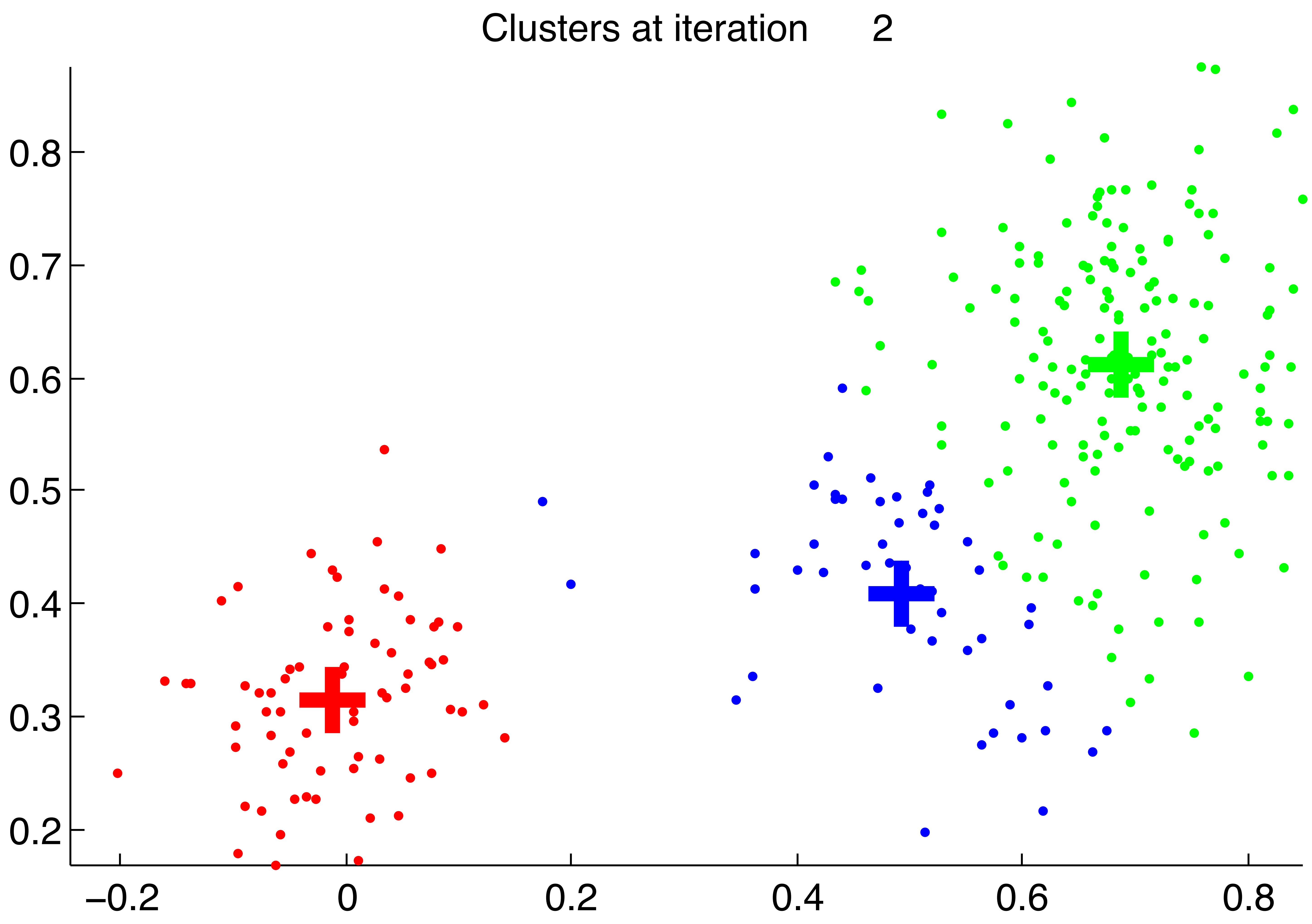
Example 1: K-Means Clustering



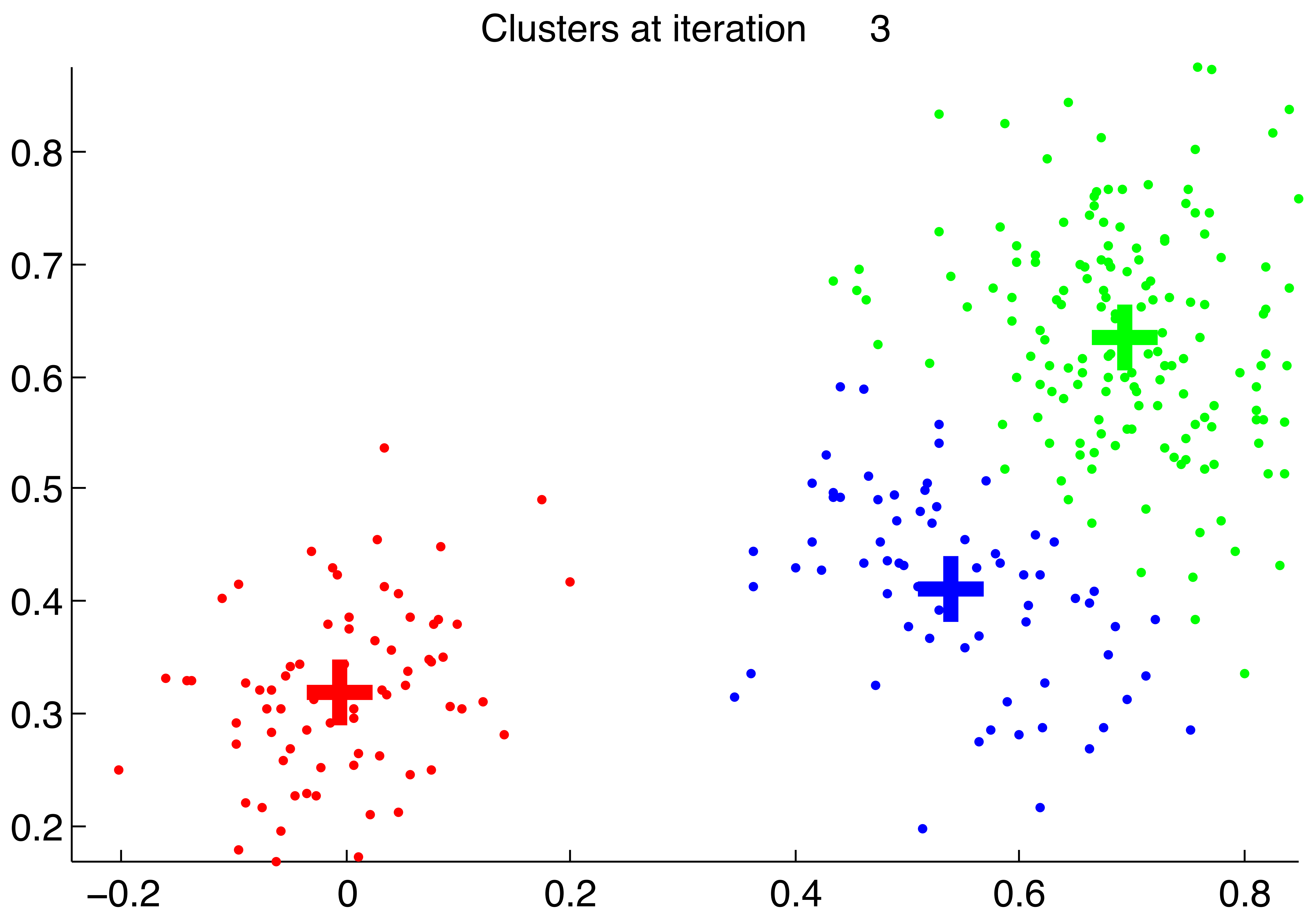
Example 1: K-Means Clustering



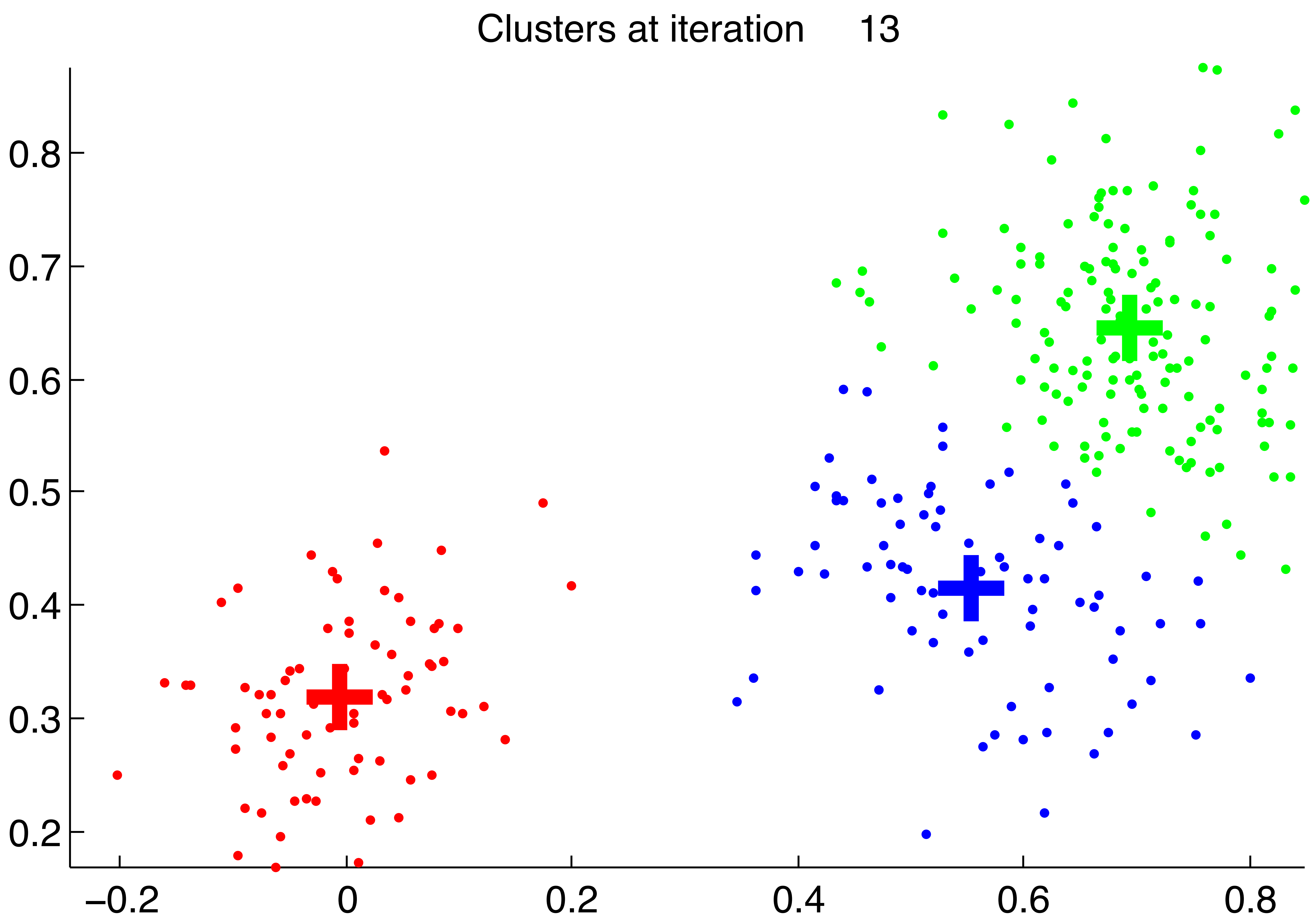
Example 1: K-Means Clustering



Example 1: K-Means Clustering



Example 1: K-Means Clustering



Course Review: Classification

Bayes’ risk, loss functions

Underfitting, overfitting

Cross-**validation**

Receiver Operating Characteristic (**ROC**) curve

Parametric vs. **non-parametric** classifiers

— K-nearest neighbour

— Bayes’ classifier

— Support vector machines

— Decision trees

Course Review: Image Classification

Visual words, codebooks

Bag of words representation

Spatial pyramid

VLAD

Standard **Bag-of-Words** Pipeline (for image classification) — **Training**

Dictionary Learning:
Learn Visual Words using clustering

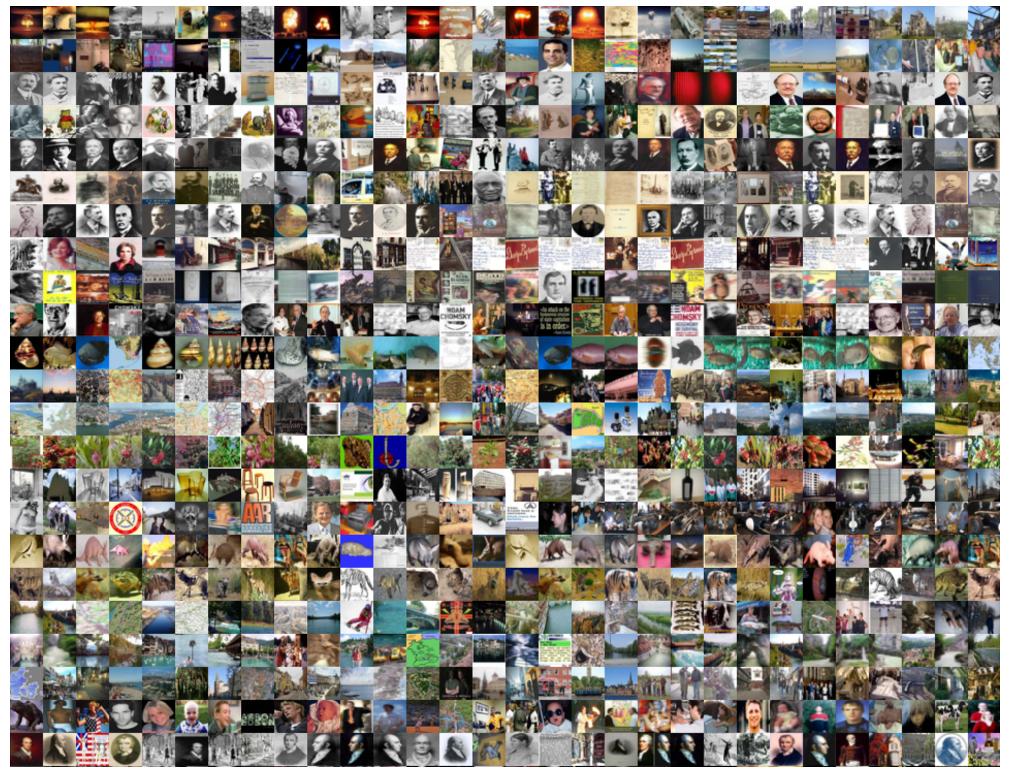
Encode:
build Bags-of-Words (BOW) vectors
for each image

Classify:
Train data using BOWs

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

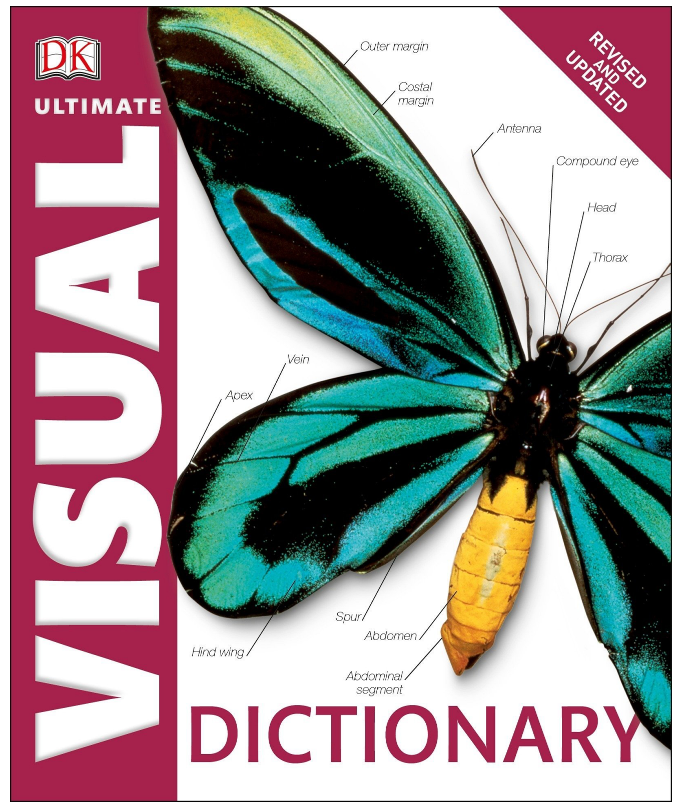
Standard **Bag-of-Words** Pipeline (for image classification) — **Training**

Input: large collection of images
(they don't even need to be training images)



Dictionary Learning:
Learn Visual Words using clustering

Output: dictionary of visual words



Encode:
build Bags-of-Words (BOW) vectors
for each image

Classify:
Train data using BOWs

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Standard **Bag-of-Words** Pipeline (for image classification) — **Training**

Input: large collection of images
(they don't even need to be training images)

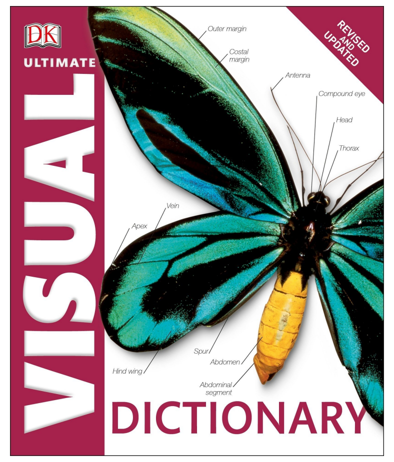
→ **Dictionary Learning**:
Learn Visual Words using clustering

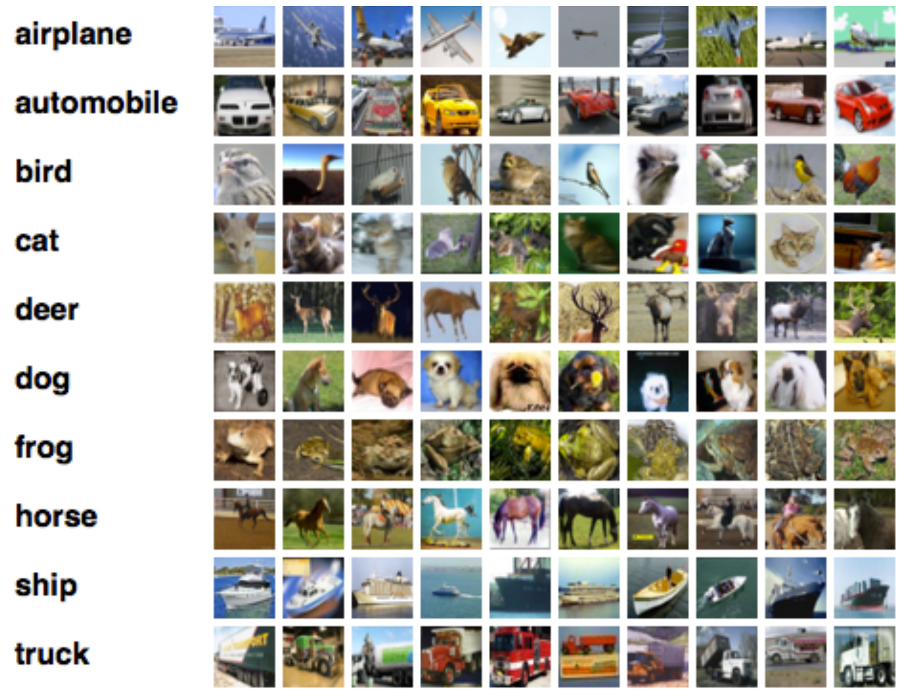
→ **Output**: dictionary of visual words

Input: training images, dictionary

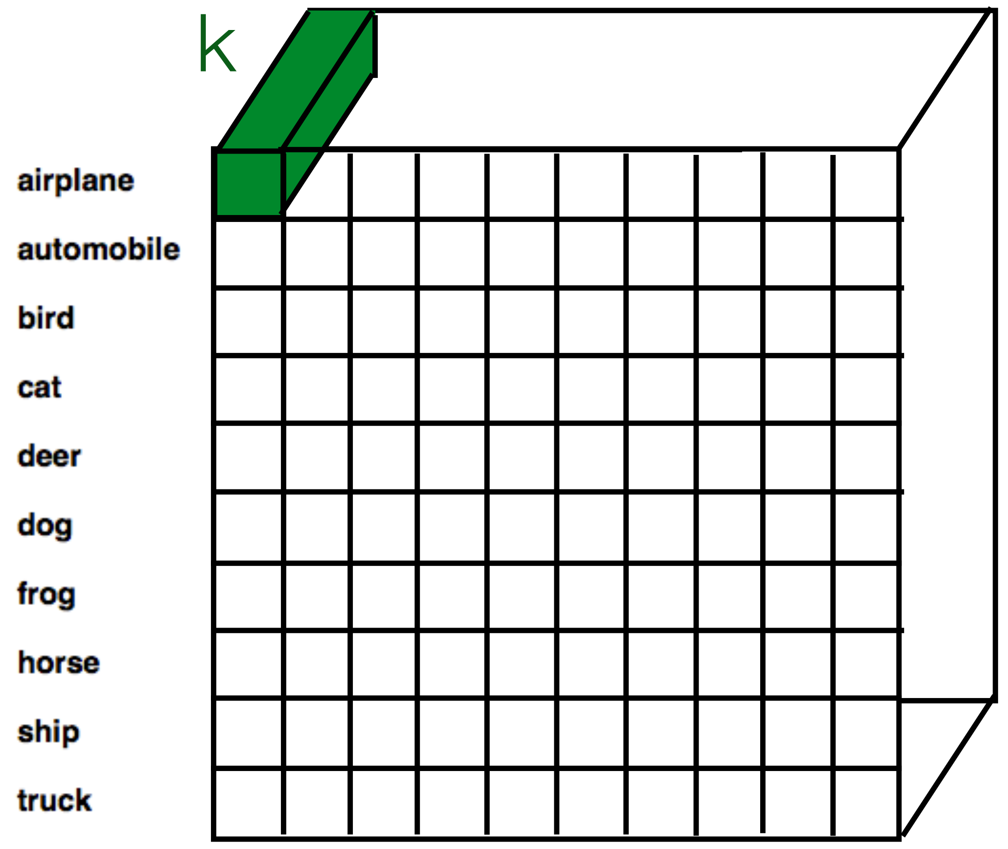
→ **Encode**:
build Bags-of-Words (BOW) vectors for each image

→ **Output**: histogram representation for each training image





Classify:
Train data using BOWs



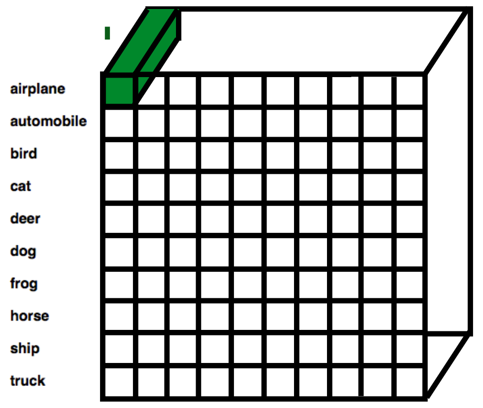
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

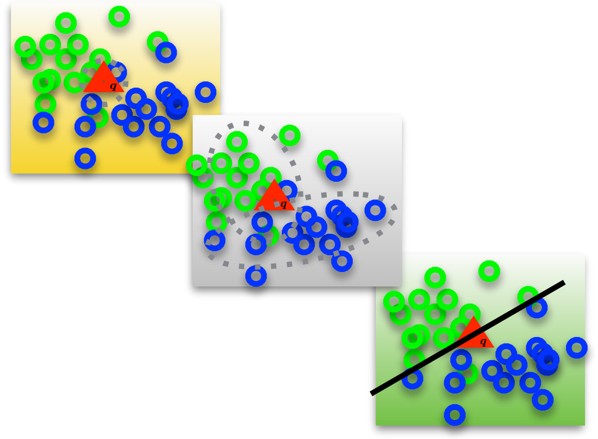
Standard **Bag-of-Words** Pipeline (for image classification) — **Training**

Input: large collection of images
(they don't even need to be training images)
→
Dictionary Learning:
Learn Visual Words using clustering
→
Output: dictionary of visual words

Input: training images, dictionary
→
Encode:
build Bags-of-Words (BOW) vectors for each image
→
Output: histogram representation for each training image

Input: histogram representation for each training image + labels
→
Classify:
Train data using BOWs
→
Output: parameters if the classifier



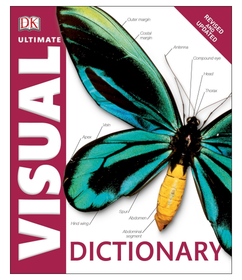


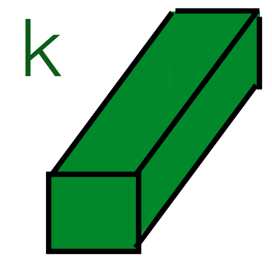
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Standard **Bag-of-Words** Pipeline (for image classification) — **Testing**

Input: large collection of images (they don't even need to be training images) → **Dictionary Learning**: Learn Visual Words using clustering → **Output**: dictionary of visual words

Input: test image, dictionary → **Encode**: build Bags-of-Words (BOW) vectors for each image → **Output**: histogram representation for test image





Classify:
Test data using BOWs

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

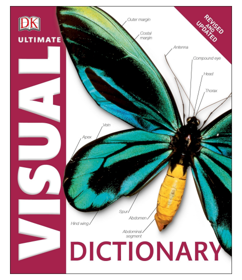
Standard **Bag-of-Words** Pipeline (for image classification) — **Testing**

Input: large collection of images
(they don't even need to be training images)

→ **Dictionary Learning**:
Learn Visual Words using clustering →

Output: dictionary of visual words

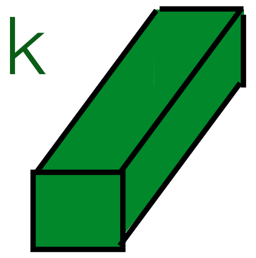
Input: test image, dictionary



→ **Encode**:
build Bags-of-Words (BOW) vectors
for each image →

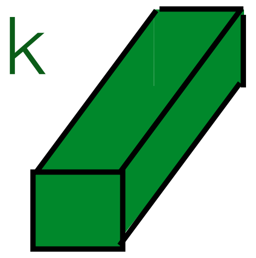
Output: histogram representation for test image

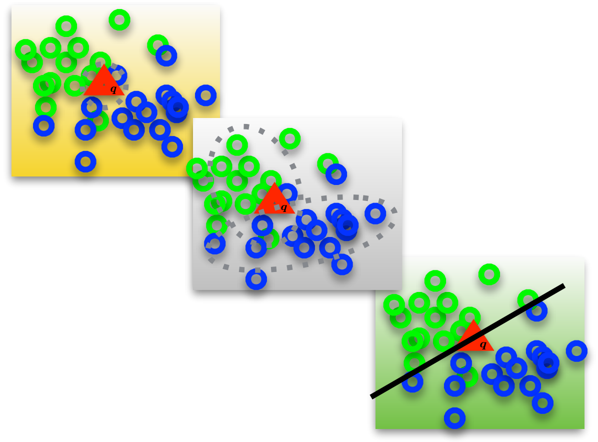
k



Input: histogram representation for test image, trained classifier

k





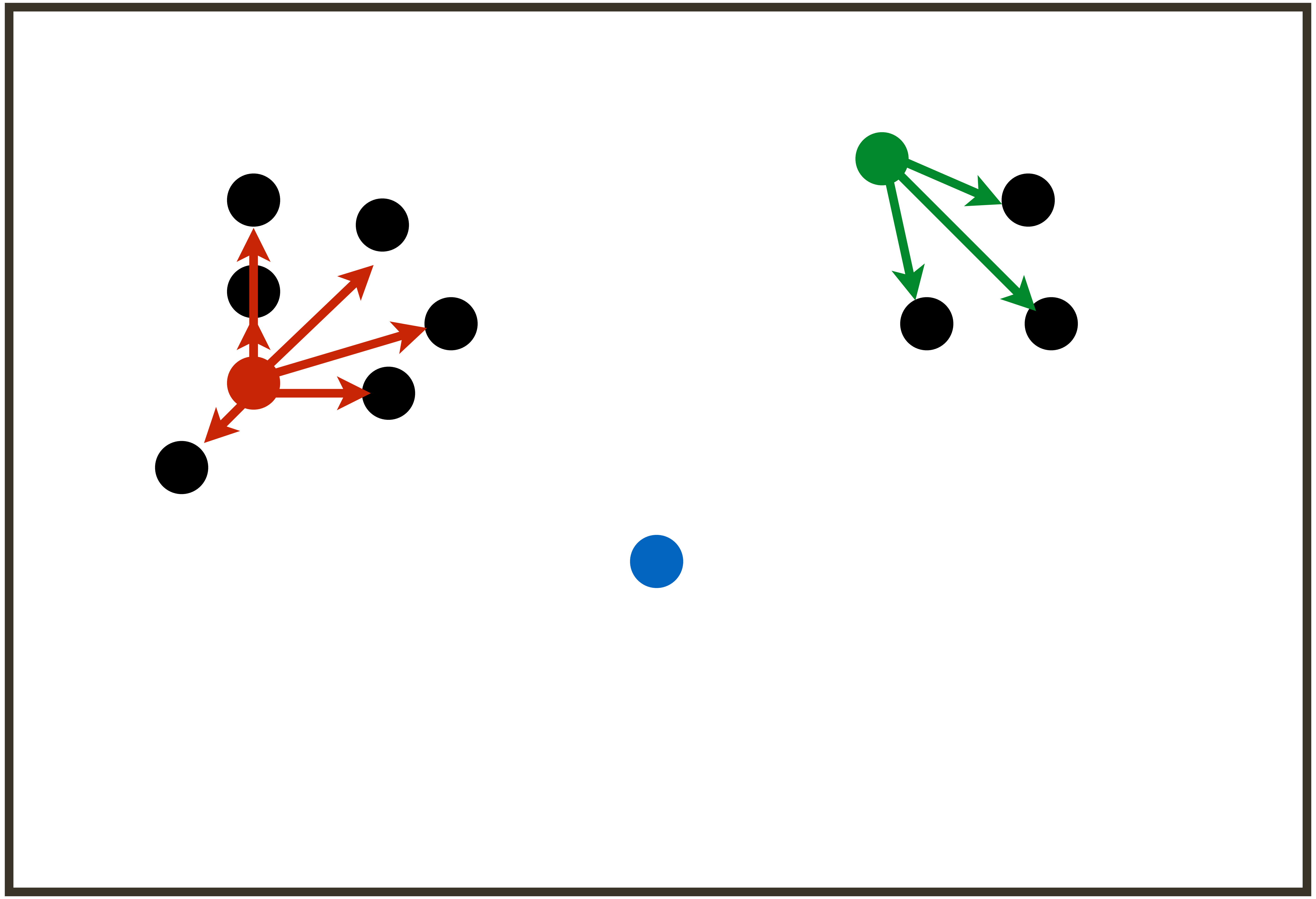
→ **Classify**:
Test data using BOWs →

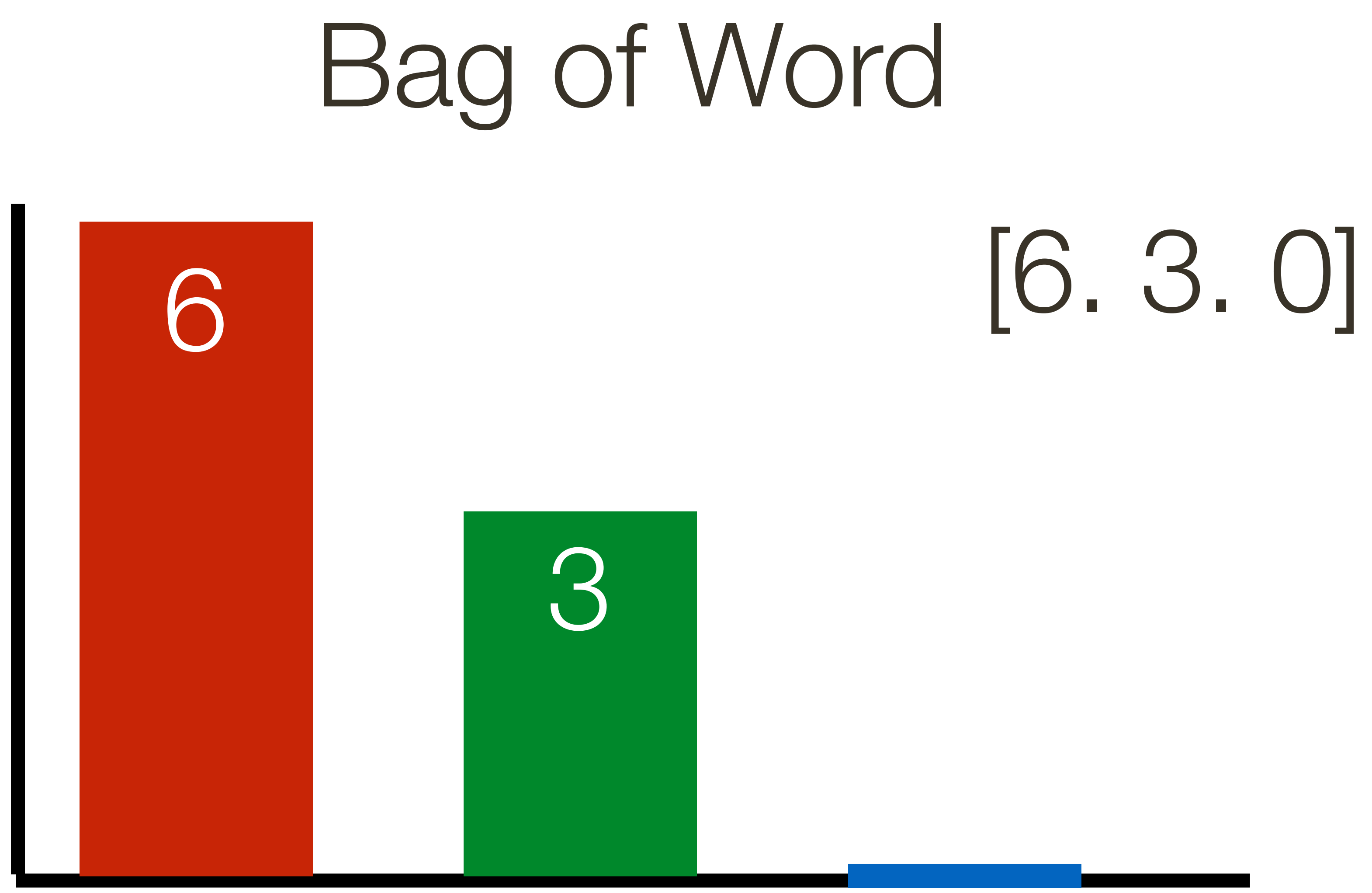
Output: prediction for test image

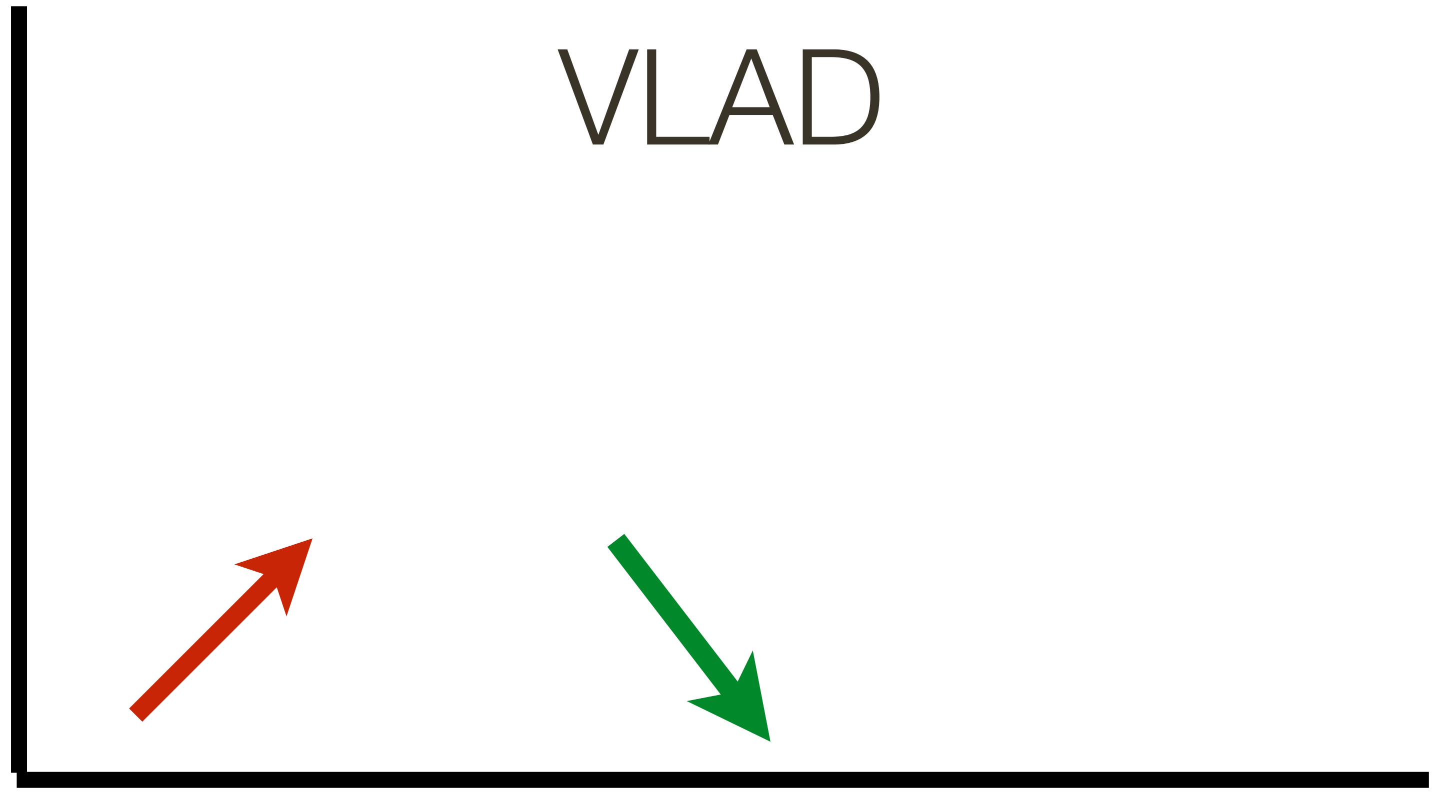
airplane

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: VLAD







Sample Question

How do we construct a codebook (vocabulary) of local descriptors, say SIFT?

Course Review: Object Detection

Sliding window

Viola-Jones face detection

Object **proposals**

Sliding Window

Train an image classifier as described previously. ‘Slide’ a fixed-sized detection window across the image and evaluate the classifier on each window.

Image credit: KITTI Vision Benchmark

This is a search over location
— We have to search over scale as well
— We may also have to search over aspect ratios

Example: Face Detection

1. Select best filter/threshold combination

   a. Normalize the weights $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$

   $$h_j(x) = \begin{cases} 1 & \text{if } f_j(x) > \theta_j \\ 0 & \text{otherwise} \end{cases}$$

   b. For each feature, *j* $\varepsilon_j = \sum_i w_i \left| h_j(x_i) - y_i \right|$

   c. Choose the classifier, h_t with the lowest error ε_t

2. Re-weight examples

$$w_{t+1,i} = w_{t,i} \beta_t^{1-|h_t(x_i)-y_i|} \qquad \beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$$

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

Viola & Jones algorithm

3. The final strong classifier is

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

$$\alpha_t = \log \frac{1}{\beta_t}$$

The final strong classifier is a weighted linear combination of the T weak classifiers where the weights are inversely proportional to the training errors

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Cascading Classifiers

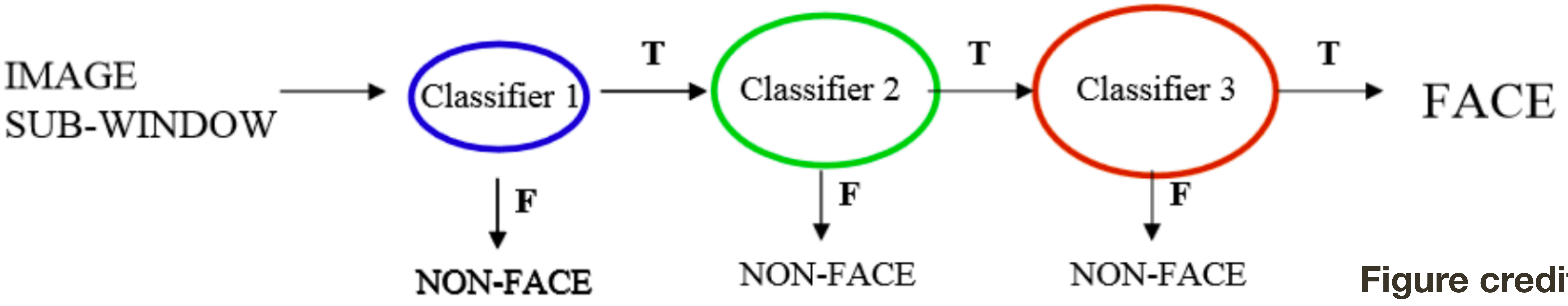


Figure credit: P. Viola

To make detection **faster**, features can be reordered by increasing complexity of evaluation and the thresholds adjusted so that the early (simpler) tests have few or no false negatives

Any window that is rejected by early tests can be discarded quickly without computing the other features

This is referred to as a **cascade** architecture

Object Proposals

First introduced by Alexe et al., who asked ‘what is an object?’ and defined an ‘objectness’ score based on several visual cues

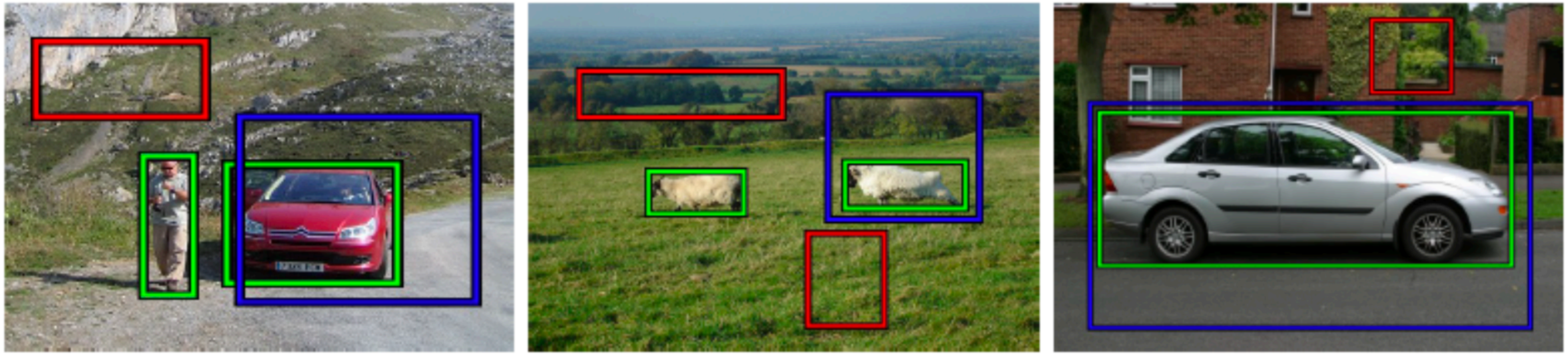


Figure credit: Alexe et al., 2012

Course Review: Convolutional Neural Networks

Neuron, activation function

Backpropagation (you only need to know properties)

Convolutional neural network architecture

Convolutional neural network layers

R-CNN

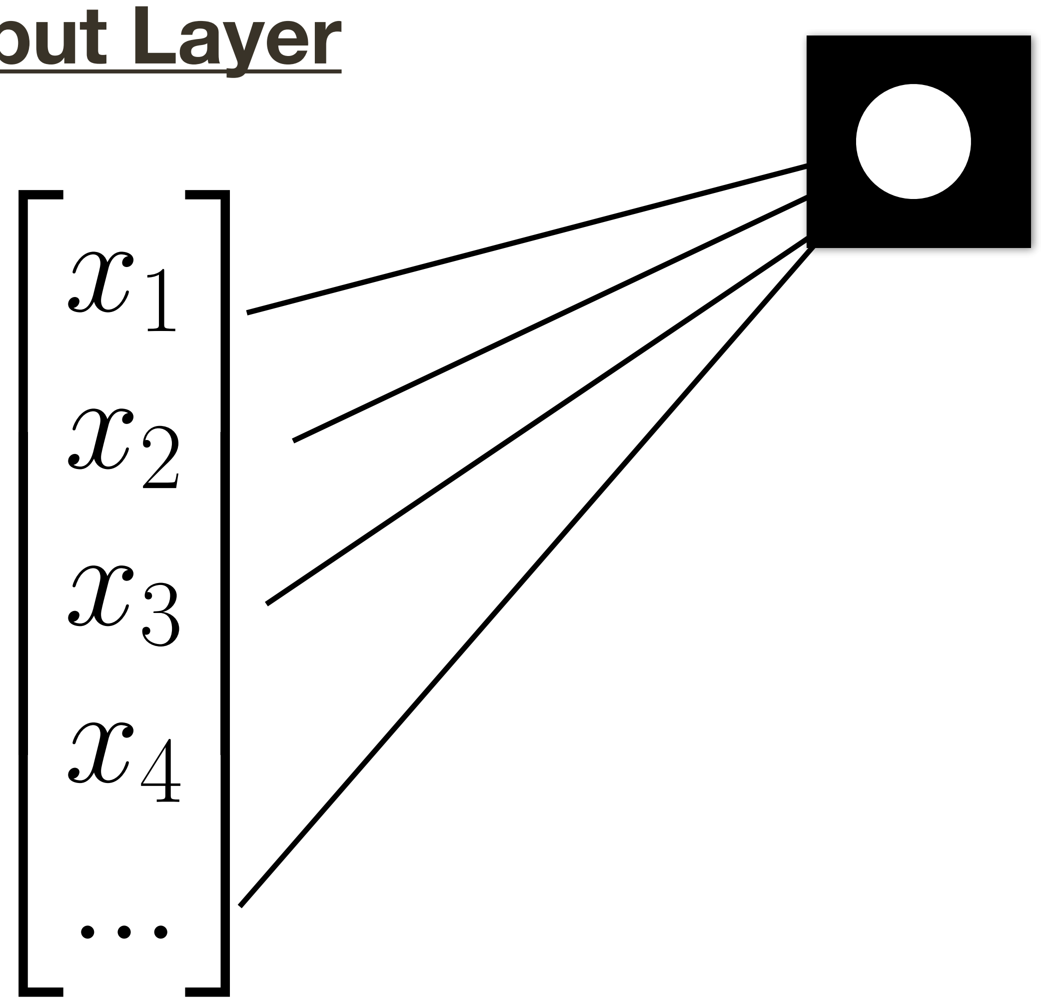
Neural Network: Short Review

Neural Network: Short Review

$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

Input Layer

Input Image



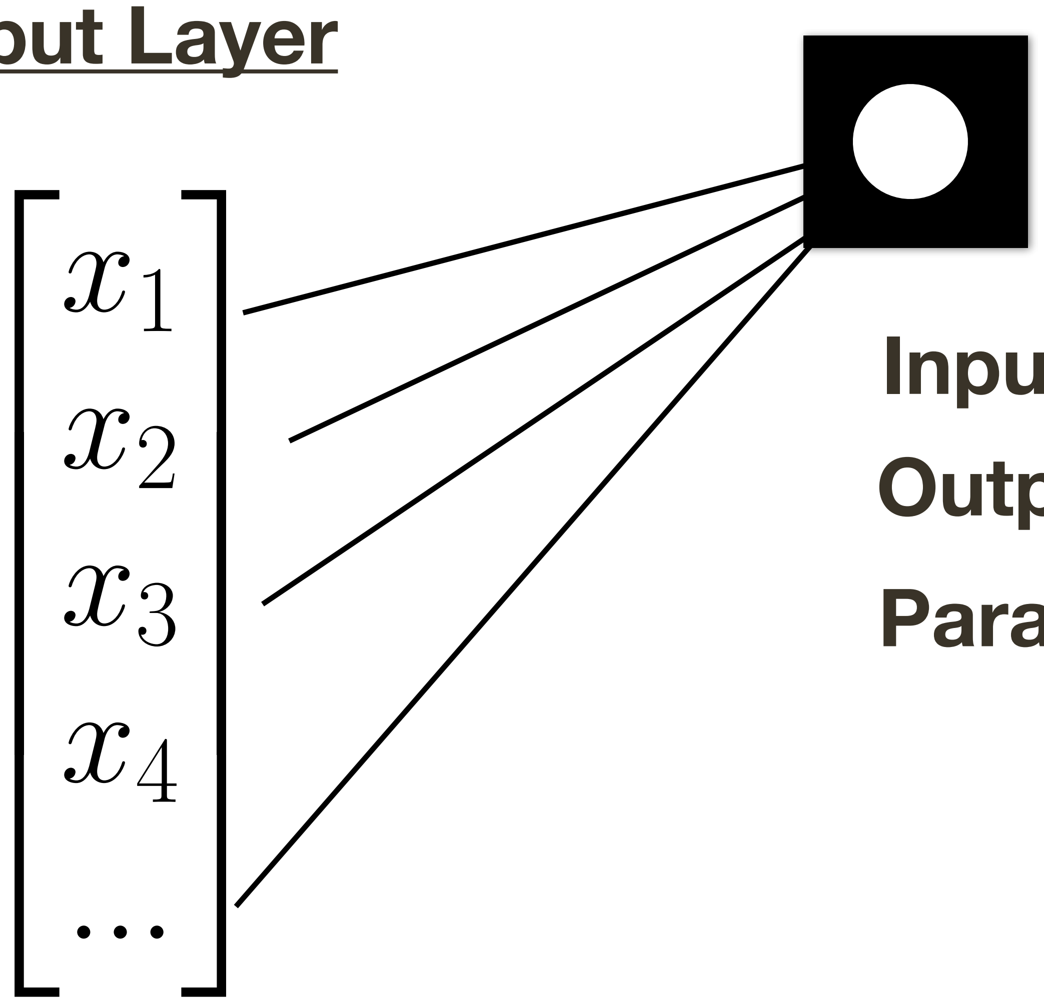
Vectorized Input
(32 x 32 x 3) = 3072

Neural Network: Short Review

$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

Input Image

Input Layer



Vectorized Input
(32 x 32 x 3) = 3072

Inputs: 3072

Outputs: 1

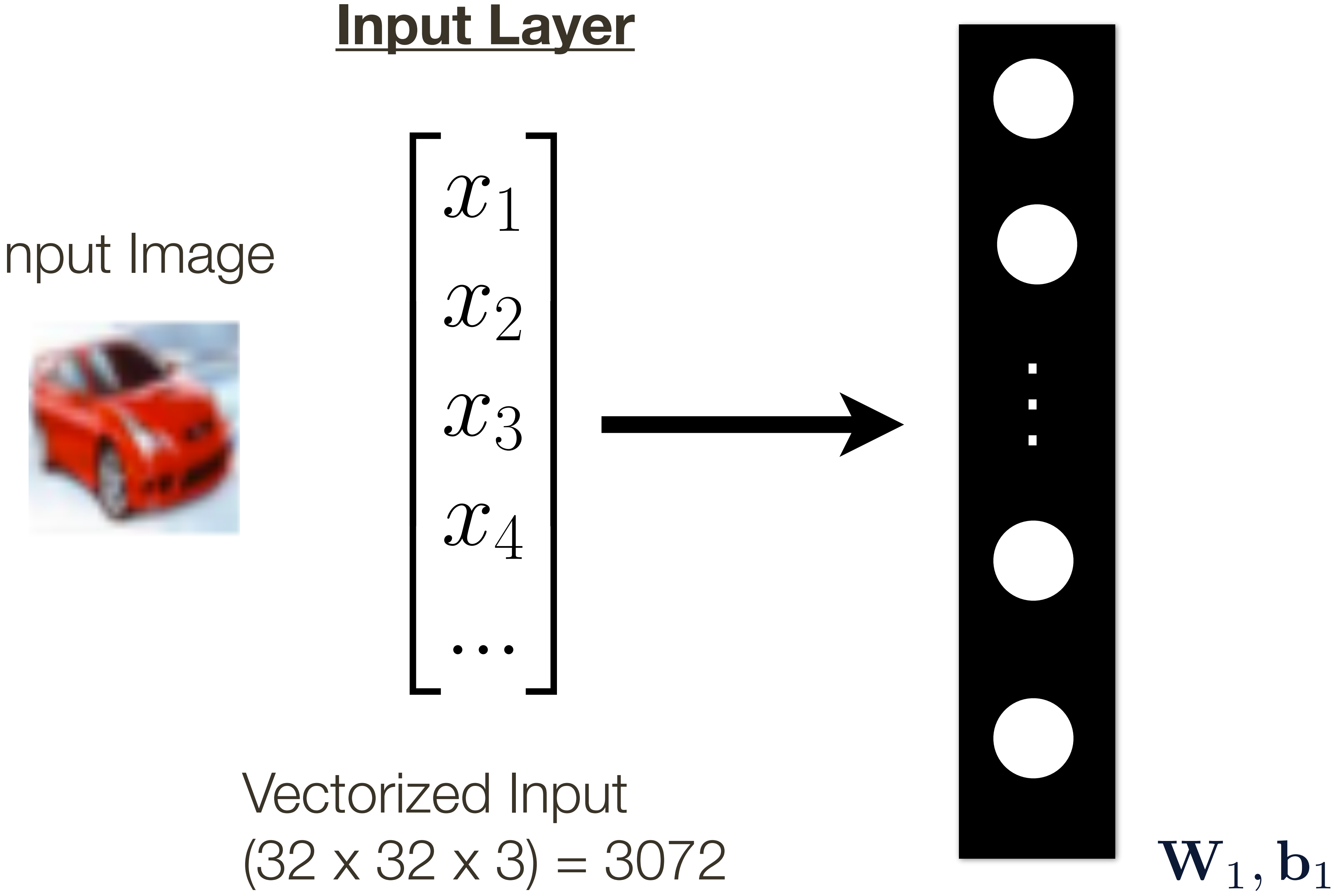
Parameters: 3072 + 1

Neural Network: Short Review

Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Input Layer

Input Image



Vectorized Input
(32 x 32 x 3) = 3072

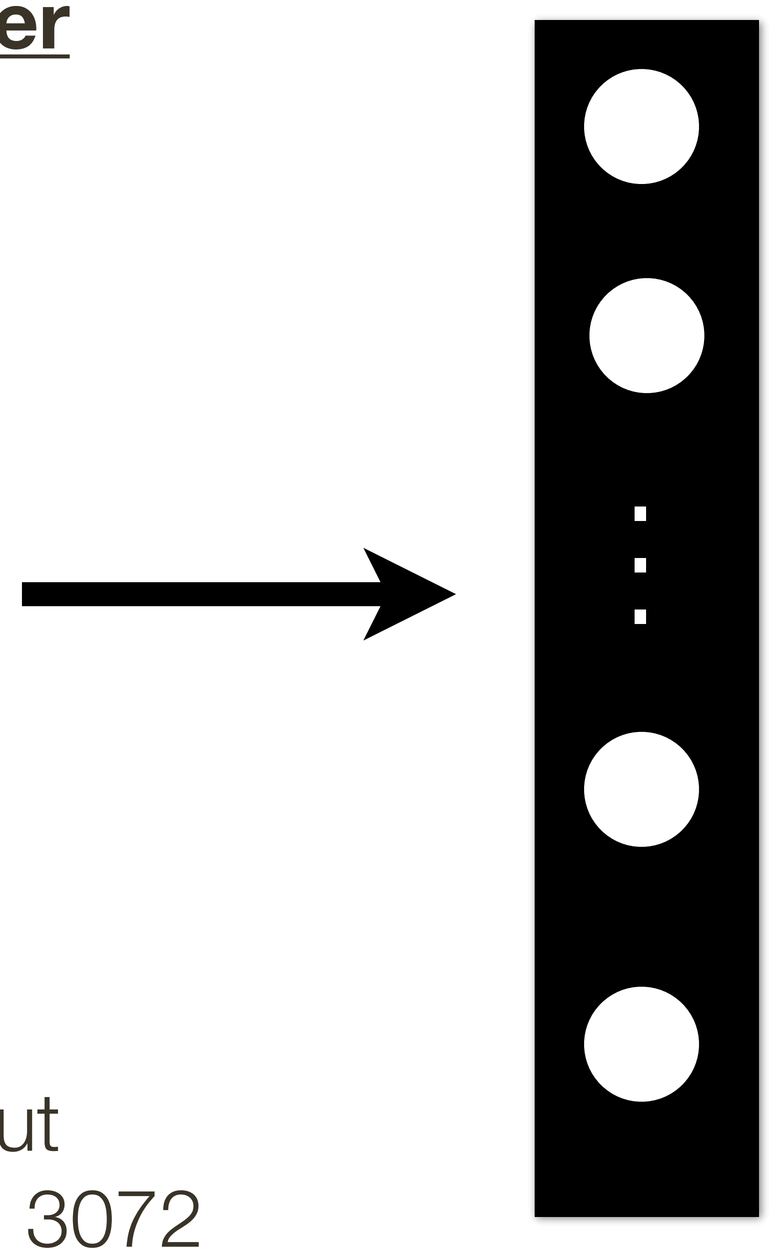
Neural Network: Short Review

Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Input Layer

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \ldots \end{bmatrix}$$



$\mathbf{W}_1, \mathbf{b}_1$

Vectorized Input
(32 x 32 x 3) = 3072

Inputs: 3072

Outputs: 400

Parameters:
3072 x 400 + 400

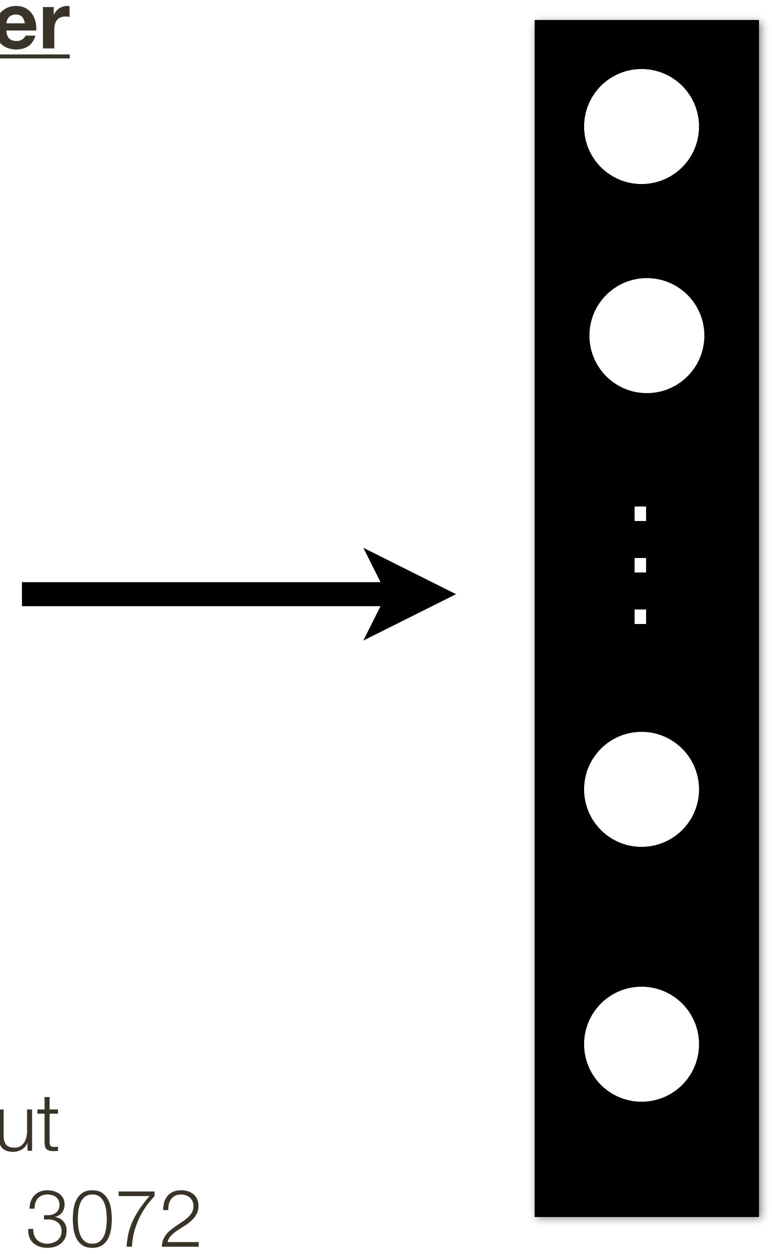
Neural Network: Short Review

Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Input Layer

Input Image

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \cdots \end{bmatrix}$$



$\mathbf{W}_1, \mathbf{b}_1$

Vectorized Input
(32 x 32 x 3) = 3072

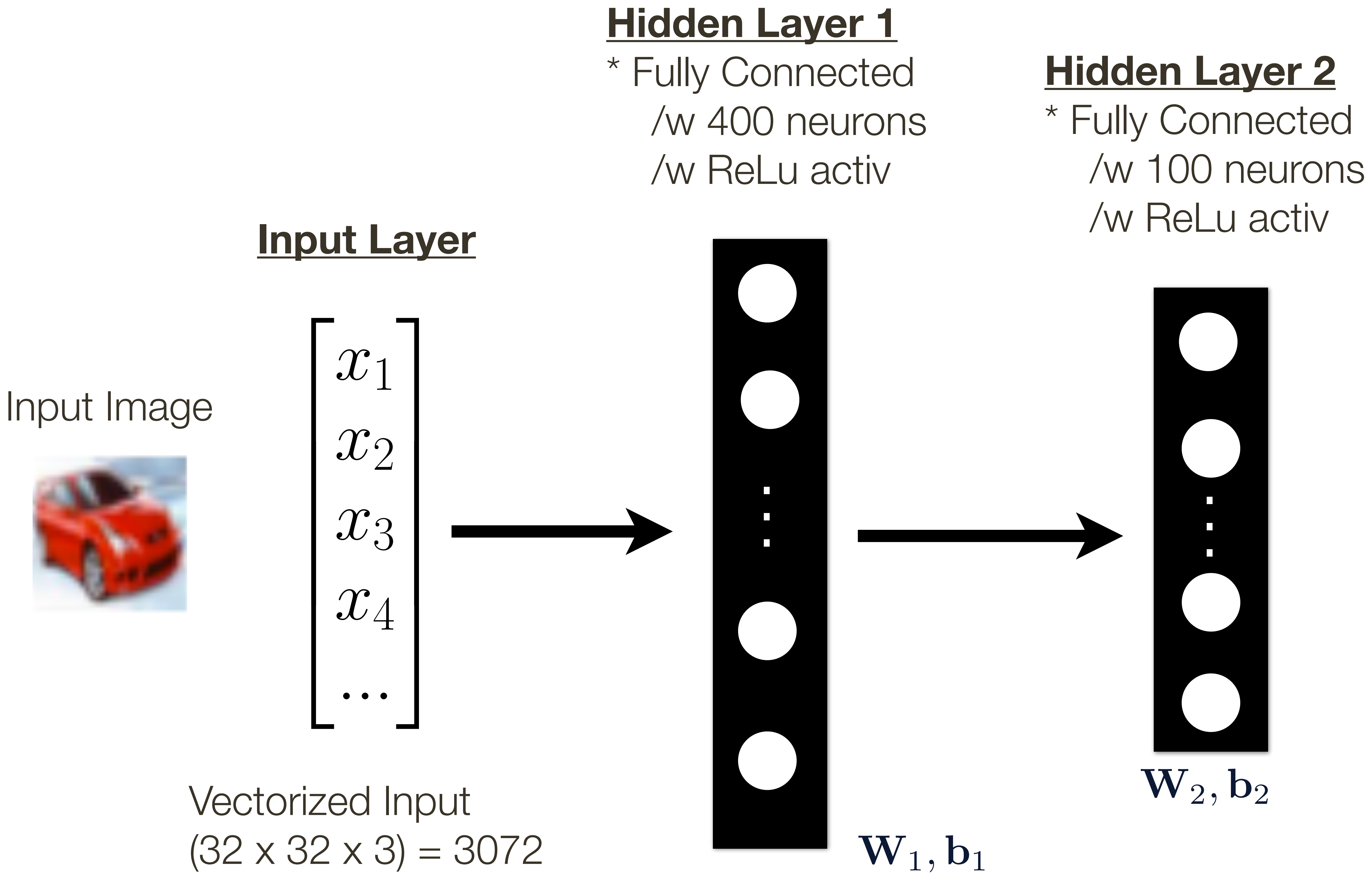
Inputs: 3072

Outputs: 400

Parameters:
3072 x 400 + 400

Note: All neurons within a layer can be computed in parallel, making computations very efficient (especially on GPUs!, which are designed for parallelism)

Neural Network: Short Review



Inputs: 3072

Outputs: 400

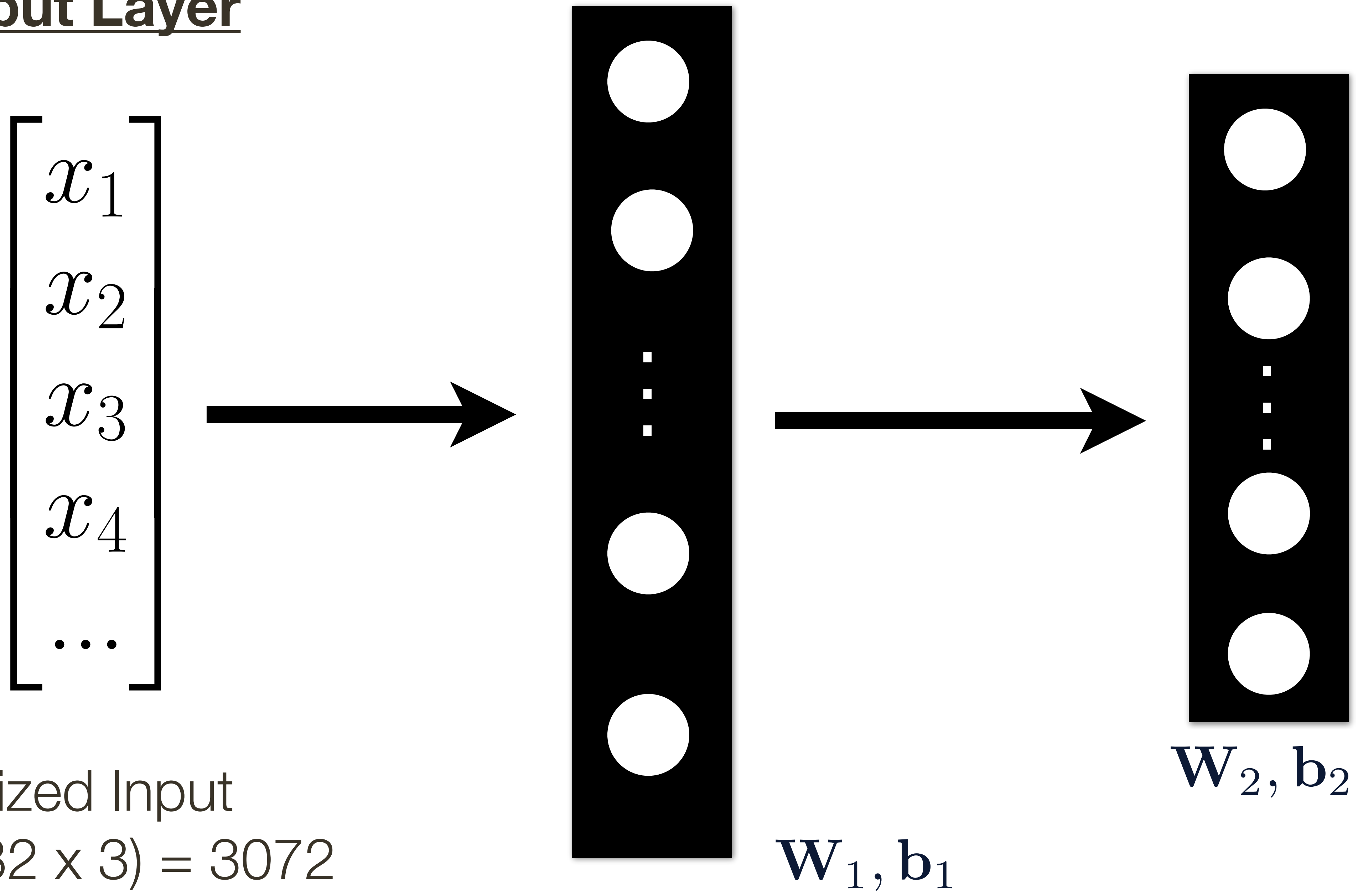
Parameters:
3072 x 400 + 400

Note: Across layers computations are sequential

Neural Network: Short Review

Input Image

Input Layer



Vectorized Input
(32 x 32 x 3) = 3072

Hidden Layer 1
* Fully Connected
  /w 400 neurons
  /w ReLu activ

Inputs: 3072

Outputs: 400

Parameters:
  3072 x 400 + 400

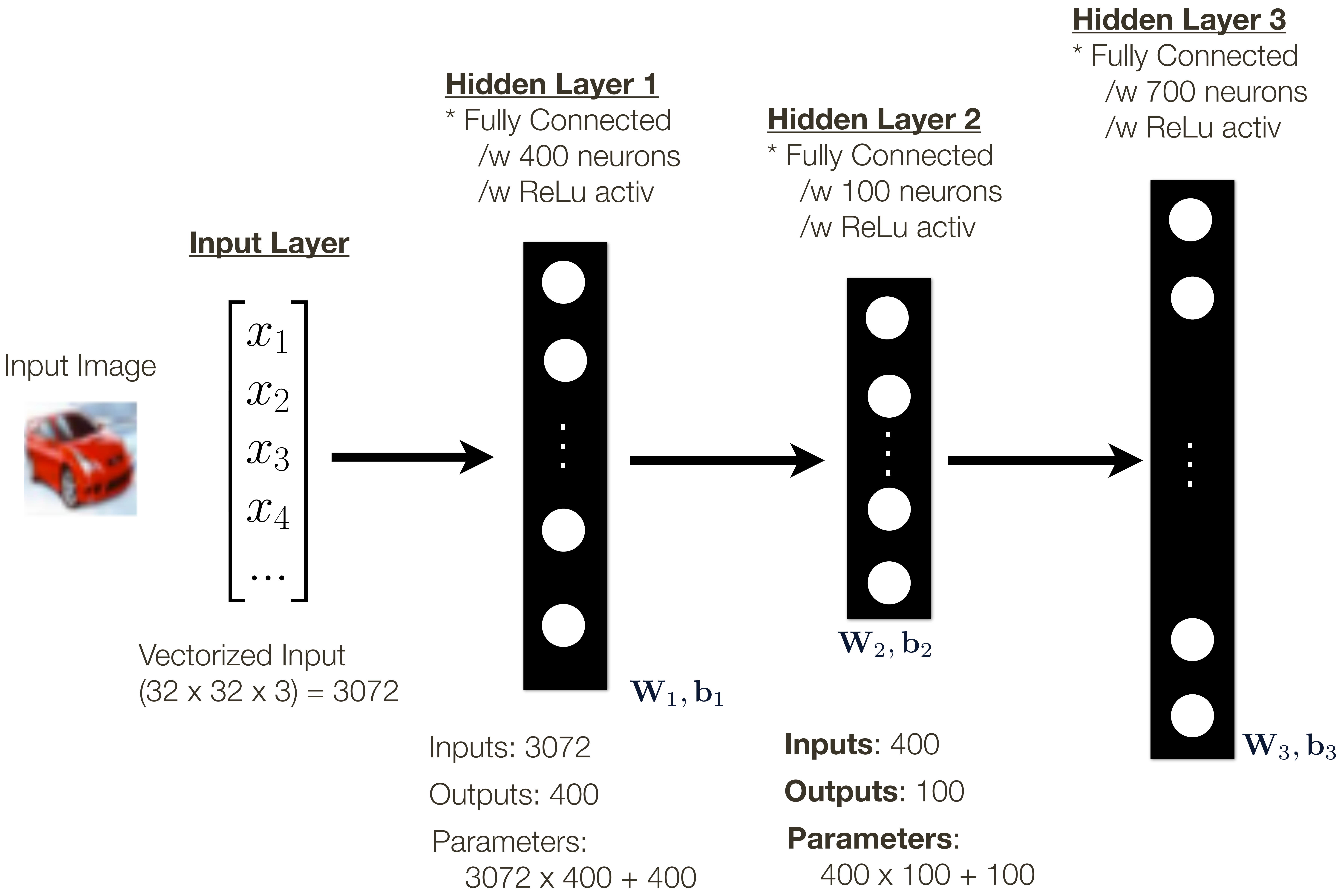
Hidden Layer 2
* Fully Connected
  /w 100 neurons
  /w ReLu activ

Inputs: 400

Outputs: 100

Parameters:
  400 x 100 + 100

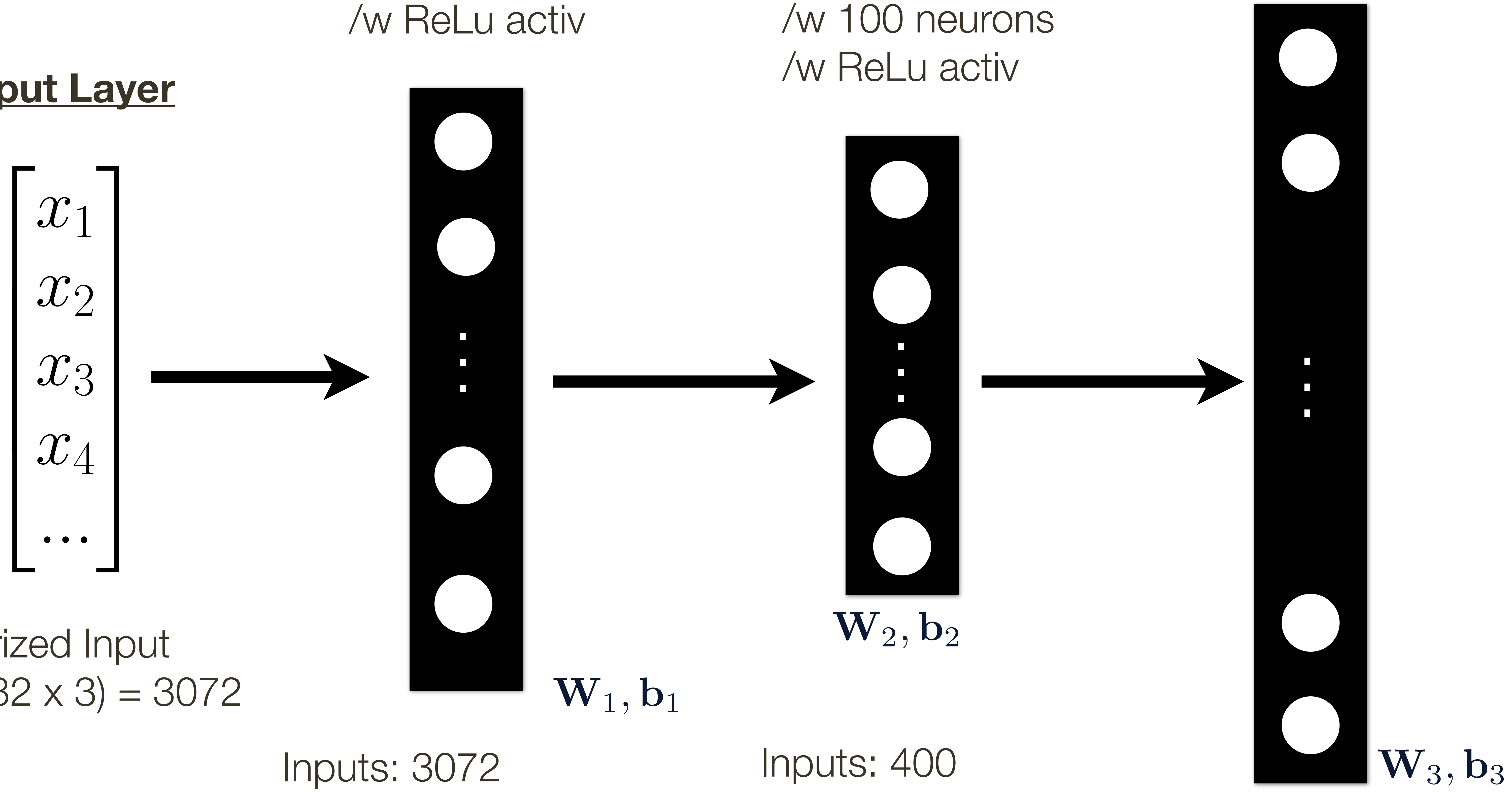
Neural Network: Short Review



Neural Network: Short Review

Input Image

Input Layer



Vectorized Input
(32 x 32 x 3) = 3072

Hidden Layer 1
* Fully Connected
/w 400 neurons
/w ReLu activ

Inputs: 3072

Outputs: 400

Parameters:
3072 x 400 + 400

Hidden Layer 2
* Fully Connected
/w 100 neurons
/w ReLu activ

Inputs: 400

Outputs: 100

Parameters:
400 x 100 + 100

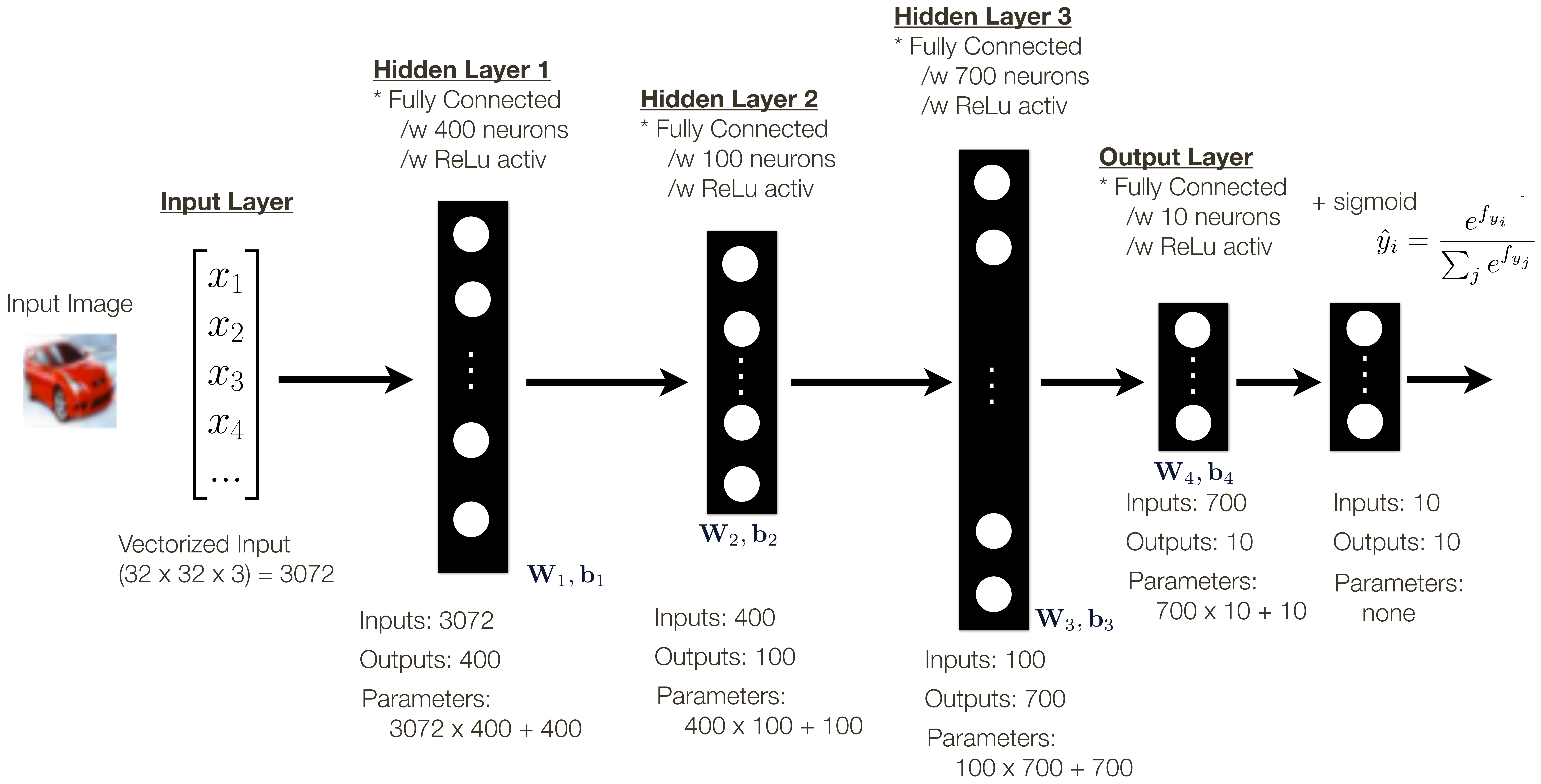
Hidden Layer 3
* Fully Connected
/w 700 neurons
/w ReLu activ

Inputs: 100

Outputs: 700

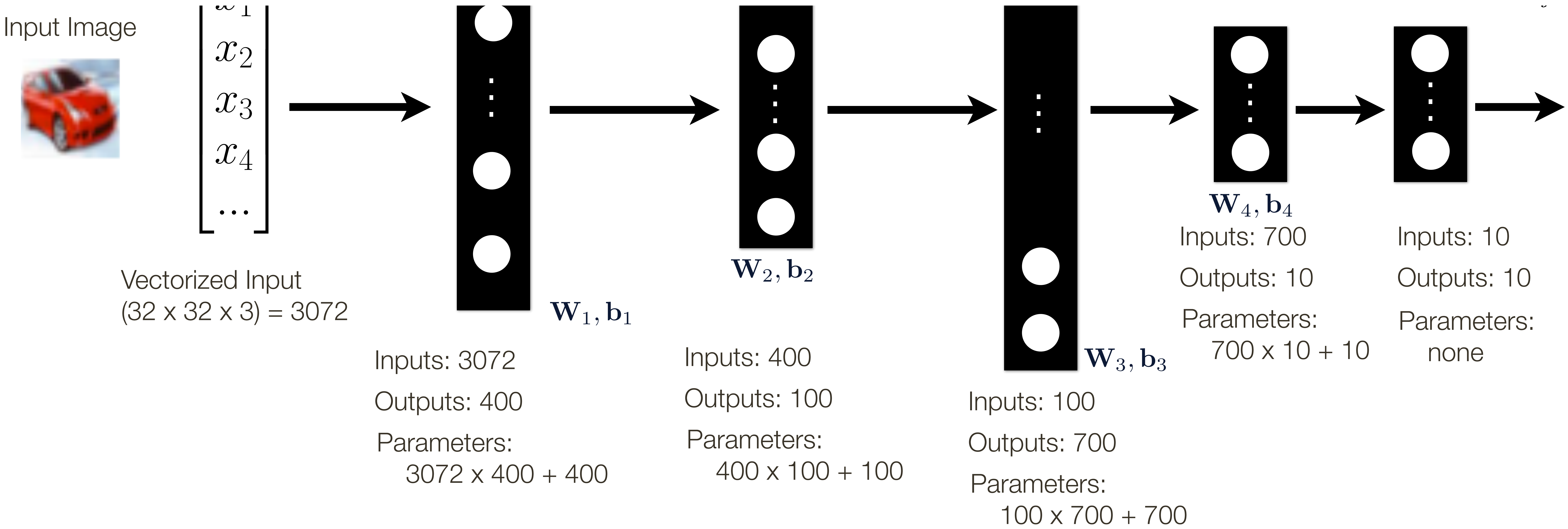
Parameters:
100 x 700 + 700

Neural Network: Short Review

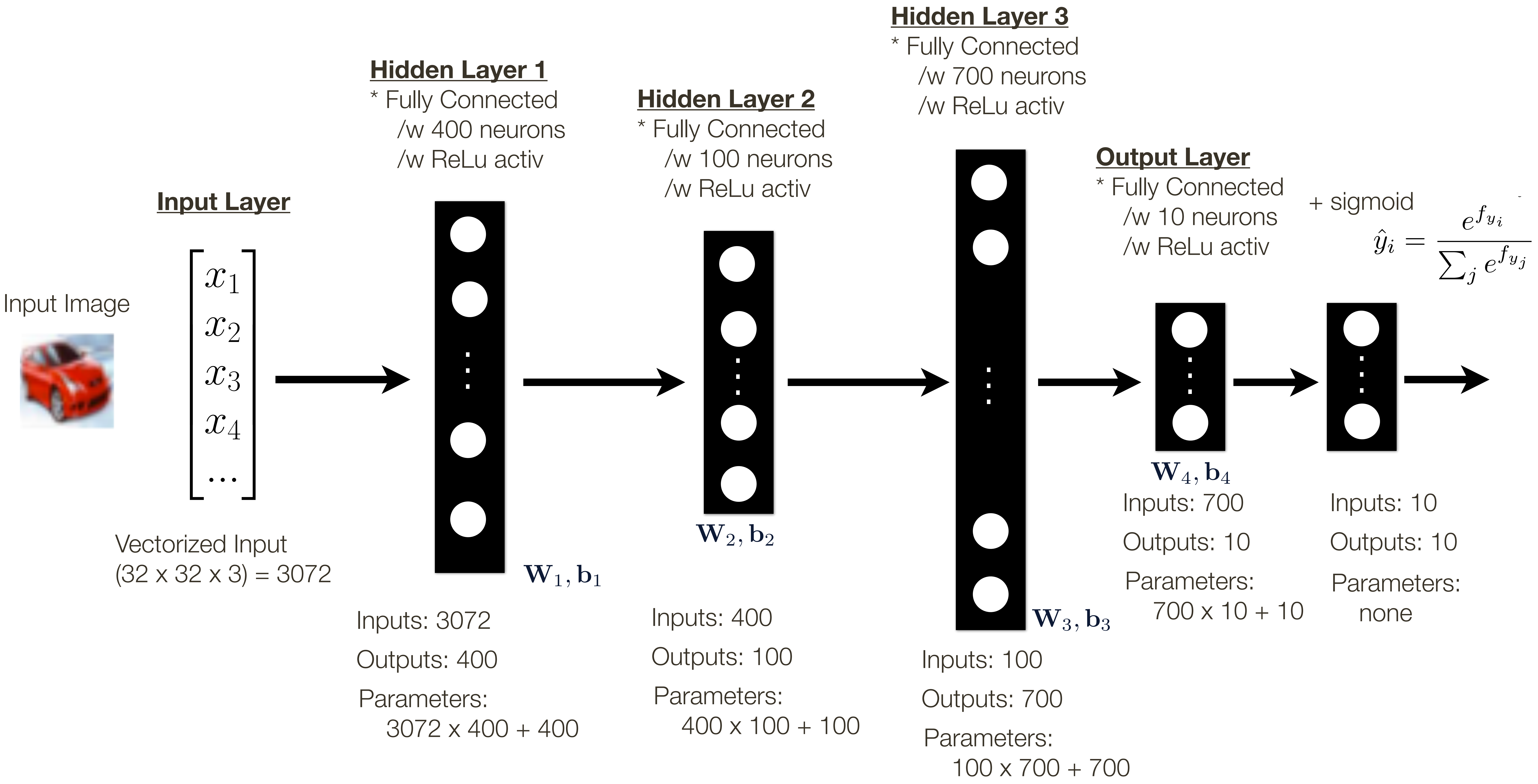


Neural Network: Short Review

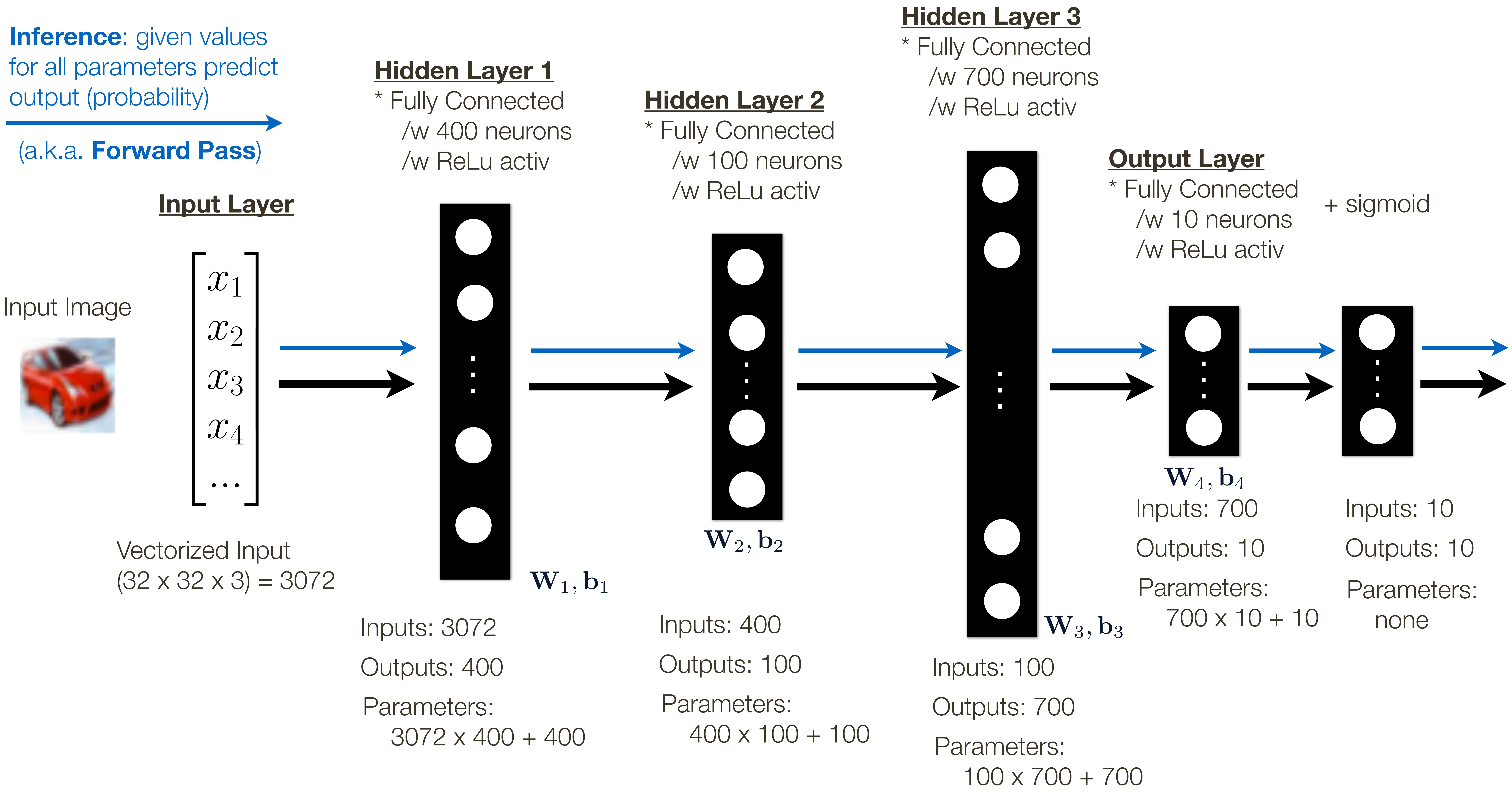
This simple neural network has nearly 1.35 million parameters



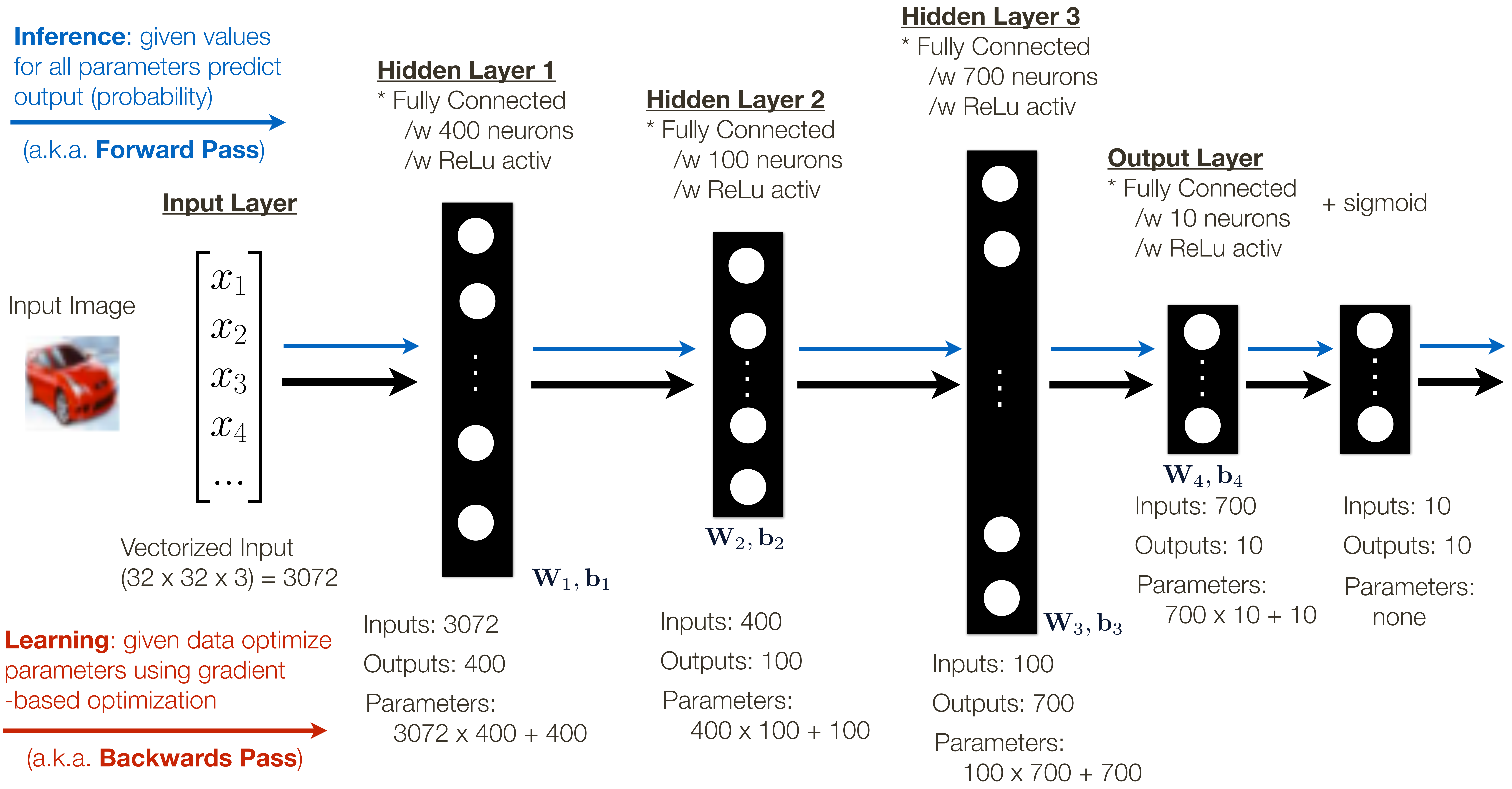
Neural Network: Short Review



Neural Network: Short Review



Neural Network: Short Review



Neural Network: Short Review

Neural Network: Short Review

Neural Network: Short Review

Neural Network: Short Review

Neural Network: Short Review

Inference: given values for all parameters predict output (probability)

(a.k.a. **Forward Pass**)

Learning: given data optimize parameters using gradient -based optimization

(a.k.a. **Backwards Pass**)

Neural Network: Short Review

Inference: given values for all parameters predict output (probability)

(a.k.a. **Forward Pass**)

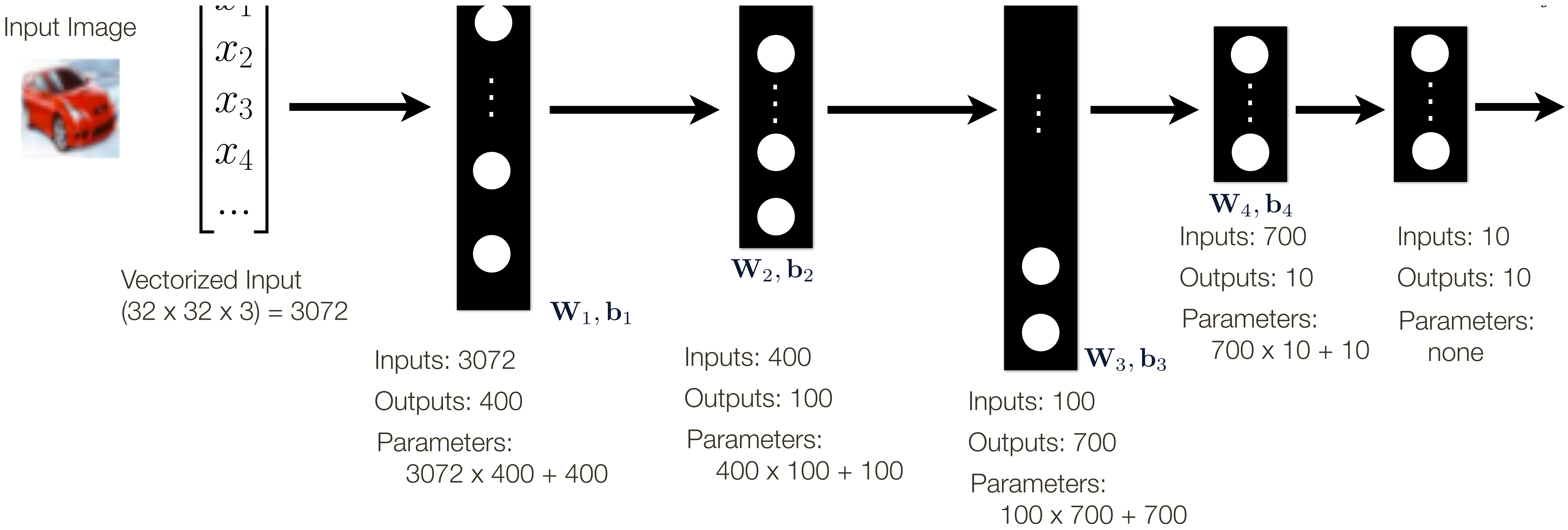
Learning: given data optimize parameters using gradient -based optimization

(a.k.a. **Backwards Pass**)

Neural Network: Short Review

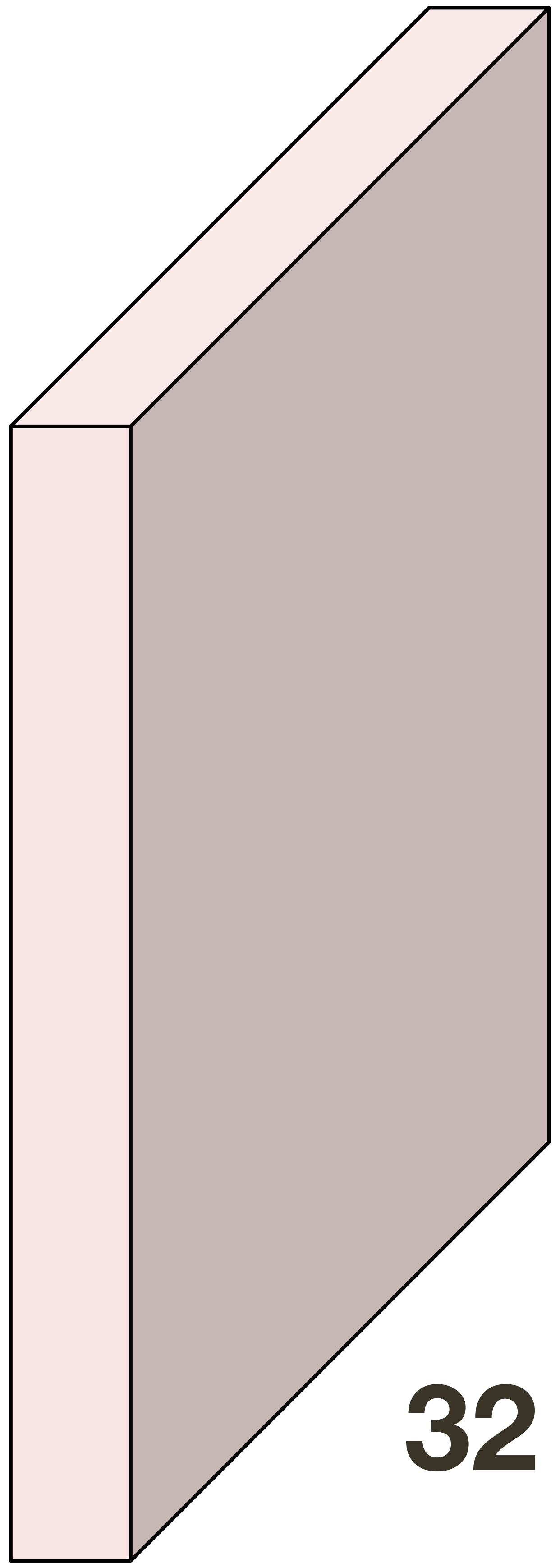
Neural Network: Short Review

This simple neural network has nearly 1.35 million parameters



Convolutional Layer

32 x 32 x 3 **image**



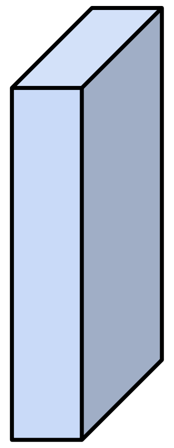
32 height

32 width

3 depth

Filters always extend the full depth of the input volume

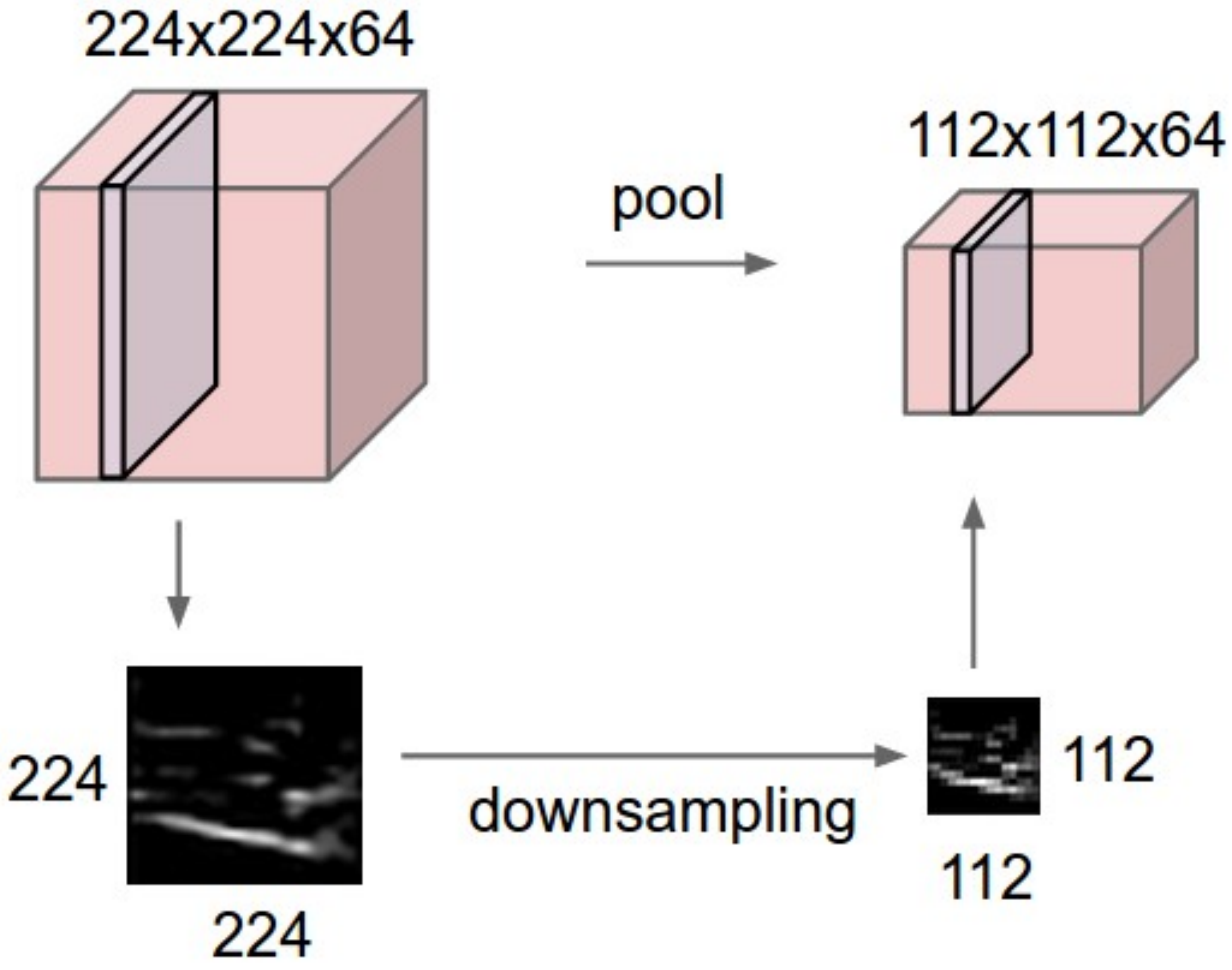
5 x 5 x 3 **filter**



Convolve the filter with the image (i.e., “slide over the image spatially, computing dot products”)

Pooling Layer

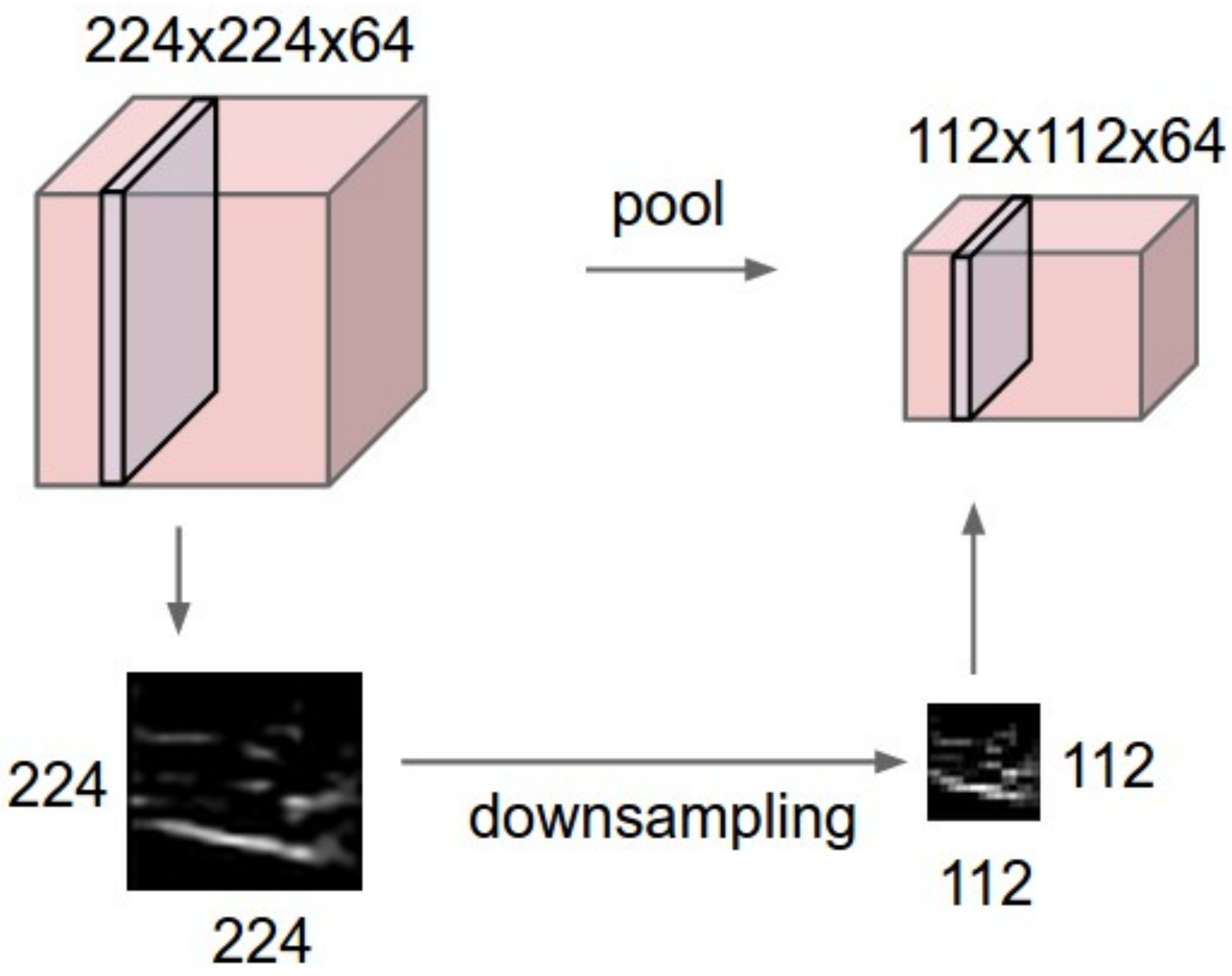
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Pooling Layer

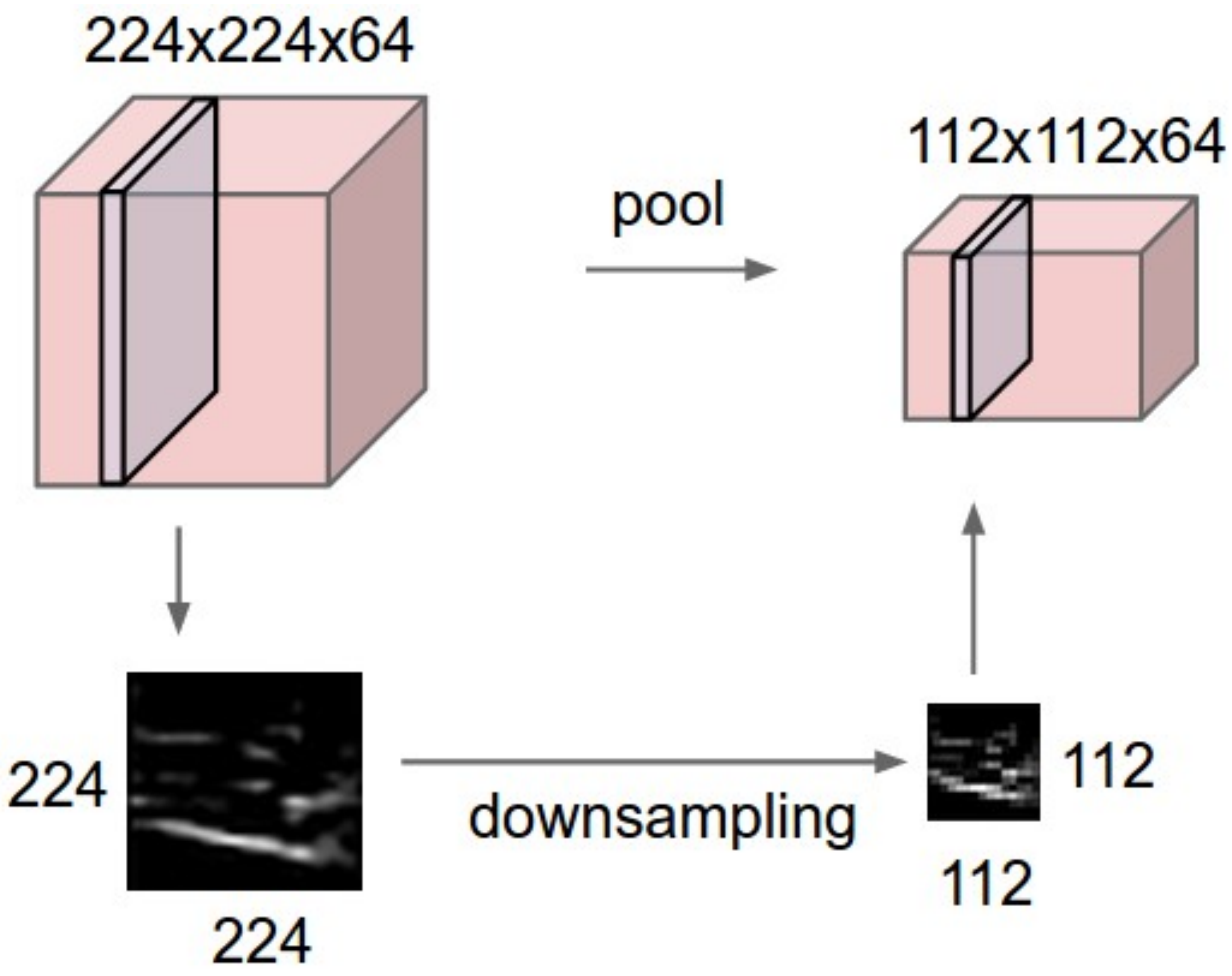
- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently



How many **parameters**?

Pooling Layer

- Makes representation smaller, more manageable and spatially invariant
- Operates over each activation map independently

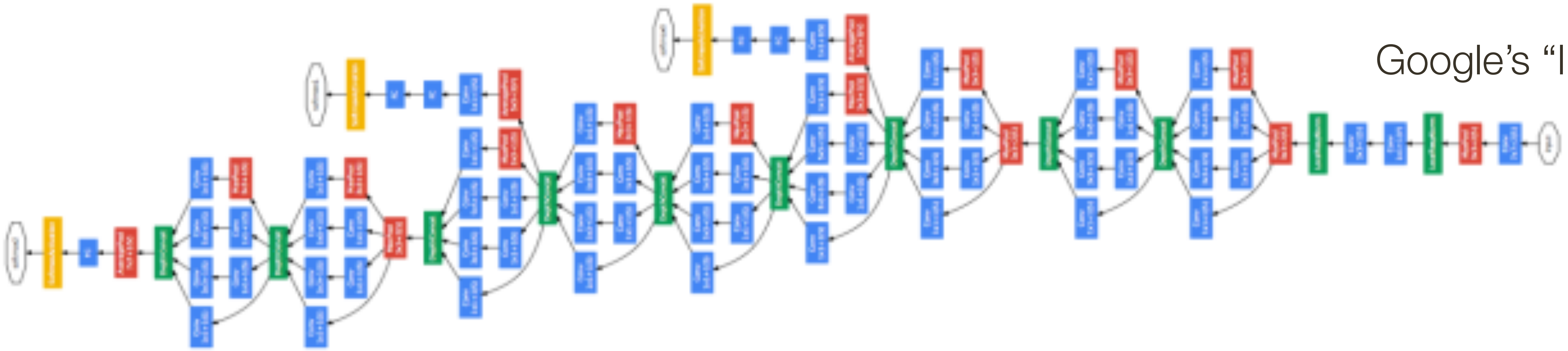


How many **parameters**?

None!

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

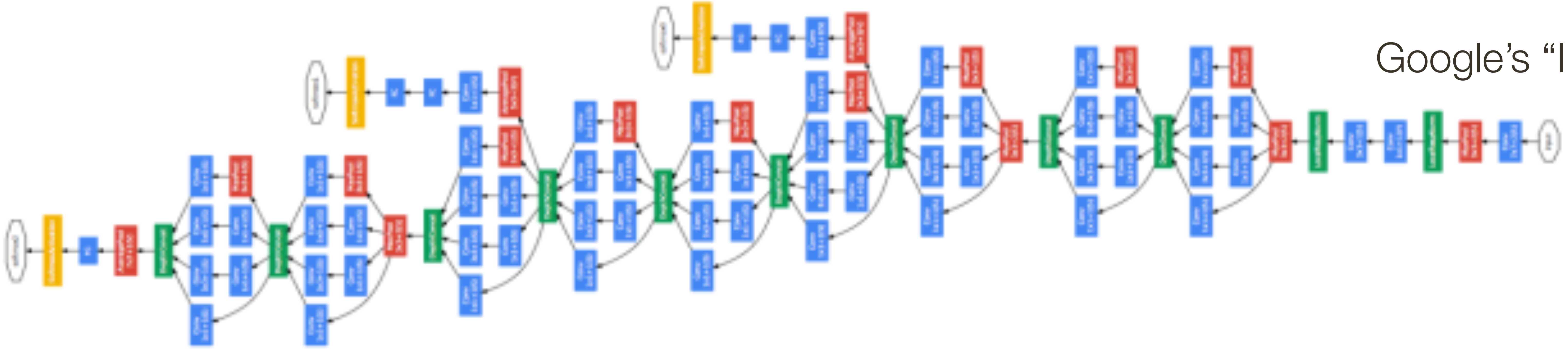
requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

- **Hyper-parameters:** parameters, including for optimization, that are not optimized directly as part of training (*e.g.*, `learning rate`, `batch size`, `drop-out rate`)

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

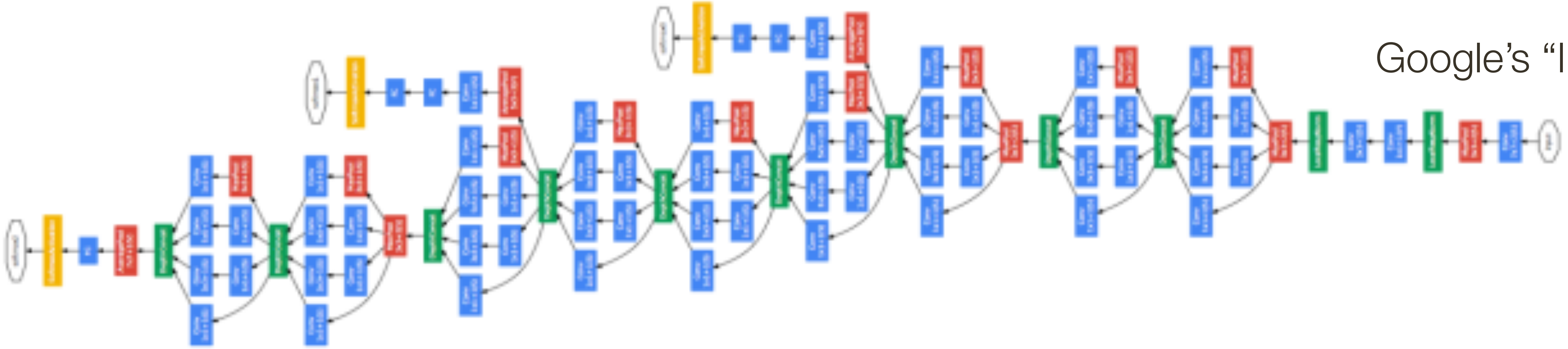
- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

- **Hyper-parameters:** parameters, including for optimization, that are not optimized directly as part of training (*e.g.*, `learning rate`, `batch size`, `drop-out rate`)

grid search

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

Specification of neural architecture will define a **computational** graph.

R-CNN

[Girshick et al, CVPR 2014]

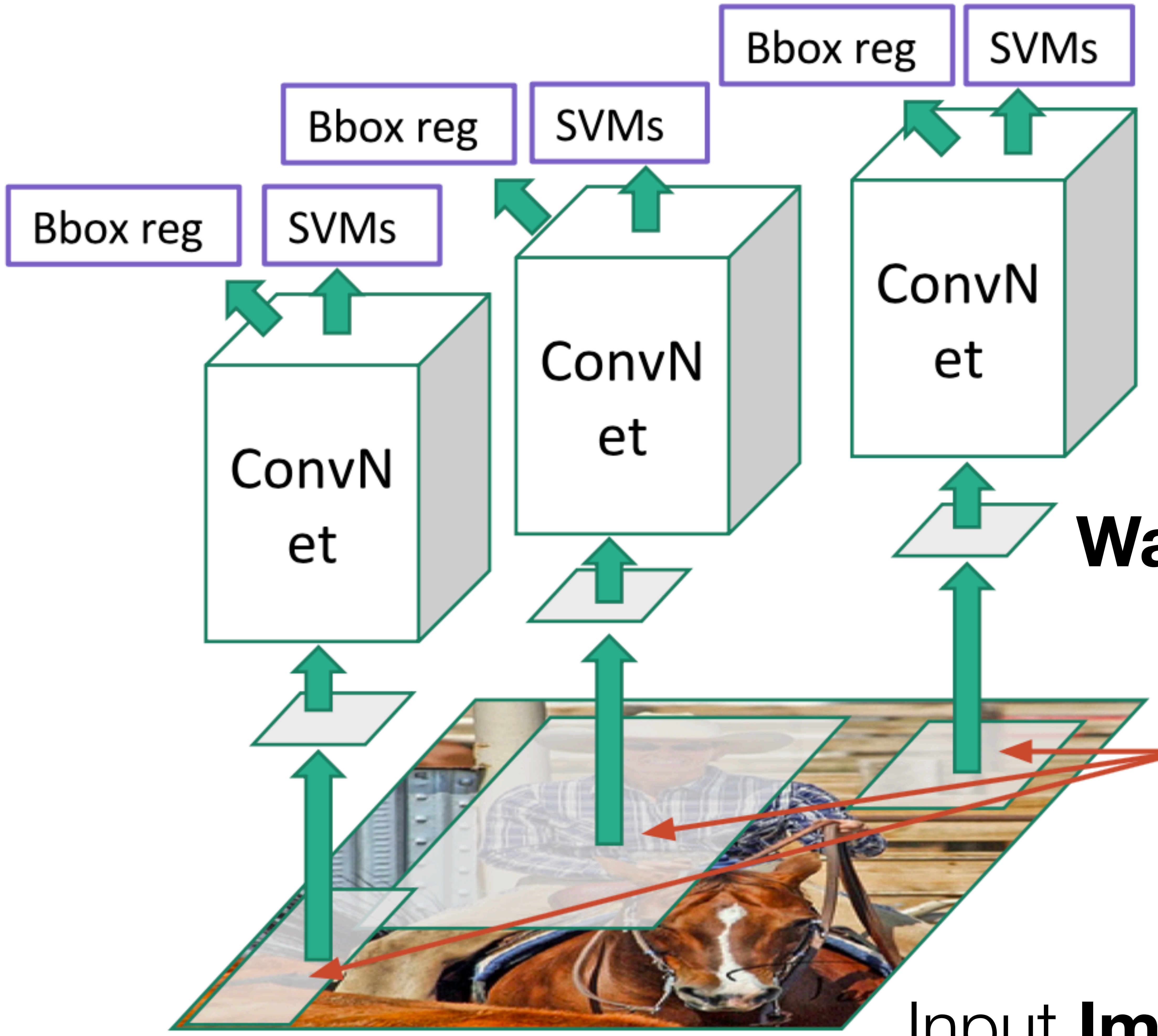
Linear Regression for bounding box offsets

Classify regions with SVM

Forward each region through a **CNN**

Warped image regions

Regions of Interest from a proposal method (~2k)

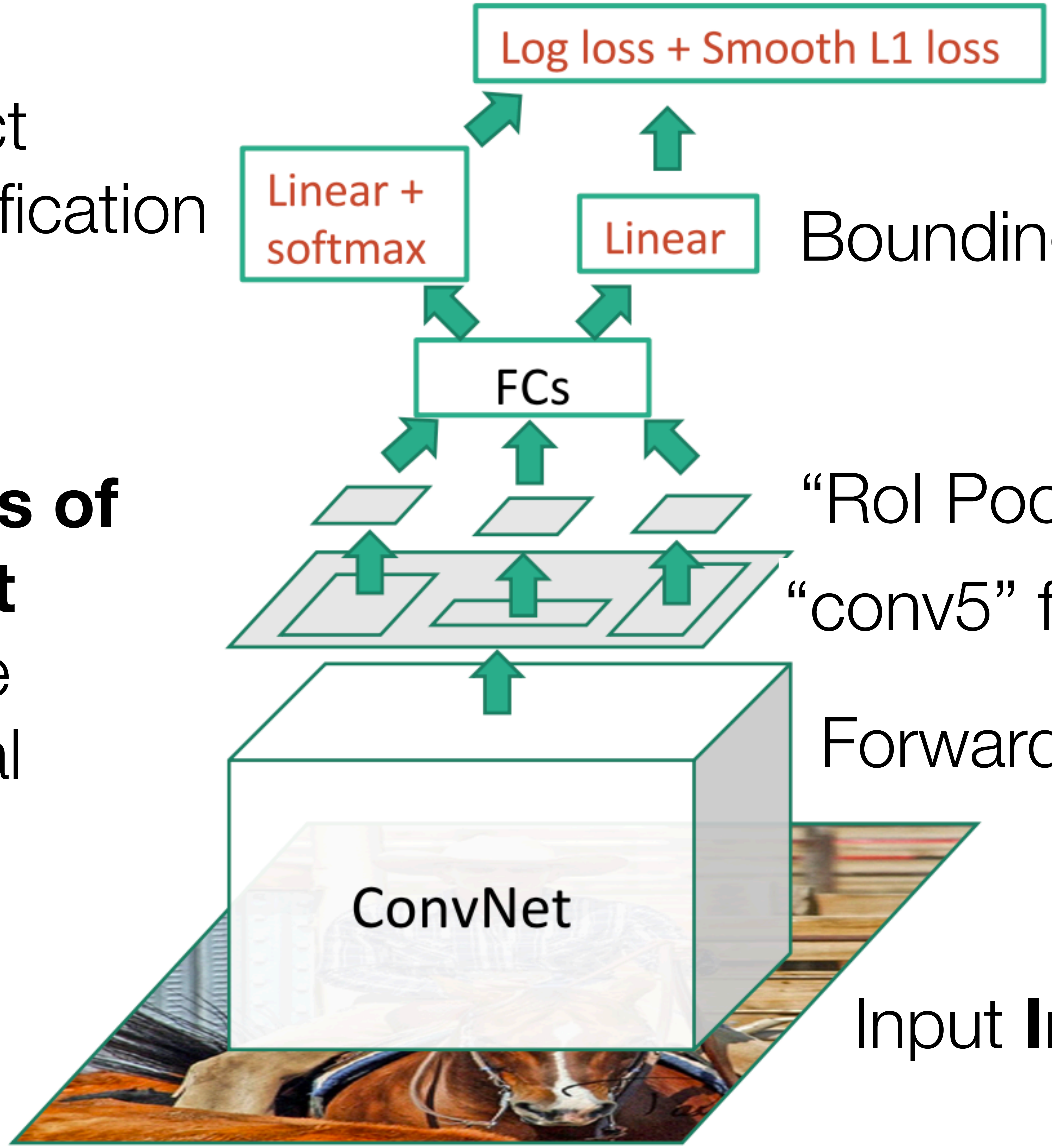


Input **Image**

* image from Ross Girshick

Fast **R-CNN**

[Girshick et al, ICCV 2015]



Multi-task loss

Object classification

Bounding box regression

"RoI Pooling" layer

"conv5" feature map

Regions of Interest from the proposal method

Forward prop the **whole image** through CNN

Input **Image**

* image from Ross Girshick

Course Review: Colour

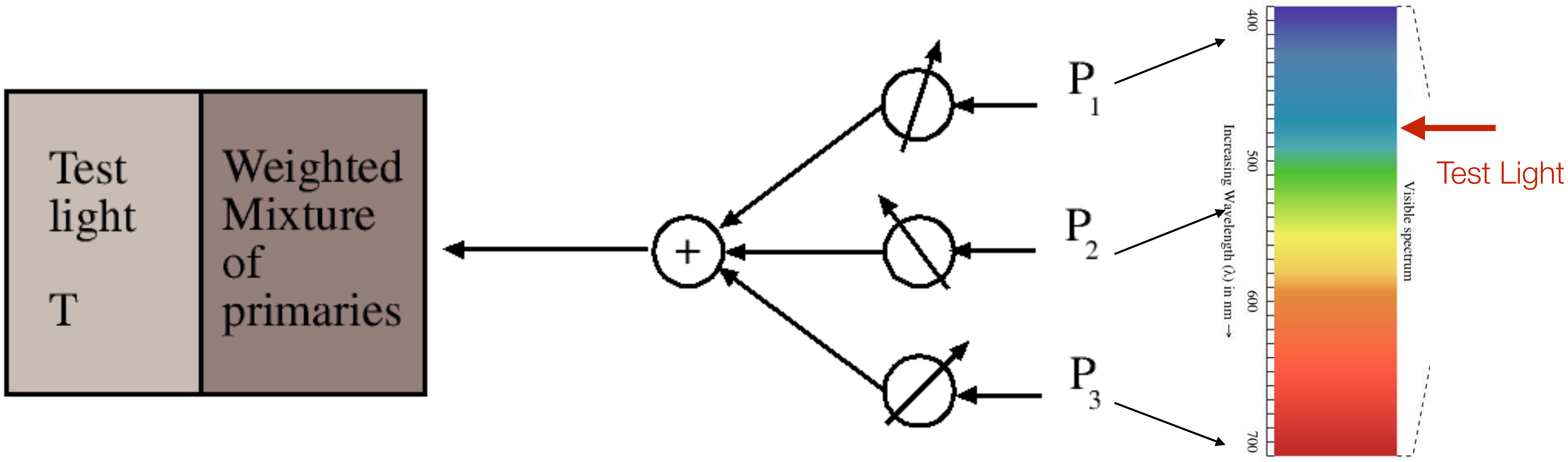
Human colour perception

RGB and **CIE XYZ** colour spaces

Uniform colour space

HSV colour space

Color Matching Experiments

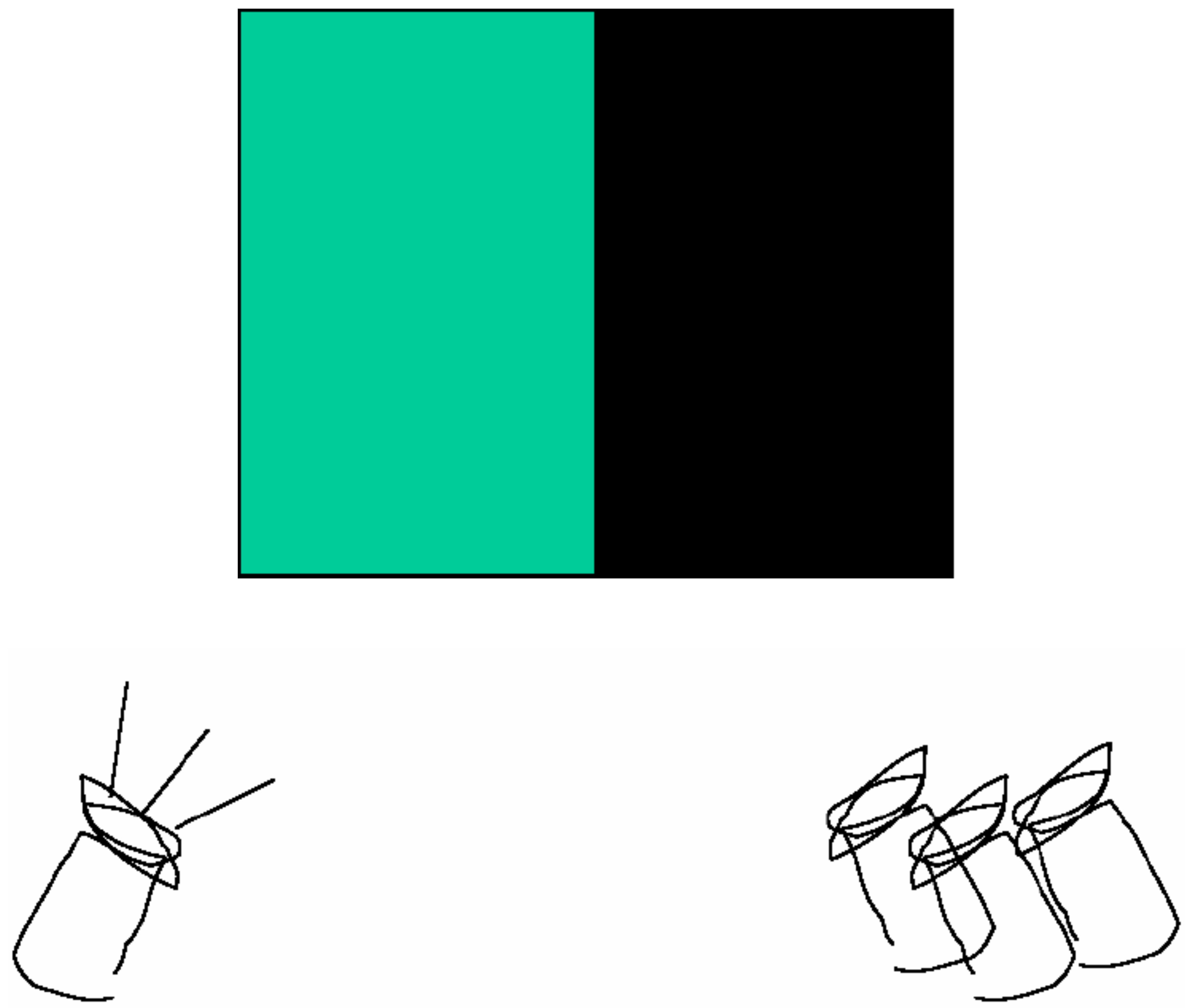


Forsyth & Ponce (2nd ed.) Figure 3.2

Show a split field to subjects. One side shows the light whose colour one wants to match. The other a weighted mixture of three primaries (fixed lights)

$$T = w_1 P_1 + w_2 P_2 + w_3 P_3$$

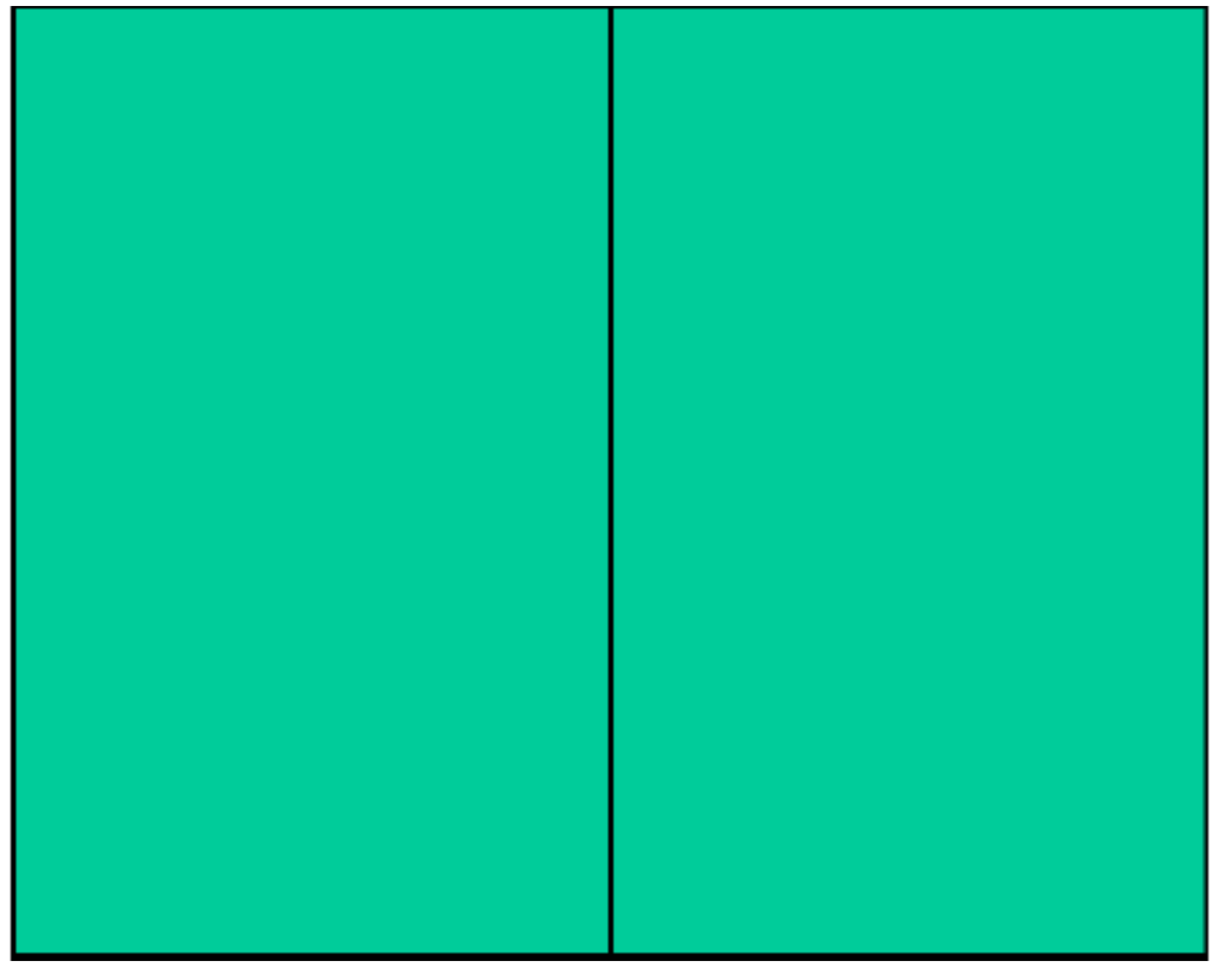
Example 1: Color Matching Experiment

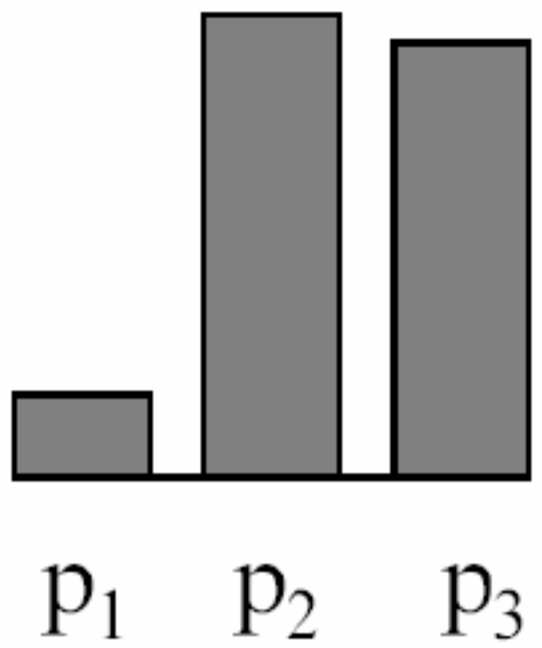


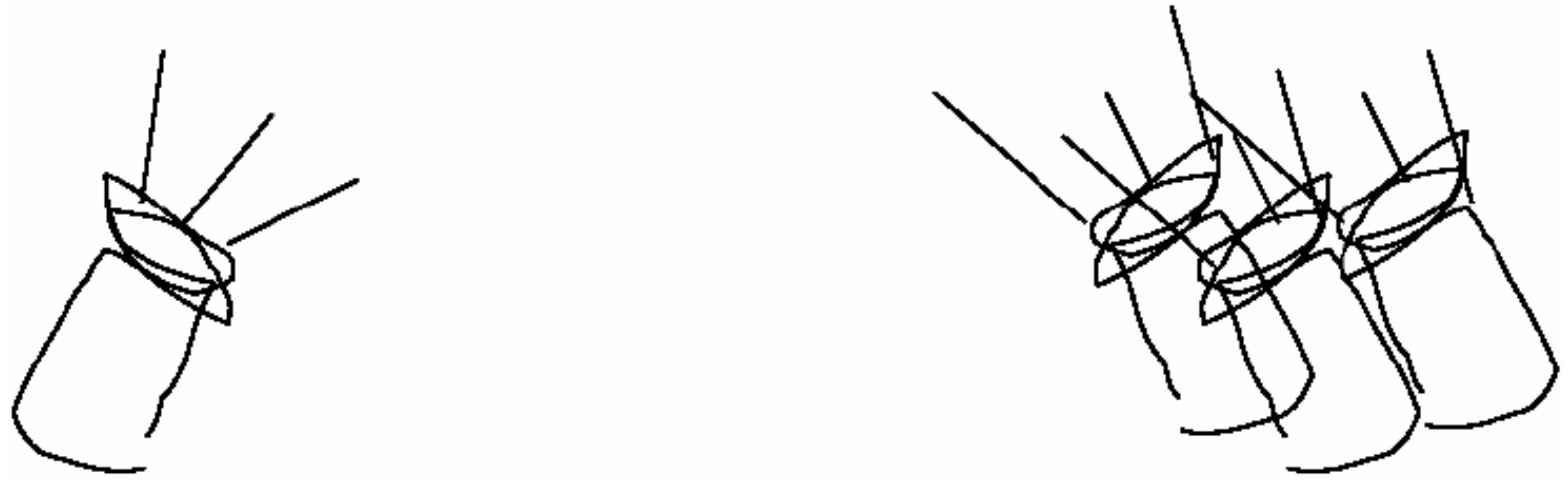
knobs here

Example Credit: Bill Freeman

Example 1: Color Matching Experiment



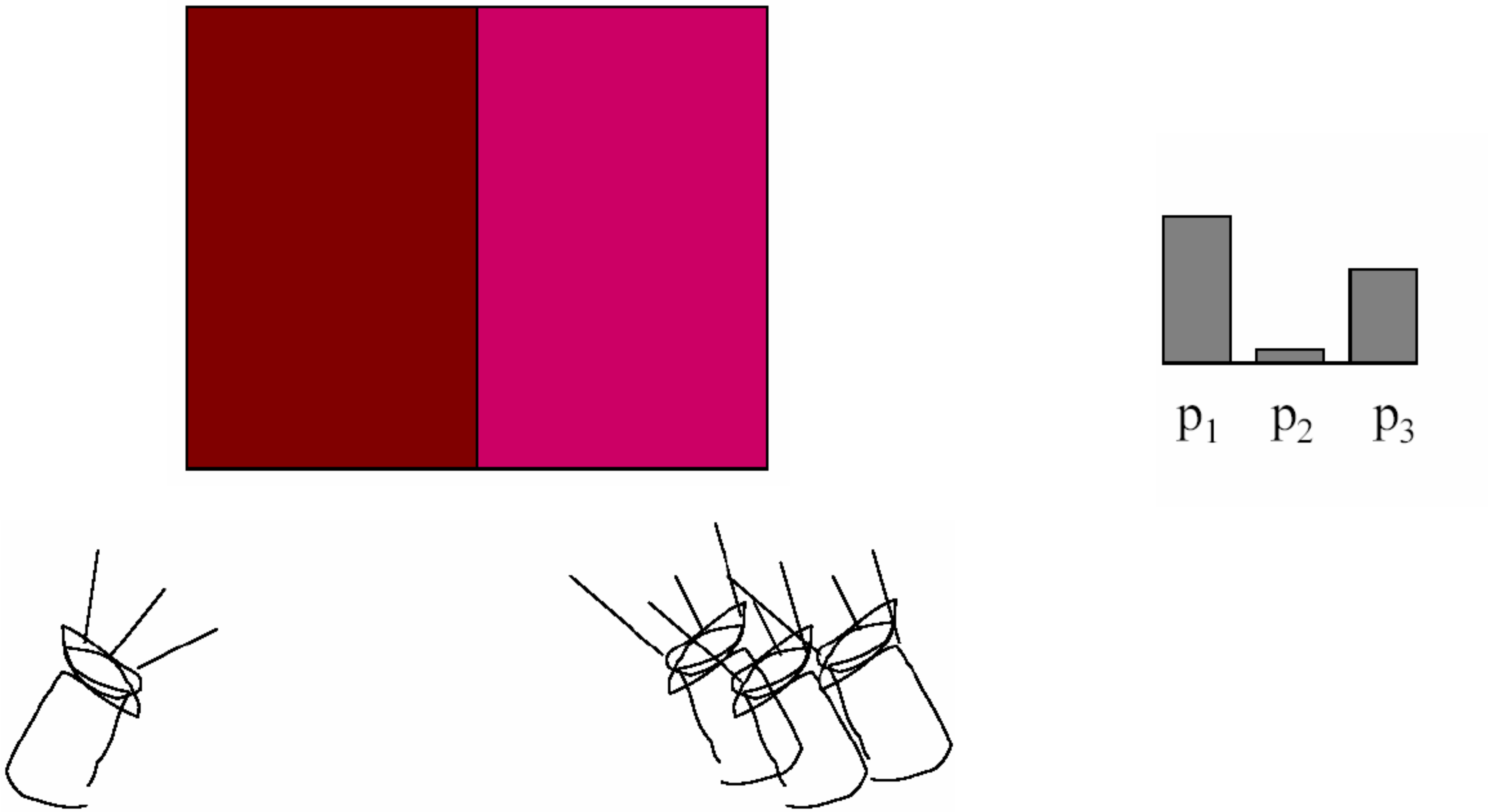




knobs here

Example Credit: Bill Freeman

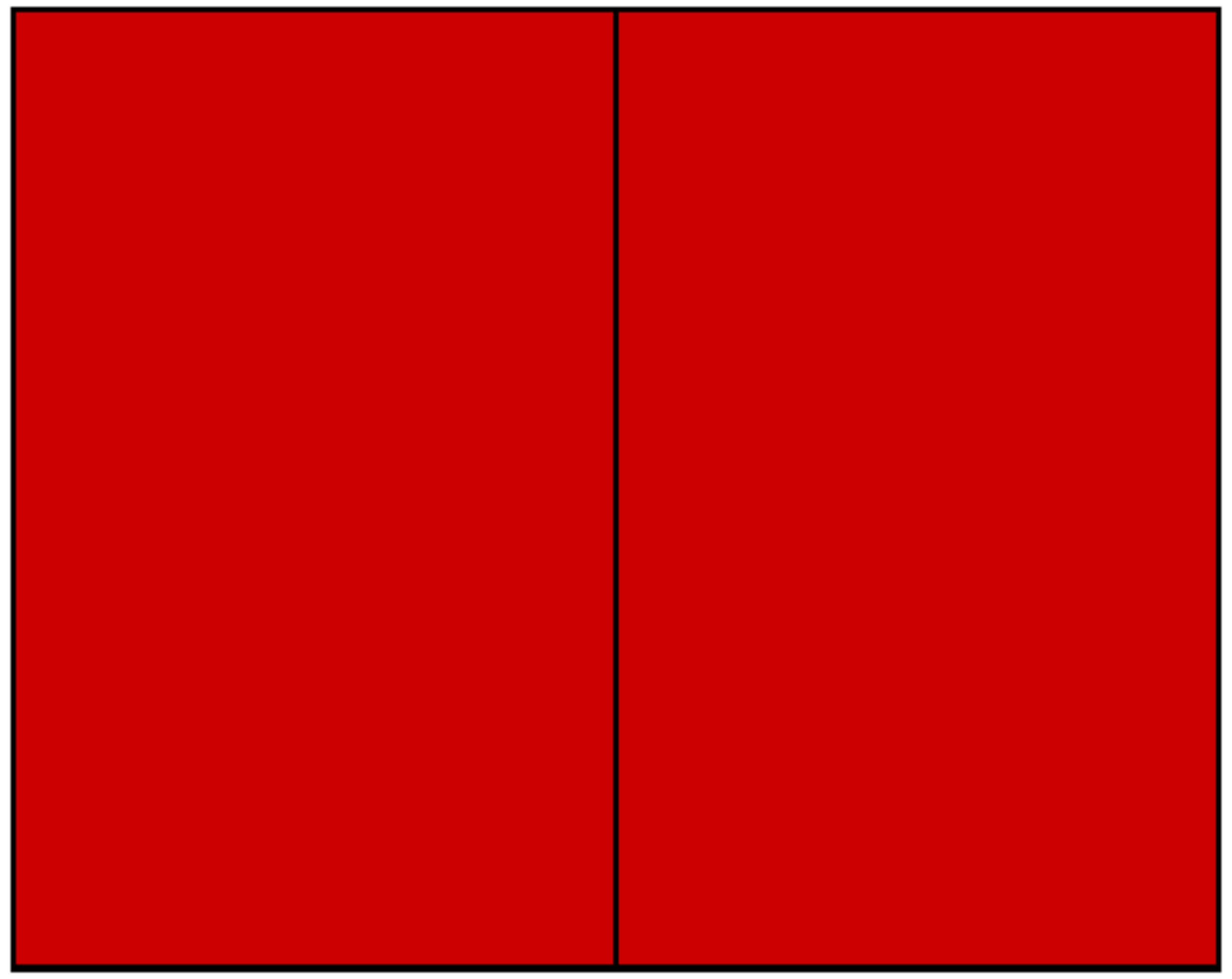
Example 2: Color Matching Experiment



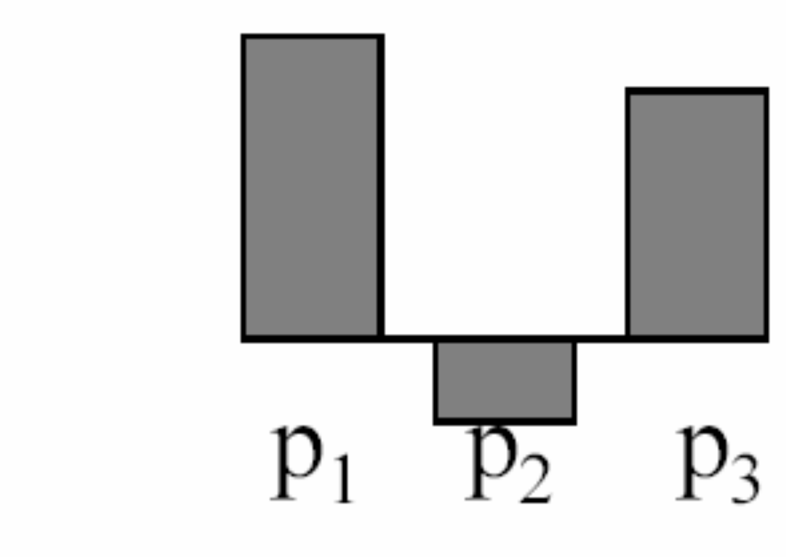
Example Credit: Bill Freeman

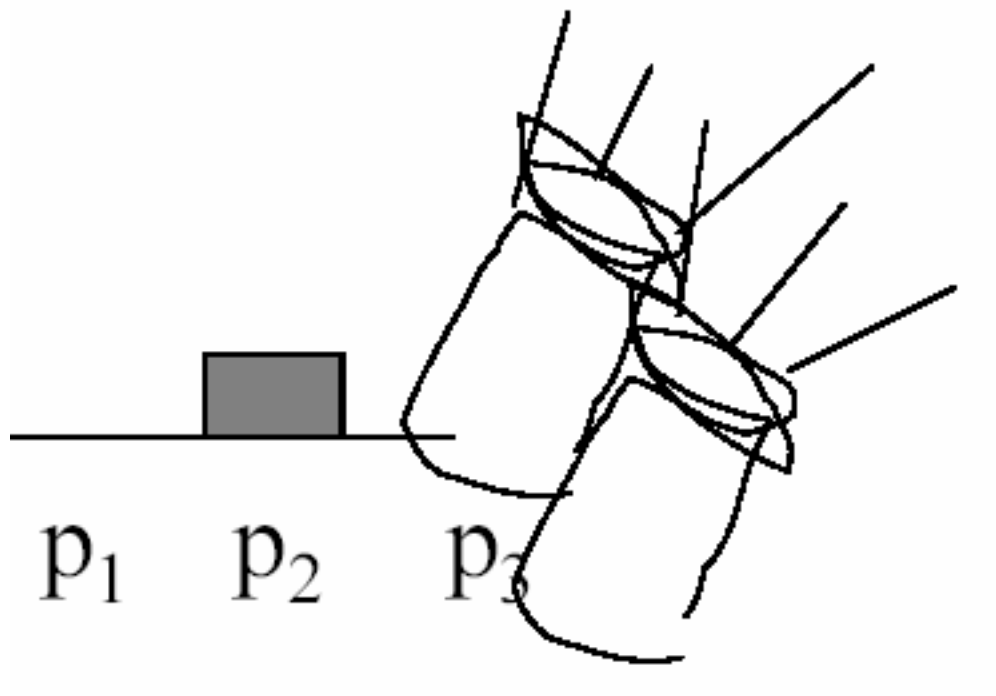
Example 2: Color Matching Experiment

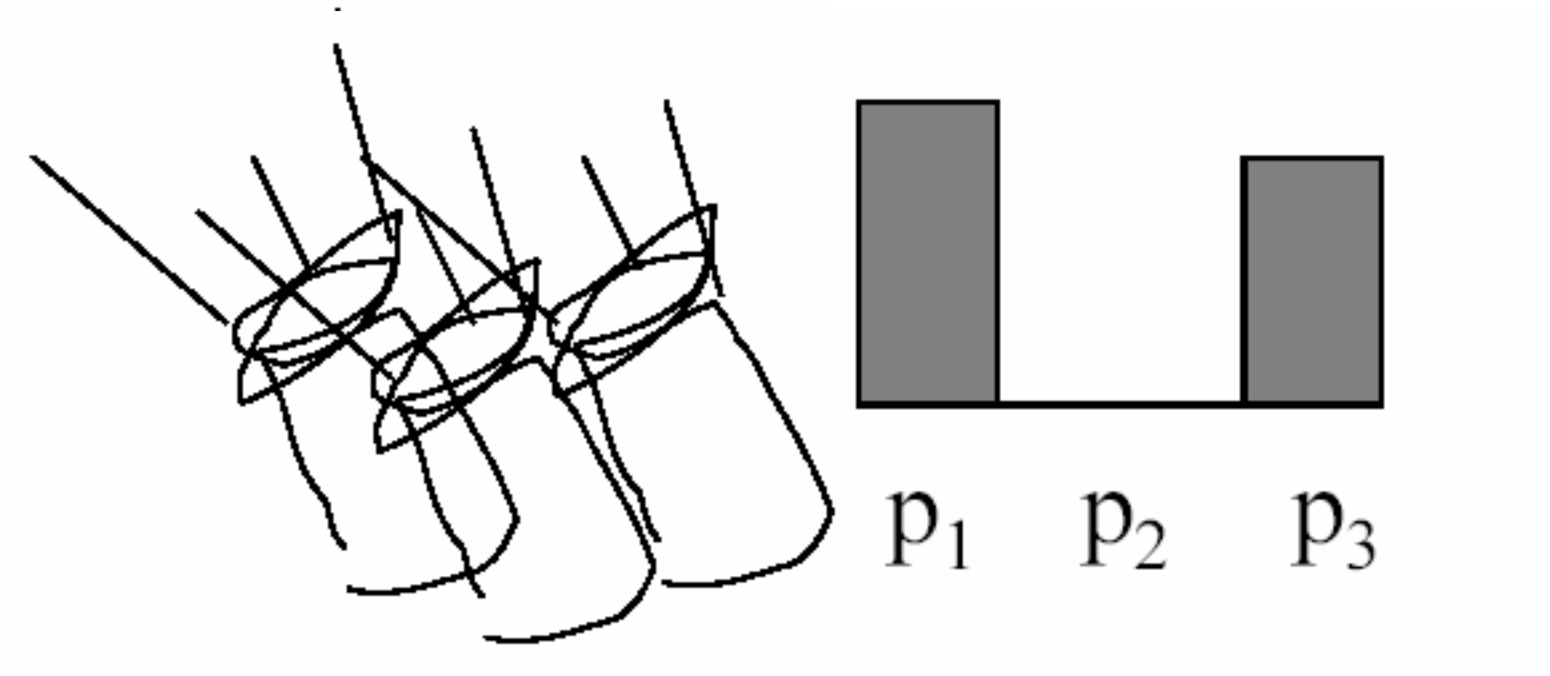
We say a “negative” amount of P_2 was needed to make a match , because we added it to the test color side



The primary color amount needed to match:



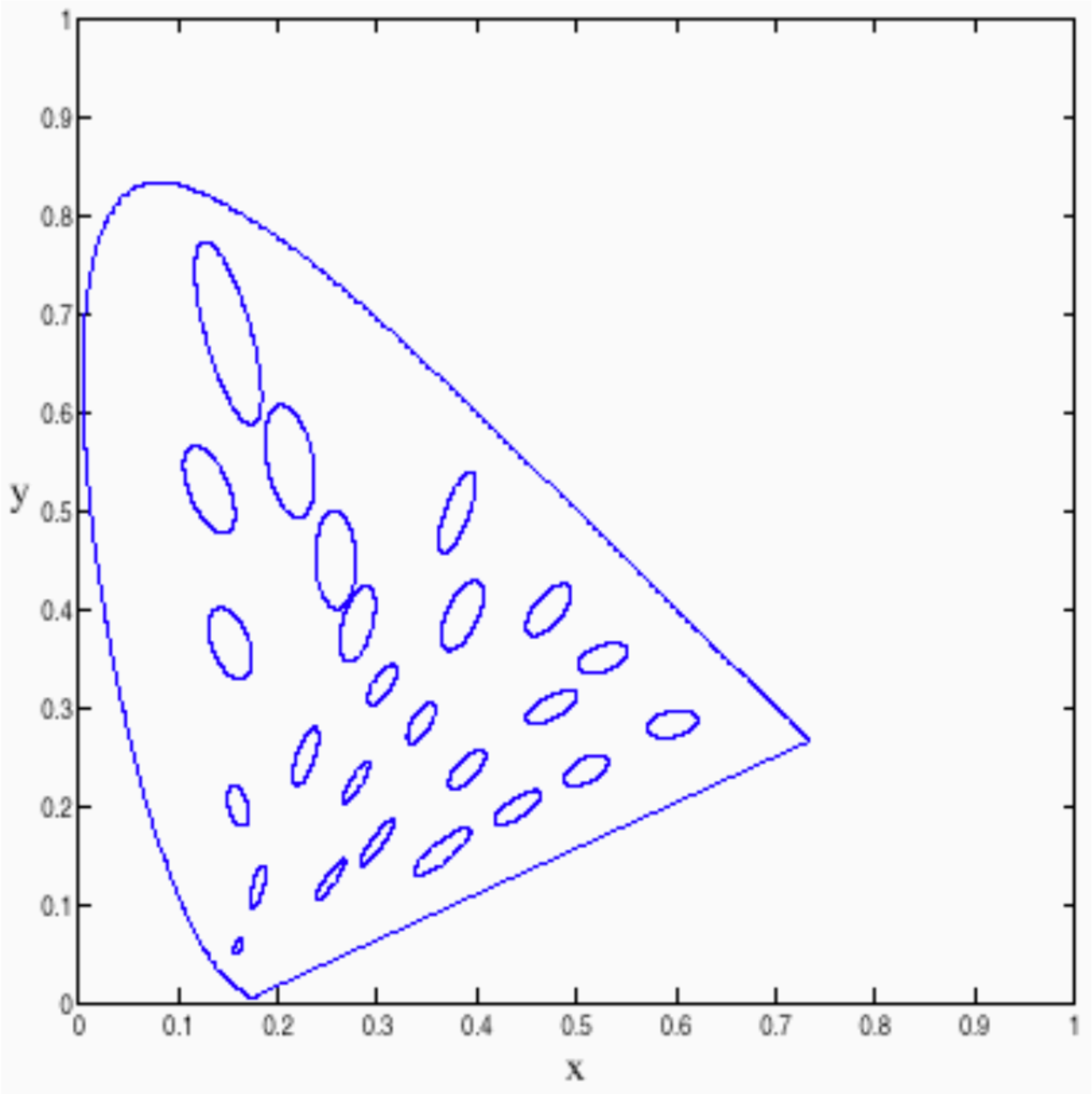




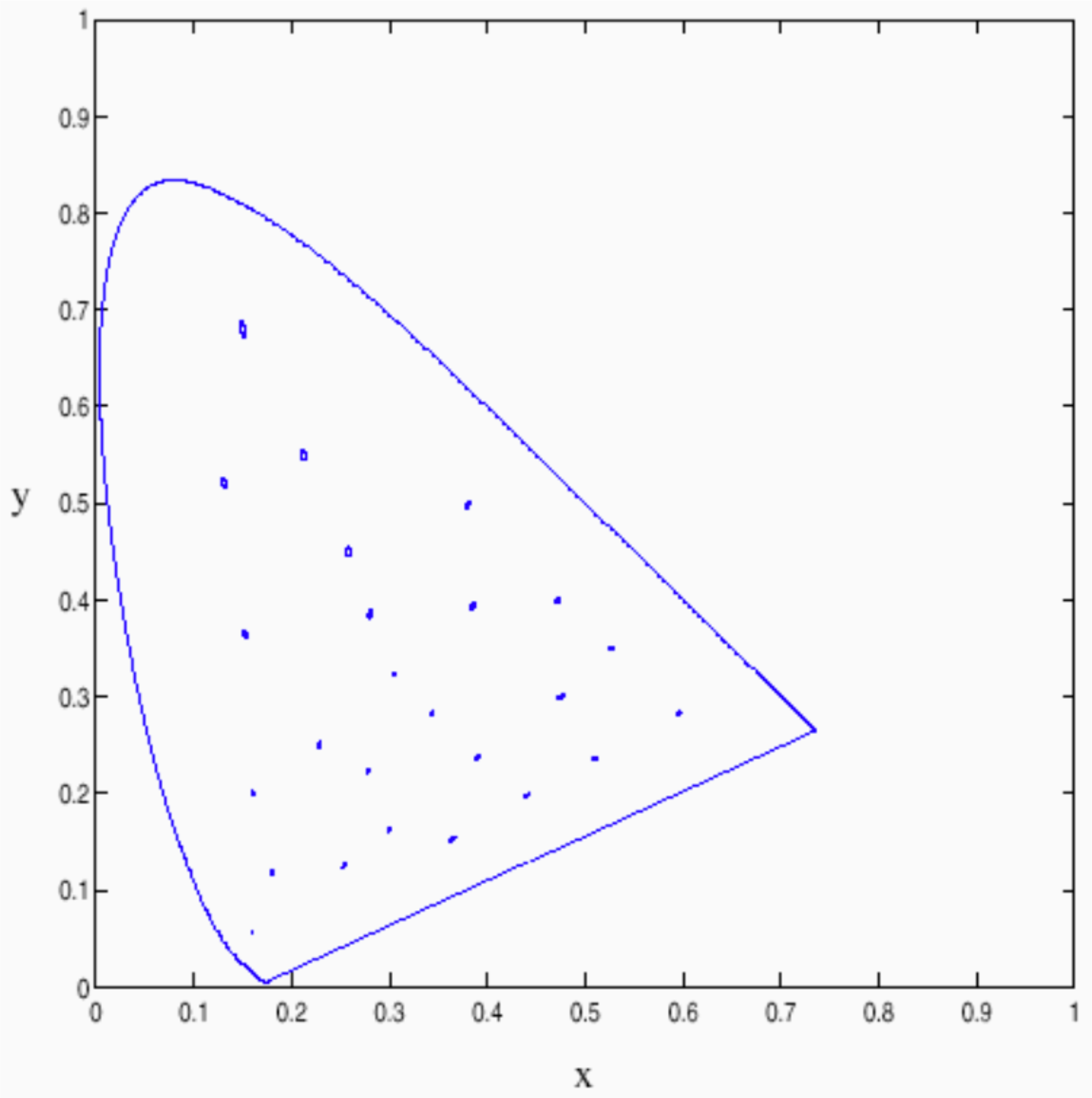
Example Credit: Bill Freeman

Uniform Colour Spaces

McAdam Ellipses: Each ellipse shows colours perceived to be the same



10 times actual size



Actual Size

Forsyth & Ponce (2nd ed.) Figure 3.14

Uniform Colour Spaces

McAdam ellipses demonstrate that differences in x , y are a poor guide to differences in perceived colour

A **uniform colour space** is one in which differences in coordinates are a good guide to differences in perceived colour

— example: CIE LAB

Hope you enjoyed the **course**!