Lecture 14

Recognition and kNN plus Dimensionality Reduction

Linda Shapiro Nov 13, 2025

Administrative

A3

- Due Nov 12

A4 coming out

- Due Nov 25

Administrative

Recitation this friday

- Optical Flow

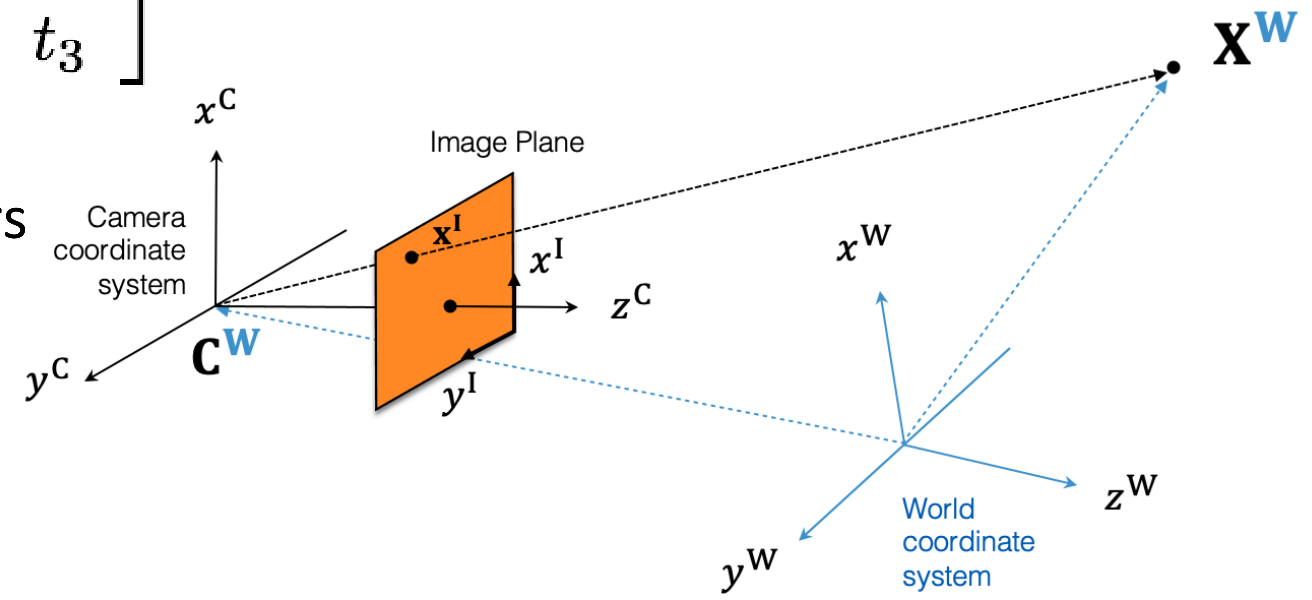
So far: General pinhole camera matrix

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$$ where $$\mathbf{t} = -\mathbf{RC}$$

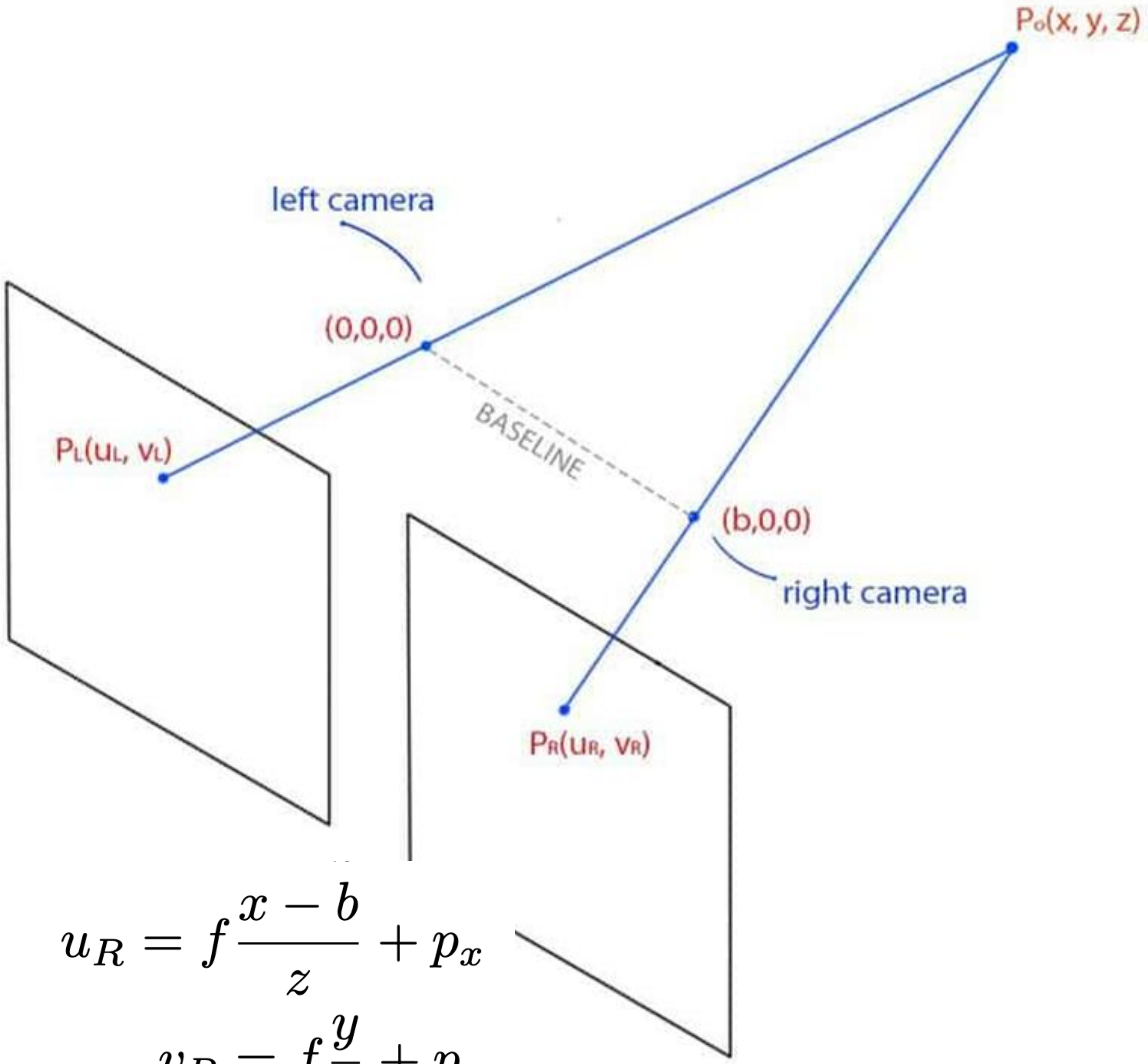
$$\mathbf{P} = \left[\begin{array}{ccc} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{array}\right] \left[\begin{array}{ccc:c} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{array}\right]$$

intrinsic parameters

extrinsic parameters



So far: Estimating depth



$$u_L = f\frac{x}{z} + p_x$$

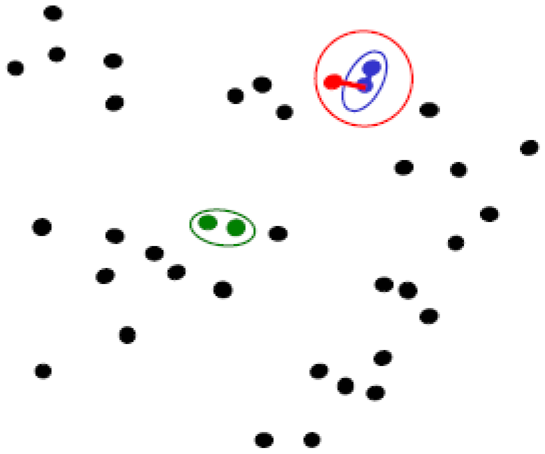
$$v_L = f\frac{y}{z} + p_y$$

$$u_R = f\frac{x-b}{z} + p_x$$

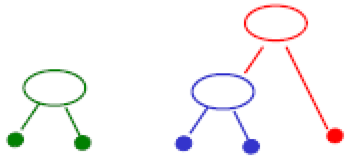
$$v_R = f\frac{y}{z} + p_y$$

K = $\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$

So far: Agglomerative clustering



1. Say "Every point is its own cluster"
2. Find "most similar" pair of clusters
3. Merge it into a parent cluster
4. Repeat



Slide credit: Andrew Moore

So far, K-means clustering

1. Initialize ($t = 0$): cluster centers $c_1, ..., c_K$

2. Compute δ^t : assign each point to the closest center
   - δ^t denotes the set of assignment for each v_j to cluster c_i at iteration *t*

$$\delta^t = \arg\min_{\delta} \frac{1}{N} \sum_{j}^{N} \sum_{i}^{K} \delta_{ij}^{t-1} (c_i^{t-1} - v_j)^2$$

3. Computer c^t : update cluster centers as the mean of the points

$$c_i^t = 1/N \sum_j \delta^t_{ij} v_j$$

Update $t = t + 1$, Repeat Step 2-3 till stopped

So far: Mean-Shift Algorithm

1. Represent each pixel *i* using some feature vector v_i
2. Generate a window **W** as a random pixel feature v_w
3. Identify all the pixels within a radius *r* of v_w
4. Calculate the mean ("center of gravity") amongst the neighbors of **W**
5. Translate the window **W** to the mean feature location
6. Repeat Step 2 until convergence

Today’s agenda

- Introduction to recognition
- A object recognition pipeline
- Choosing the right features
- A training algorithm: KNN
- Testing an algorithm
- Challenges with kNN
- Dimensionality reduction

Today's agenda

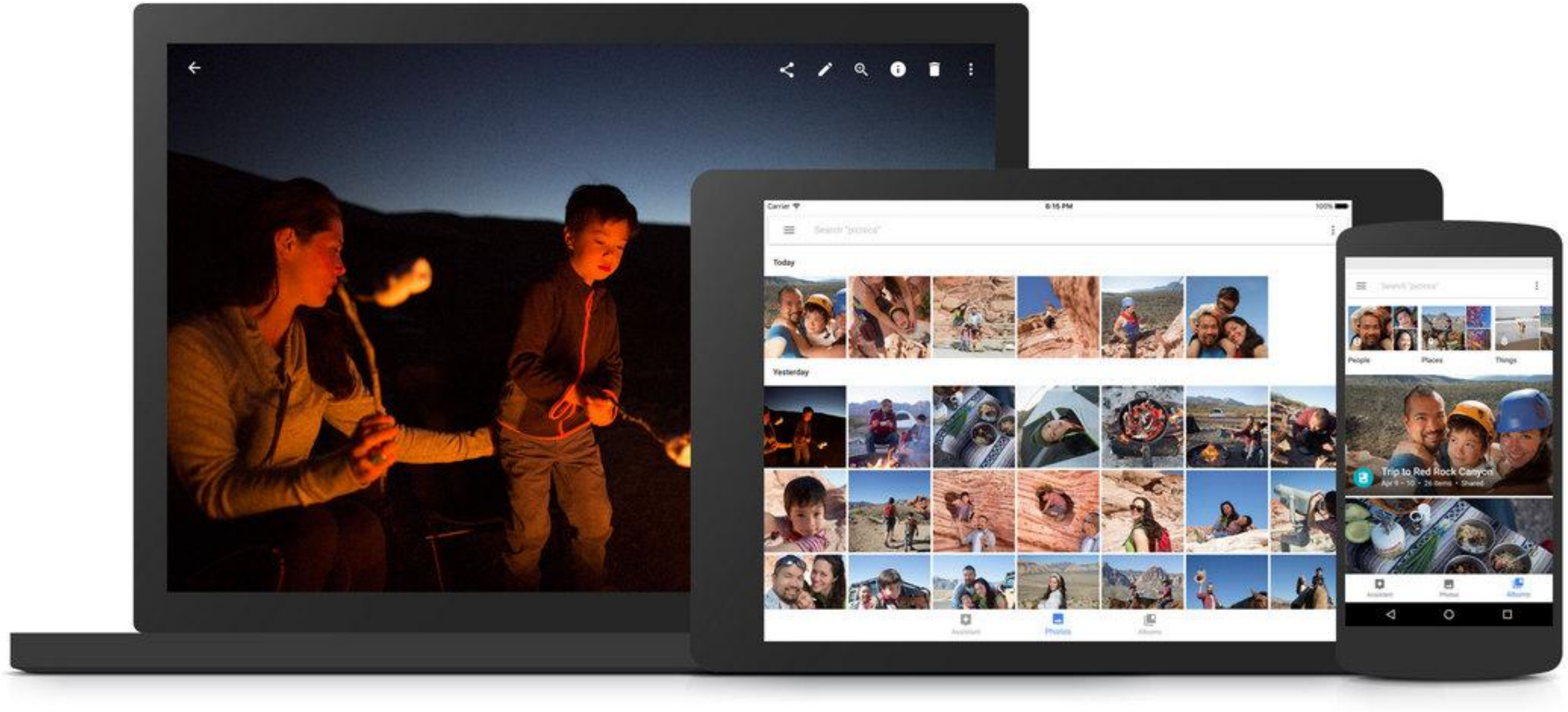
- Introduction to recognition
- A object recognition pipeline
- Choosing the right features
- A training algorithm: KNN
- Testing an algorithm
- Challenges with kNN
- Dimensionality reduction

What do we mean by recognition?

Classification: Does this image contain a building? [yes/no]

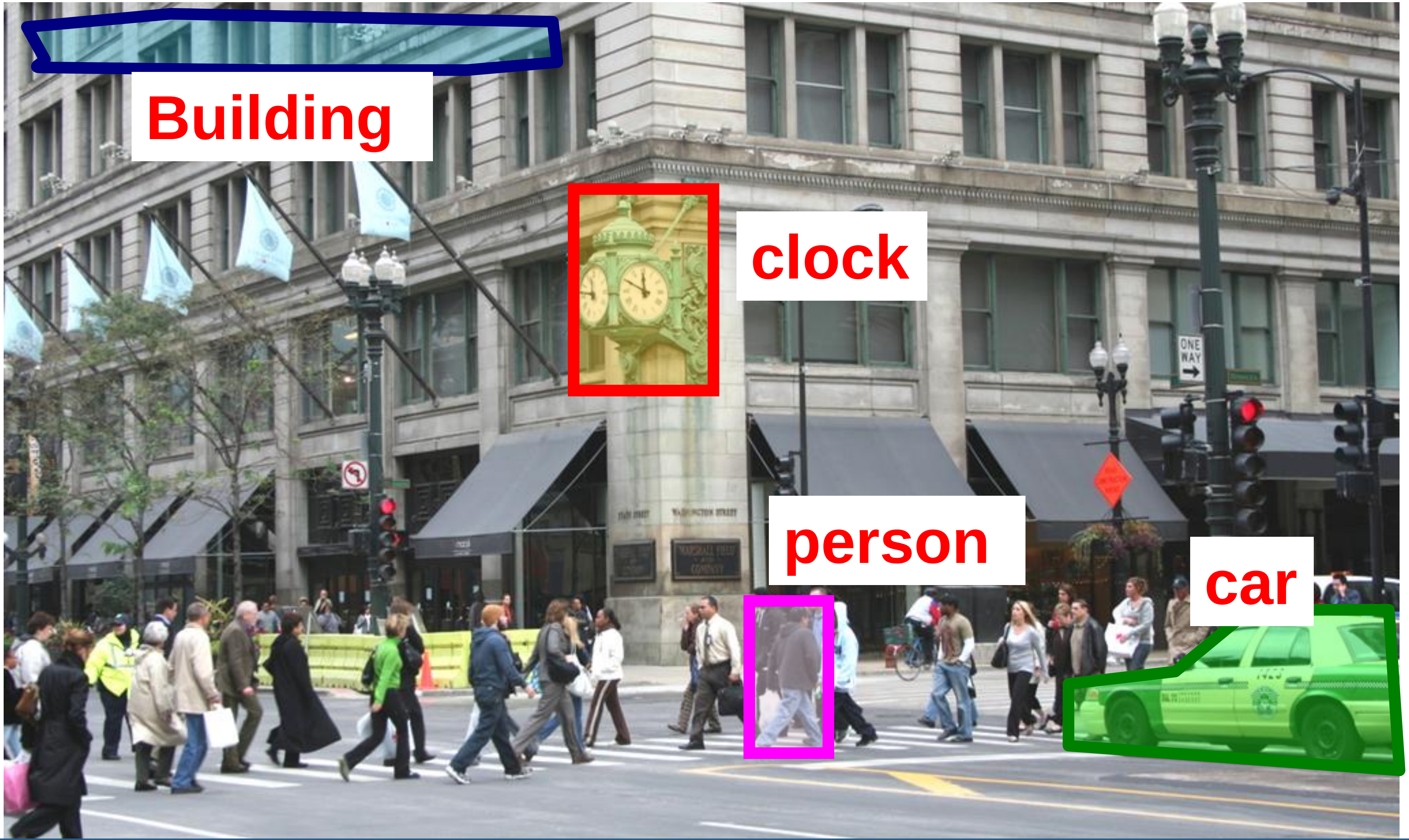
Classification: Is this a beach?

Applications: Image Search & Organizing photo collections



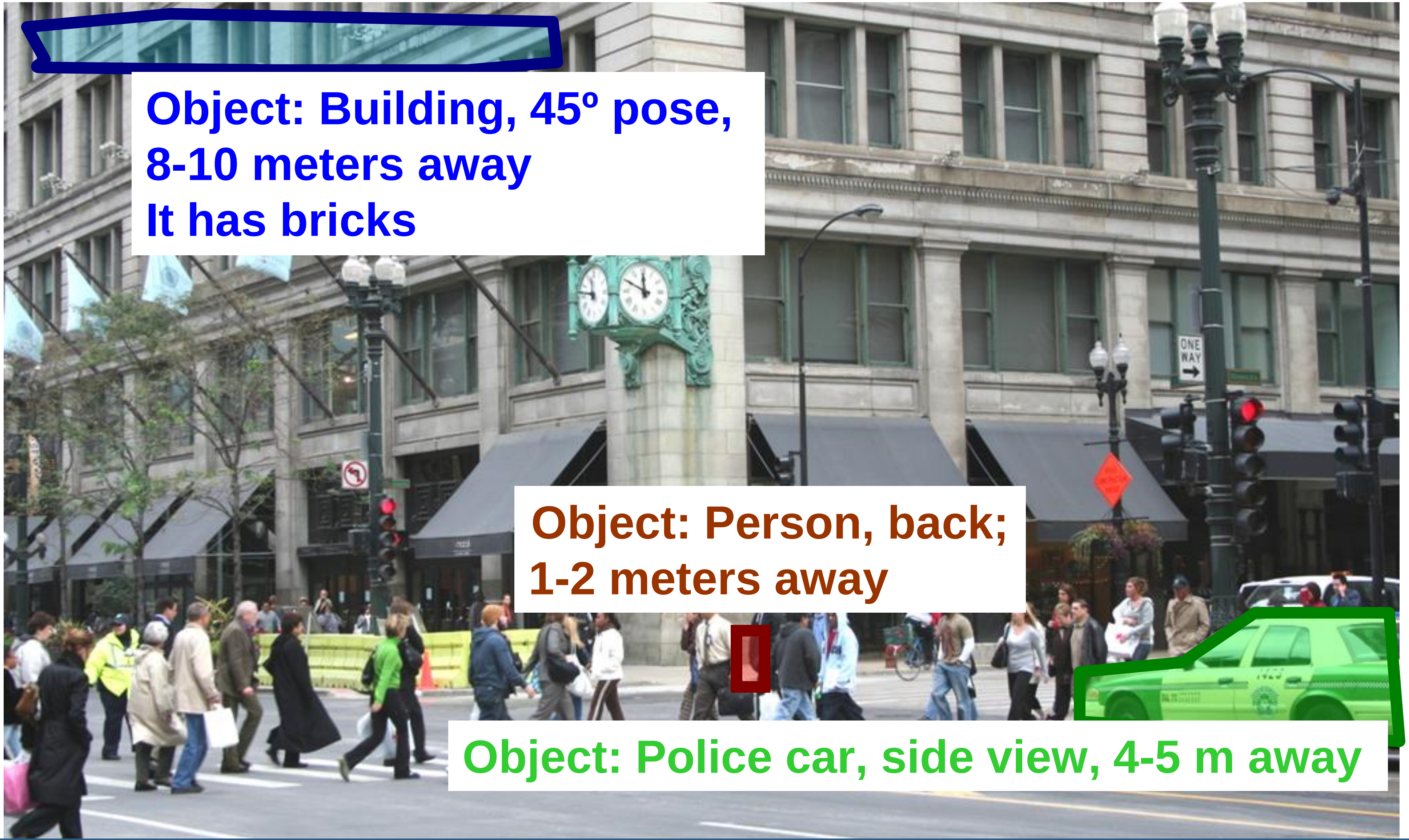
Detection: Does this image contain a car? [where?]

Detection: Which object does this image contain? [where?]



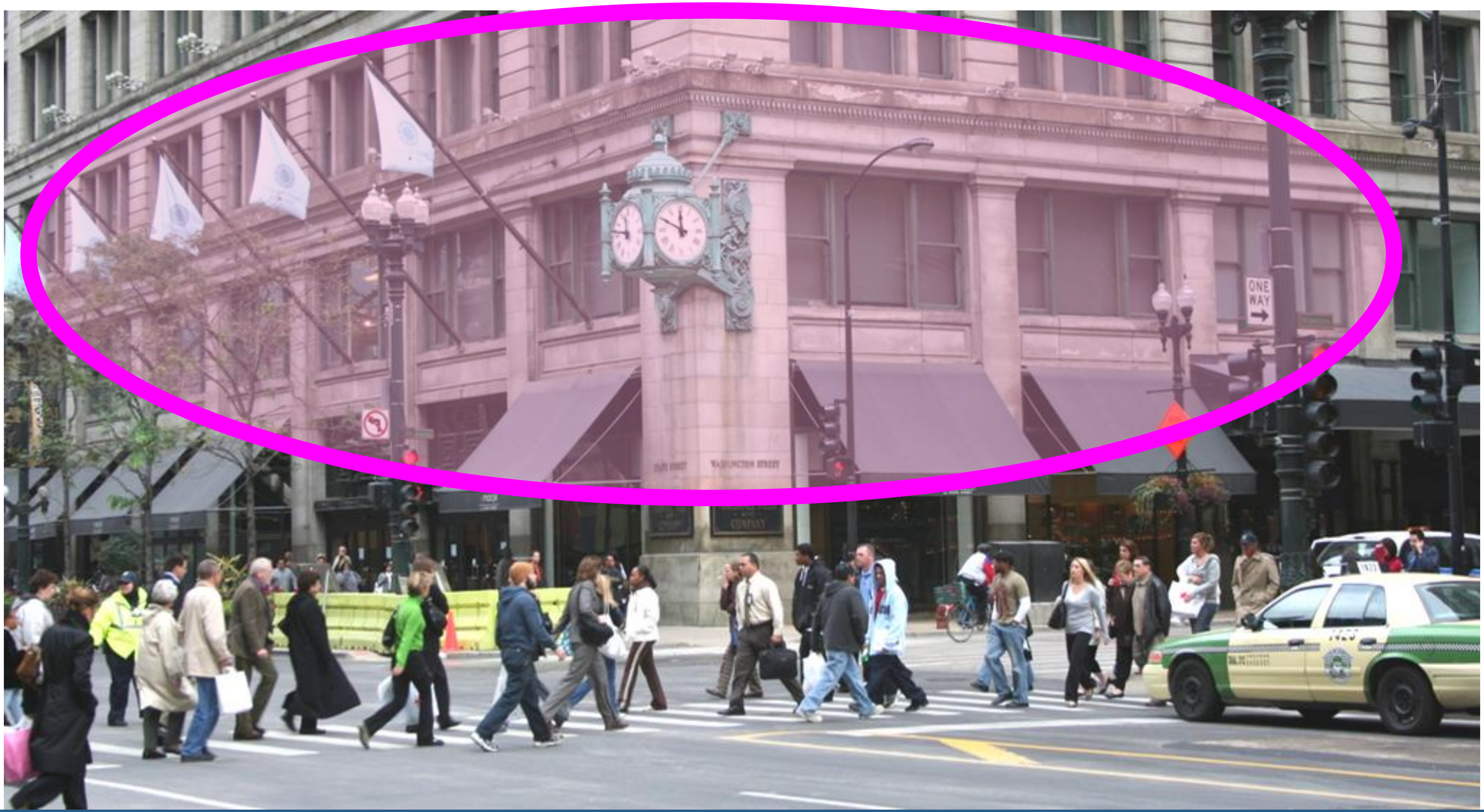
Detection: Accurate localization (segmentation)

Detection: Estimating object semantic & geometric attributes



Levels of recognition: Category-level vs instance-level

Does this image contain the (downtown) Chicago Macy’s building?



Categorization vs Single instance recognition

We have seen a form of single instance categorization already: **Where is the crunchy nut?**

Applications of computer vision

Recognizing landmarks in mobile devices

Activity recognition: What are these people doing?

Visual Recognition

- Design algorithms that can:
  - Classify images or videos
  - Detect and localize objects
  - Estimate semantic and geometrical attributes
  - Classify human activities and events

Why is this challenging?

How many object categories are there?

Challenges: viewpoint variation

Michelangelo 1475-1564

Challenges: illumination

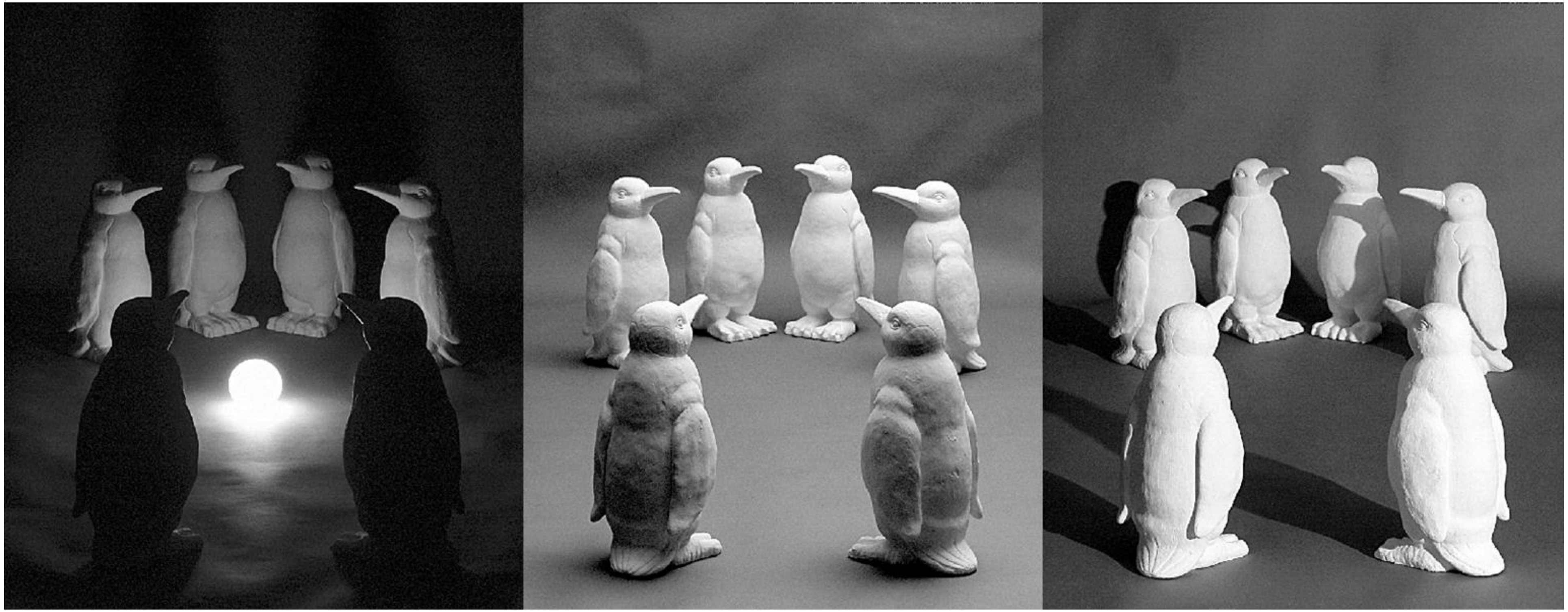
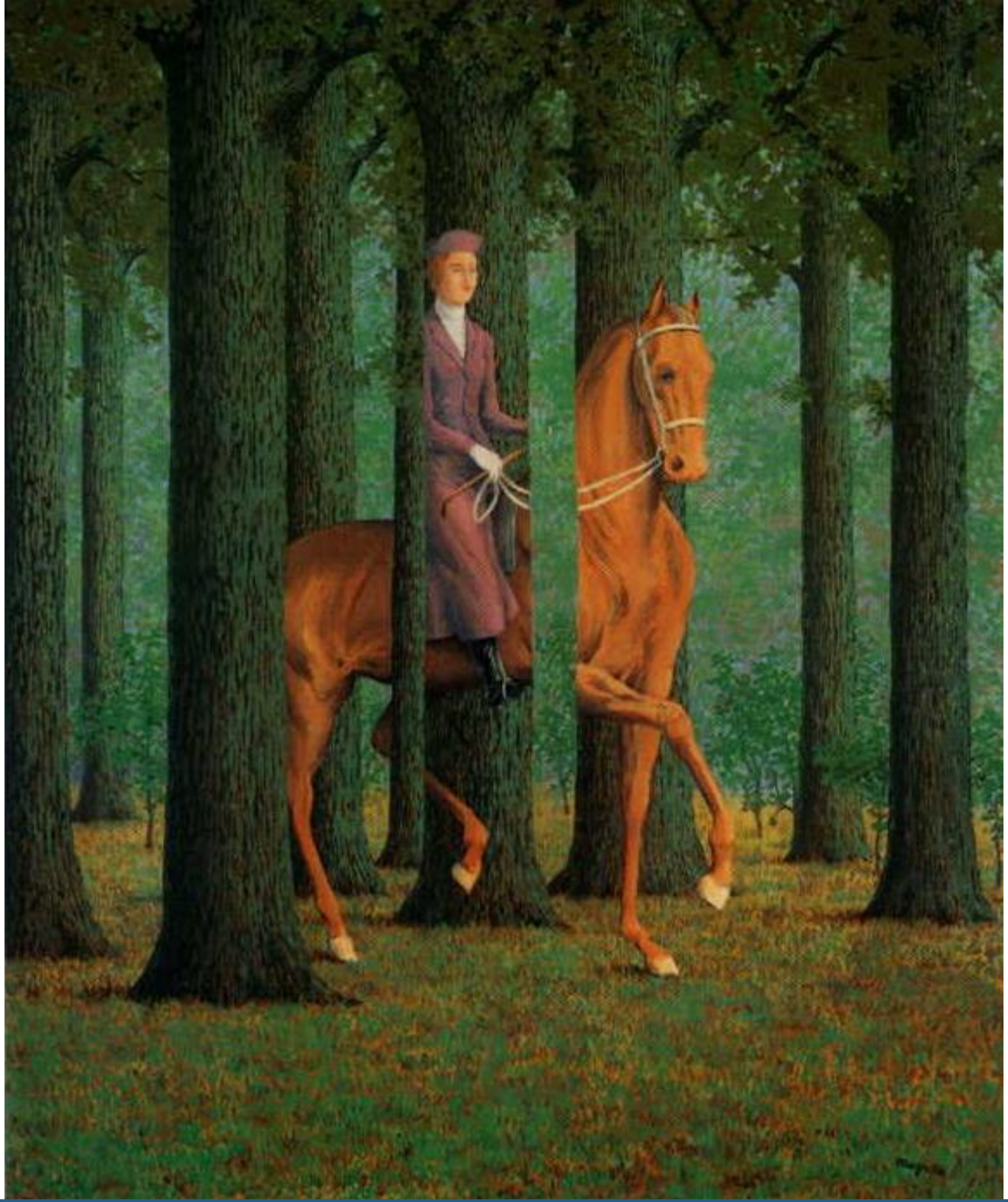


image credit: J. Koenderink

Challenges: scale

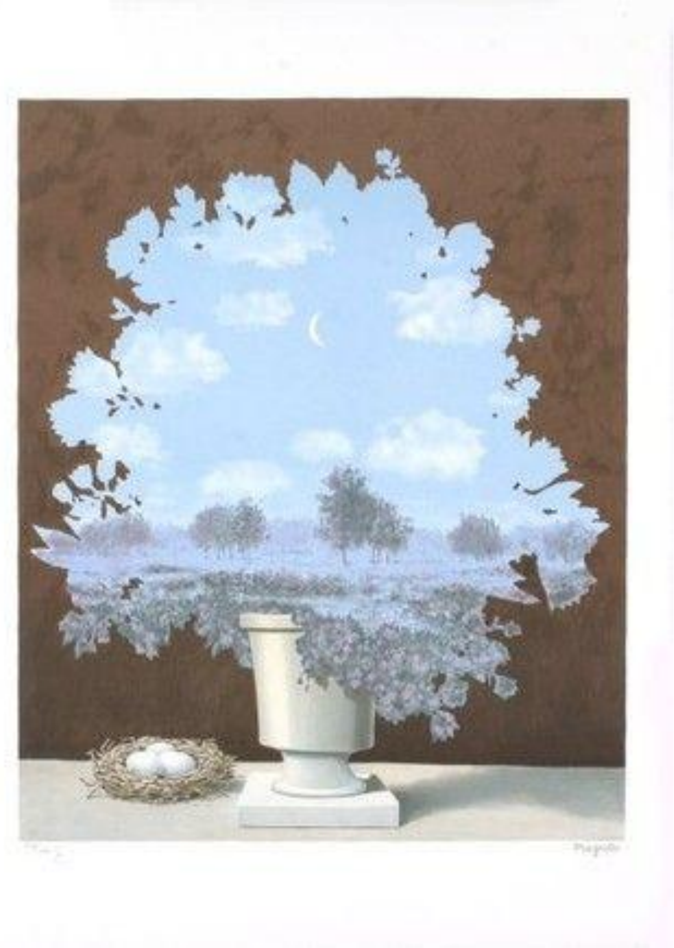
Challenges:
deformation

Challenges: occlusion



Magritte, 1957

Art Segway - [Magritte]



Challenges: background clutter

Kilmeny Niland. 1995

Challenges: intra-class variation






Today’s agenda

- Introduction to recognition
- A object recognition pipeline
- Choosing the right features
- A training algorithm: KNN
- Testing an algorithm
- Challenges with kNN
- Dimensionality reduction

Object recognition:
a classification framework

- Apply a prediction function to a feature representation of the image to get the desired output:

f() = “apple”

f() = “tomato”

f() = “cow”

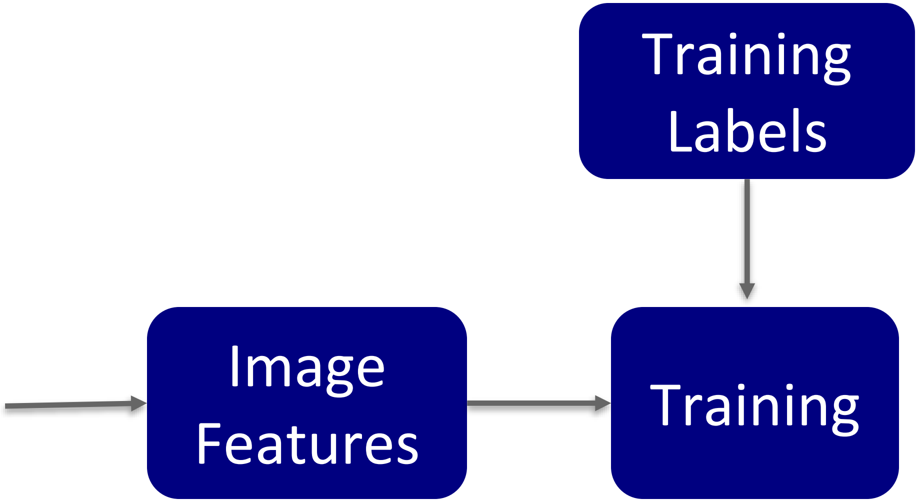
A simple pipeline - Training

Training
Images

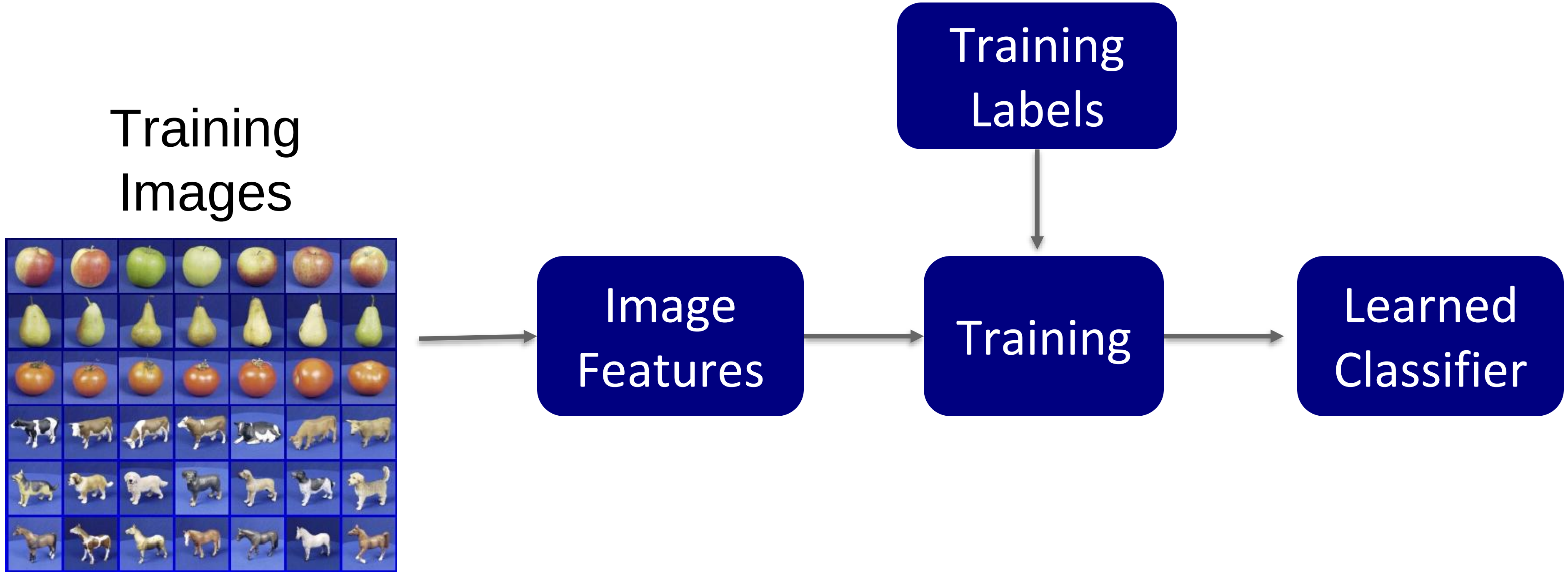
Image
Features

A simple pipeline - Training

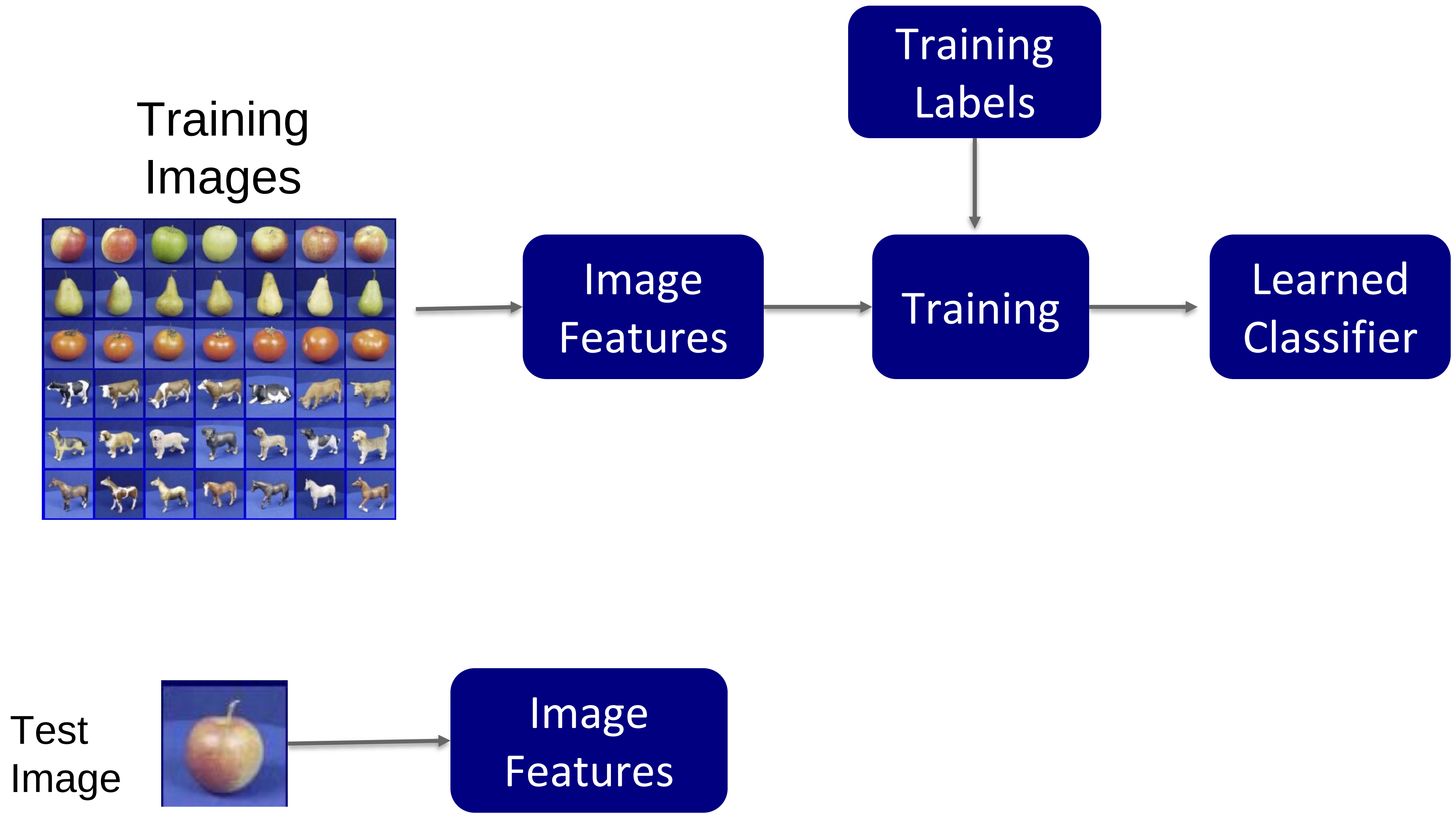
Training Images



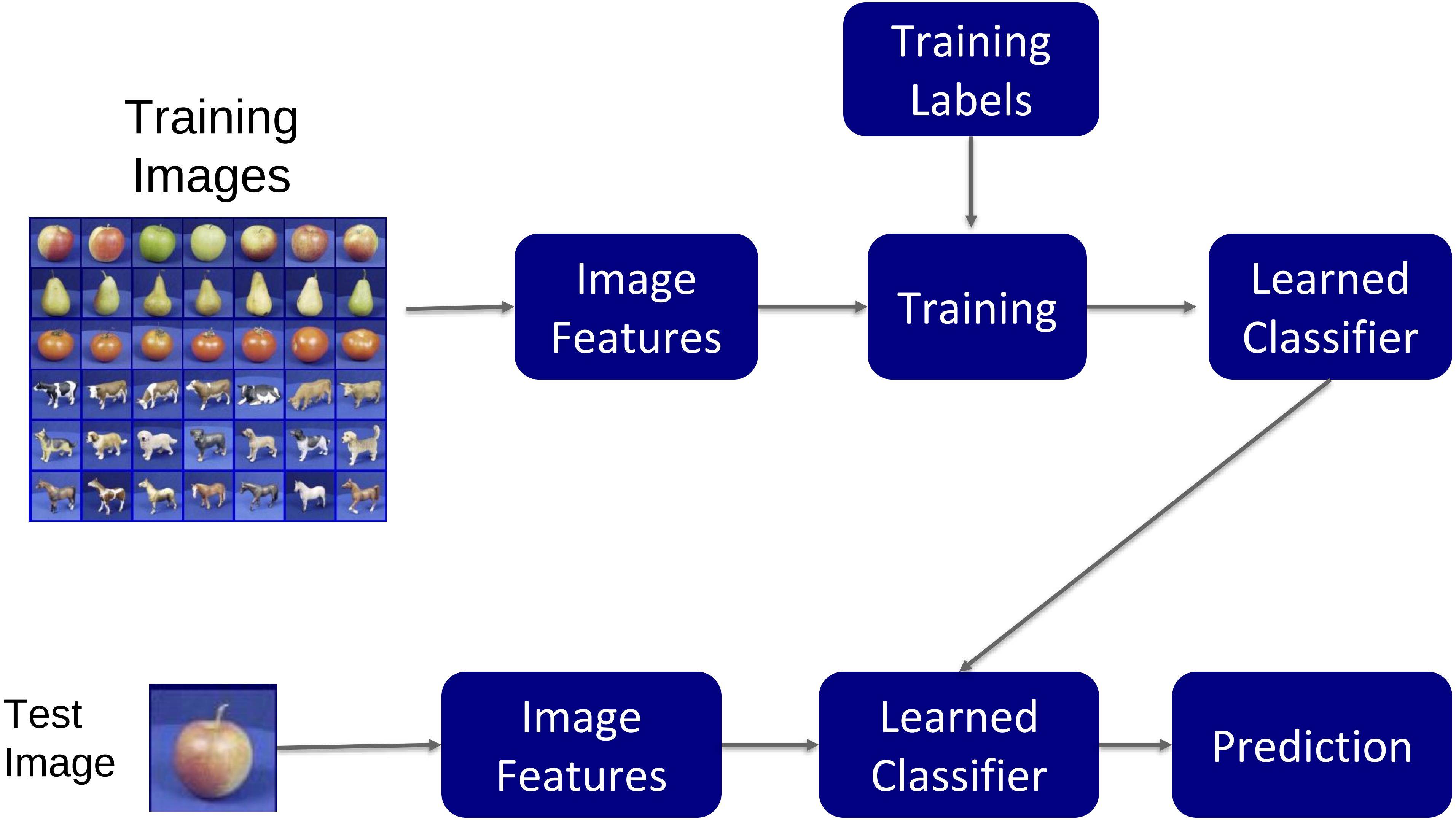
A simple pipeline - Training



A simple pipeline - Training



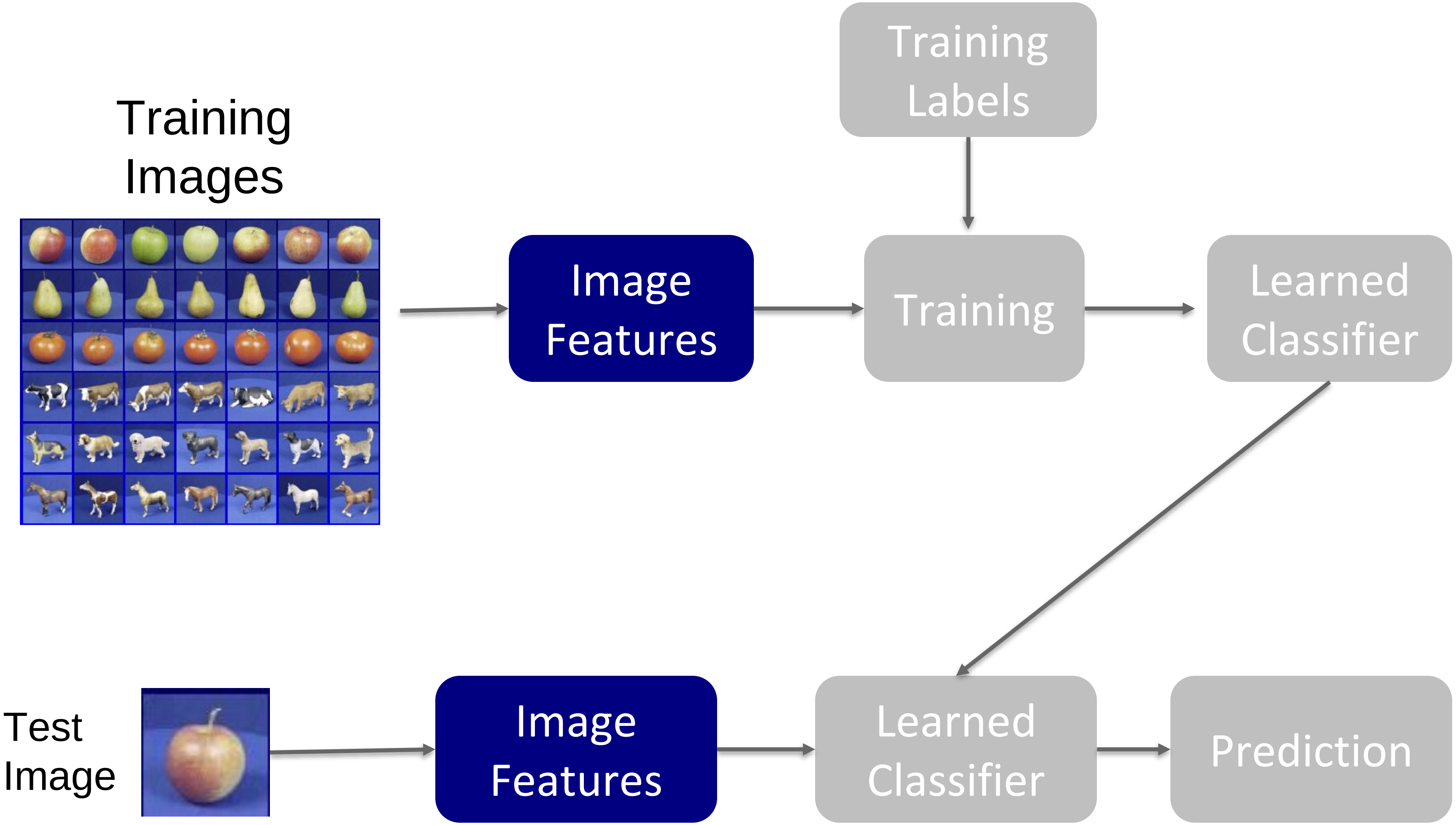
A simple pipeline - Training



What we will learn today?

- Introduction to recognition
- A object recognition pipeline
- Choosing the right features
- A training algorithm: KNN
- Testing an algorithm
- Challenges with kNN
- Dimensionality reduction

A simple pipeline - Training



Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	**?**						

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑						

✓ (global color counts don’t change if the image shifts)

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	?					

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✕					

✓ (if the *entire* image is uniformly scaled, the color distribution remains the same)

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✕	?	?			

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✕	☑	✕			

✓ (rotating the entire image does not change overall color distribution)
✗ (appearance/colors can change if out-of-plane rotation reveals different surfaces)

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	?		

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗		

✗ (removing part of the image can significantly alter color histogram)

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✕	☑	✕	✕	?	?

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✖	☑	✖	✖	✖	✖

✗ (shifts in illumination change color intensities/distribution)
✗ (noise directly alters pixel distribution)

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	?	?	?	?			

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	YES	✗	✗	✗			

✗ (local bins move)
✗ (needs re-computation at multiple scales)
✗ (oriented gradients are tied to an image grid)
✗ (same reason as the ^)

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✕	☑	✕	✕	✕	✕
HoG	$\boxed{\text{YES}}$	✕	✕	✕	?	?	?

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	YES	✗	✗	✗	✗	☑	✗

✗ (partial occlusion would result in no match)
✓ (gradients are more stable under monotonic intensity changes)
✗ (gradient orientations can be disrupted by significant noise)

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	YES	✗	✗	✗	✗	☑	✗
SIFT	**?**	**?**	**?**	**?**			

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	YES	✗	✗	✗	✗	☑	✗
SIFT	☑	☑	☑	✗			

✓ (keypoint-based, unaffected by shift)
✓ (built-in scale normalization)
✓ (SIFT normalizes orientation)
✗ (local keypoints might disappear if the object rotates)

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	YES	✗	✗	✗	✗	☑	✗
SIFT	☑	☑	☑	✗	?	?	?

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	YES	✗	✗	✗	✗	☑	✗
SIFT	☑	☑	☑	✗	☑	☑	☑
Deep learning							

✓ (local keypoints can still match if some are visible)
✓ (gradient-based + normalization)
✓ (SIFT is relatively robust to moderate noise)

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	$\boxed{\text{YES}}$	✗	✗	✗	✗	☑	✗
SIFT	☑	☑	☑	✗	☑	☑	☑
Deep learning	**?**	**?**	**?**	**?**			

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	YES	✗	✗	✗	✗	☑	✗
SIFT	☑	☑	☑	✗	☑	☑	☑
Deep learning	usually	usually	usually	✗			

~ Deep learning features are usually invariant to translation, scale, and planar rotation if the training data has these translations. It is not invariant to other rotations.

Aside: ImageNet has objects centered in the middle of images. So models trained on ImageNet are not translation or scale invariant.

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	$\boxed{\text{YES}}$	✗	✗	✗	✗	☑	✗
SIFT	☑	☑	☑	✗	☑	☑	☑
Deep learning	usually	usually	usually	✗	**?**	**?**	**?**

Choices of features

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	YES	✗	✗	✗	✗	☑	✗
SIFT	☑	☑	☑	✗	☑	☑	☑
Deep learning	usually	usually	usually	✗	✗	☑	☑

✗ (standard CNNs are not strictly occlusion-invariant; partial robustness depends on training)
✓ (can learn robustness if trained on varied lighting)
✓ (CNNs can learn to be noise-robust with proper training)

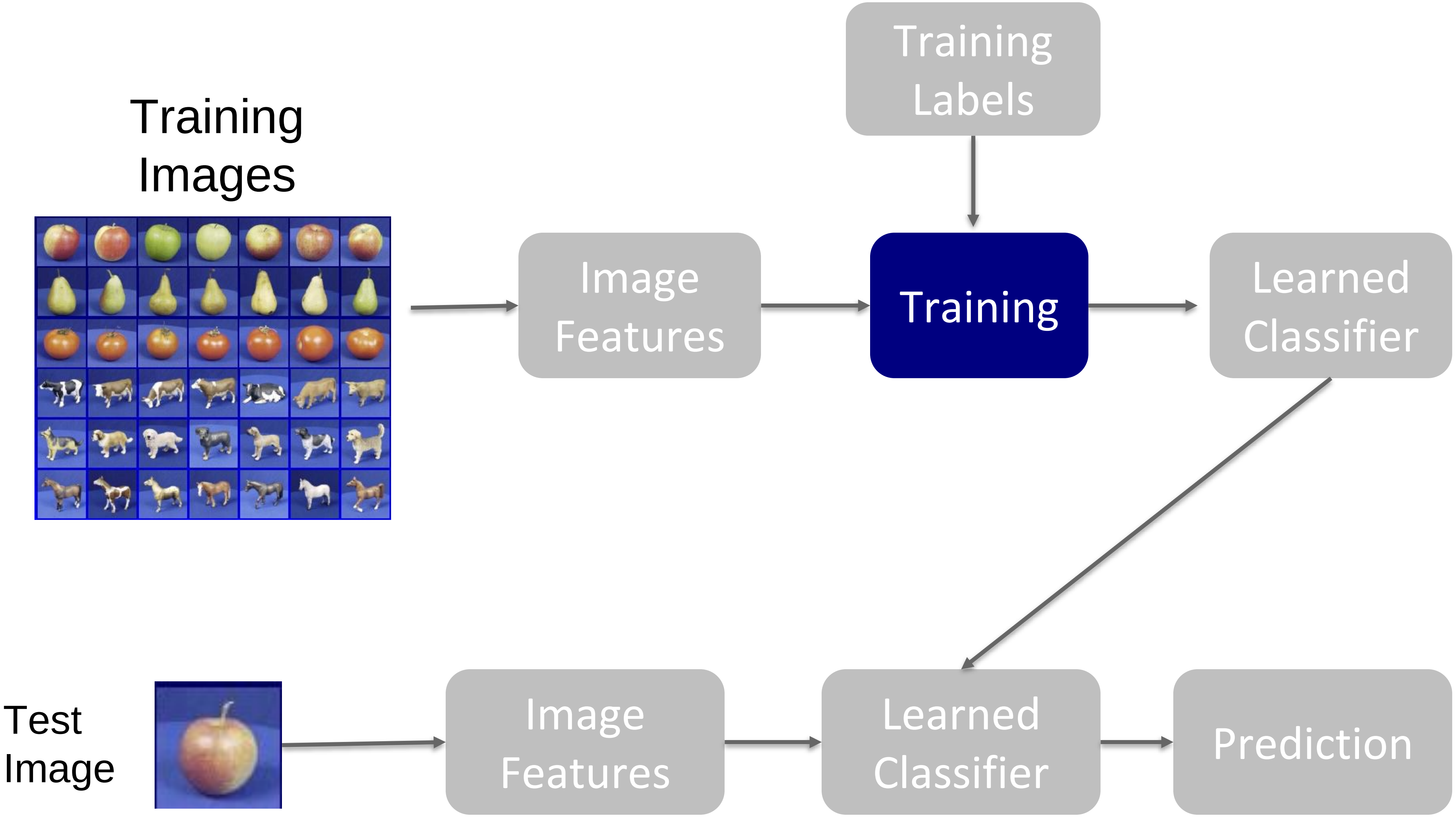
So, which features should we choose?

	Invariances						
	Translation	**Scale**	**Rotation** (relative to camera plane)	**Rotation** (unconstrained)	**Partial Occlusion**	**Illumination**	**Gaussian Noise**
RGB-histogram	☑	✗	☑	✗	✗	✗	✗
HoG	$\boxed{\text{YES}}$	✗	✗	✗	✗	☑	✗
SIFT	☑	☑	☑	✗	☑	☑	☑
Deep learning	usually	usually	usually	✗	✗	☑	☑

What we will learn today?

- Introduction to recognition
- A simple Object Recognition pipeline
- Choosing the right features
- A training algorithm: KNN
- Testing an algorithm
- Challenges with kNN
- Dimensionality reduction

A simple pipeline - Training



Many classifiers to choose from

- **K-nearest neighbor**
- SVM
- Neural networks
- Naïve Bayes
- Bayesian network
- Logistic regression
- Randomized Forests
- Boosted Decision Trees
- RBMs
- Etc.

Which is the best one?

Learning a classifier to map inputs to outputs

$$y = f(\mathbf{x})$$

output

prediction function

Image feature

- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_N,y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$

An example training dataset

Apples

Pear

Tomatos

Cow

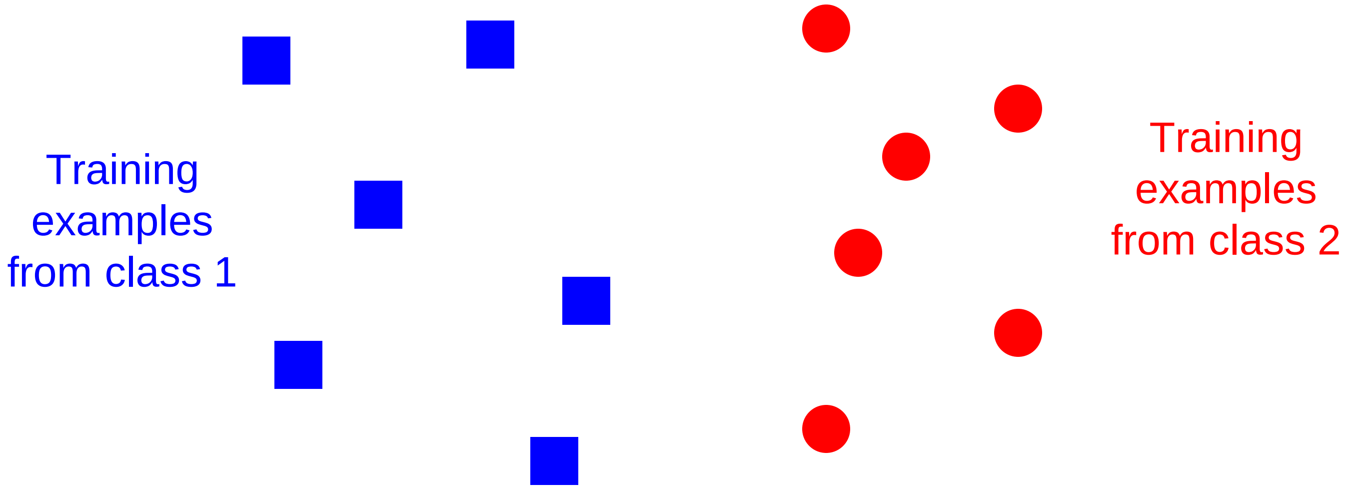
Dog

Horse

Training set (labels known)

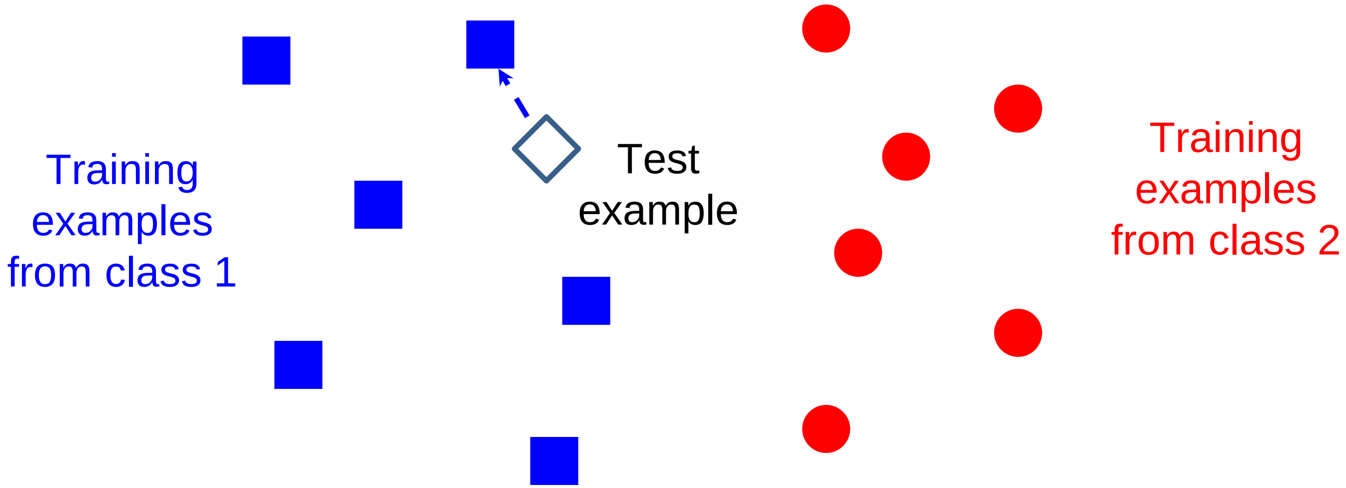
For kNN classifier, training simply means to store all training data.

A stored training set



Slide credit: L. Lazebnik

During testing, we assign the label of the nearest neighbor in feature space

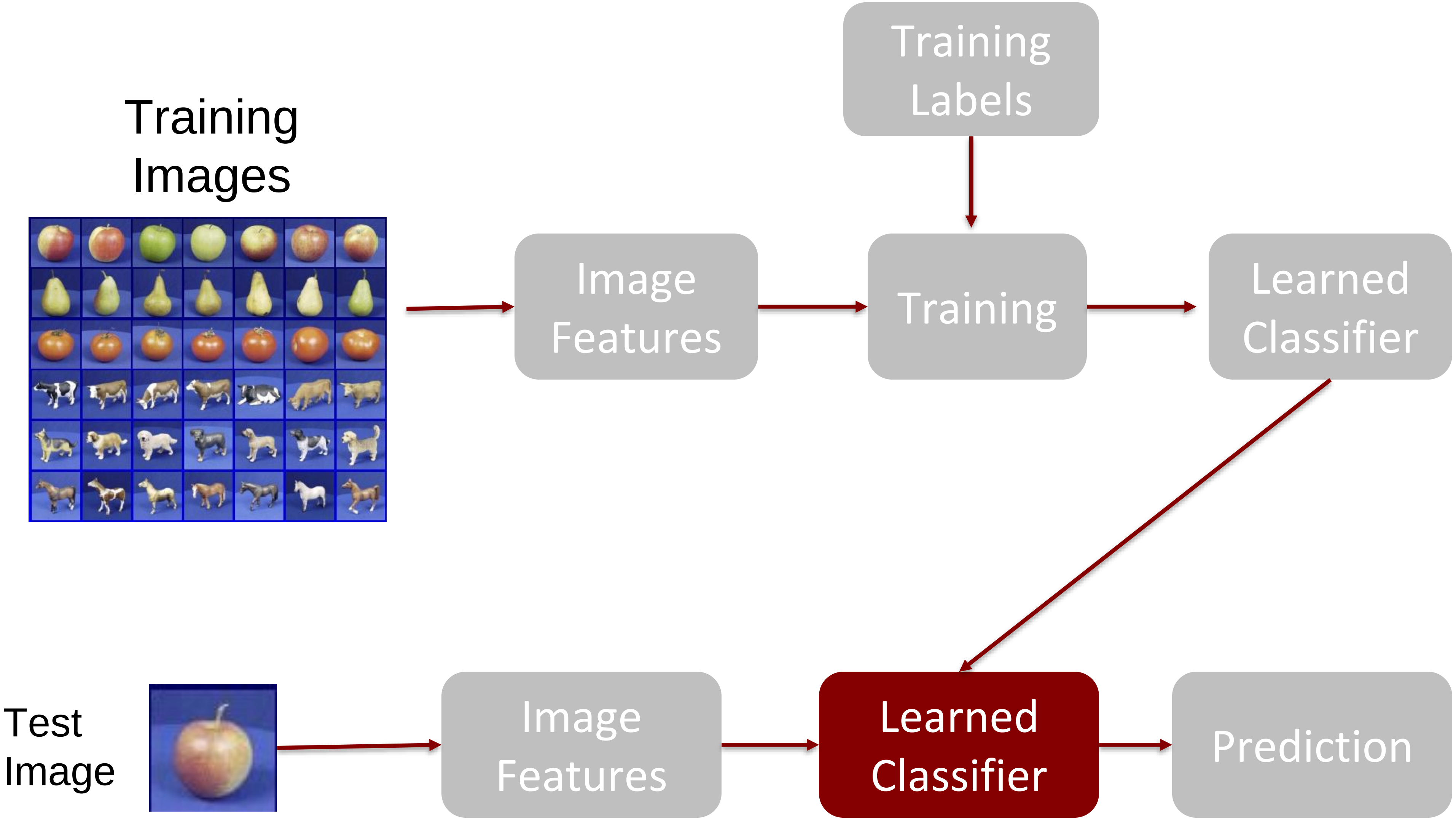


Slide credit: L. Lazebnik

What we will learn today?

- Introduction to recognition
- A simple Object Recognition pipeline
- Choosing the right features
- A training algorithm: kNN
- Testing an algorithm
- Challenges with kNN
- Dimensionality reduction

A simple pipeline - Training



Generalization

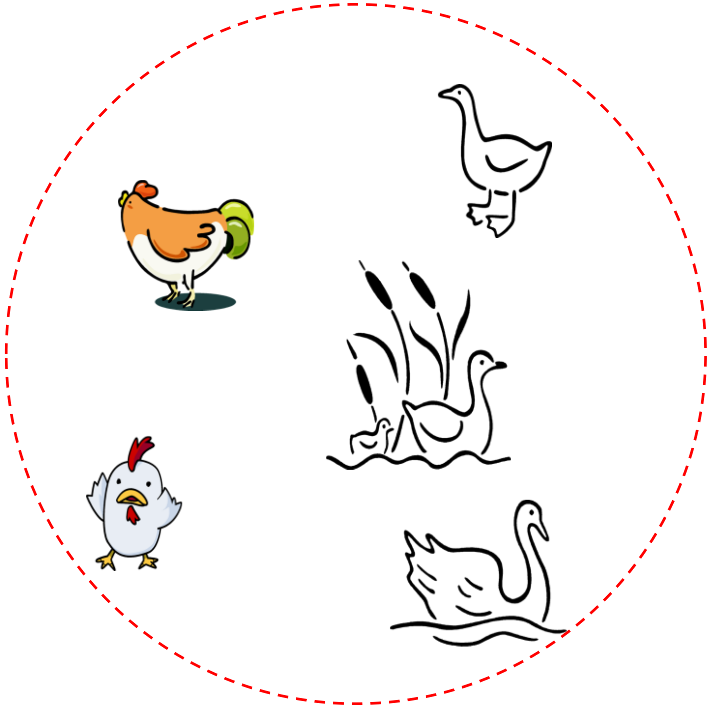
Training set (labels known)

Test set (labels unknown)

- How well does a learned model generalize from the data it was trained on to a new test set?

Intuition for Nearest Neighbor Classifier

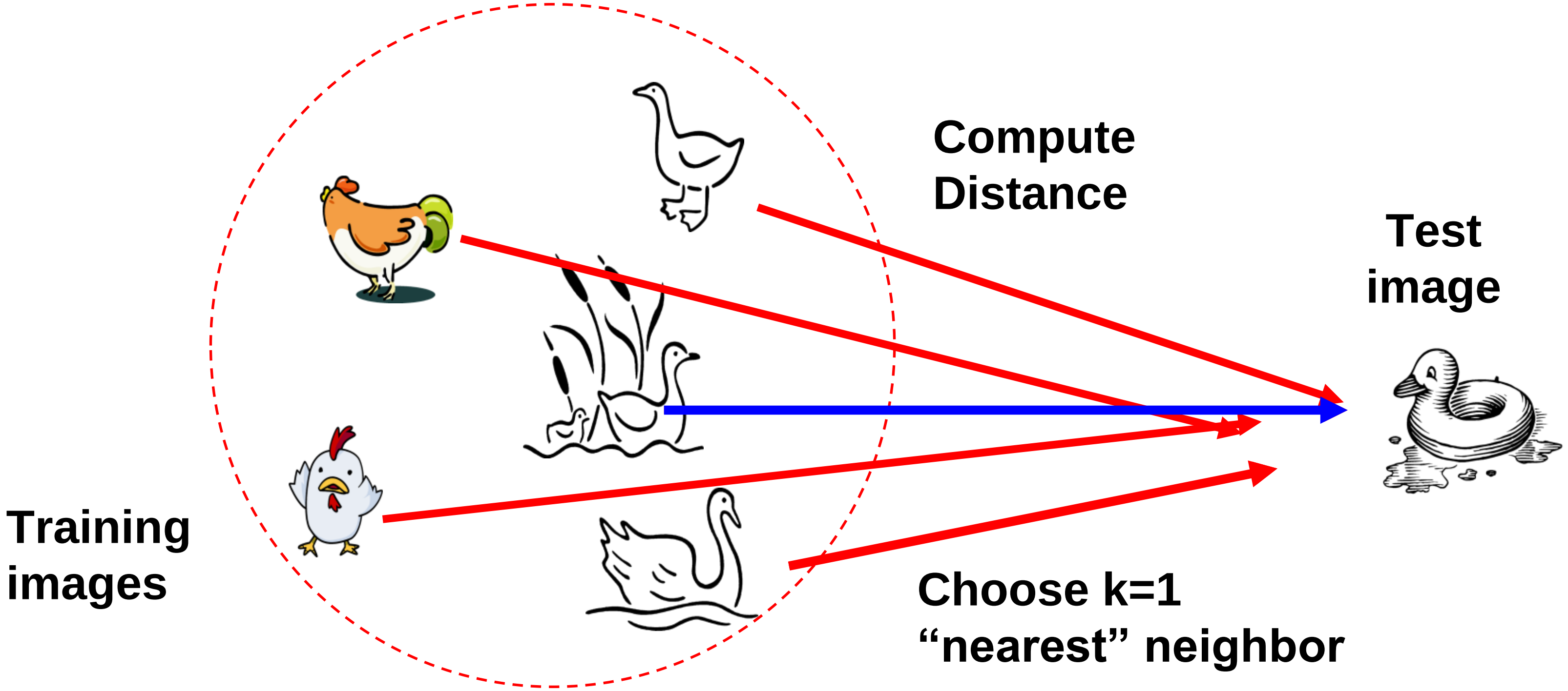
Given a training dataset, simply store each image's features and their corresponding label.



Training images

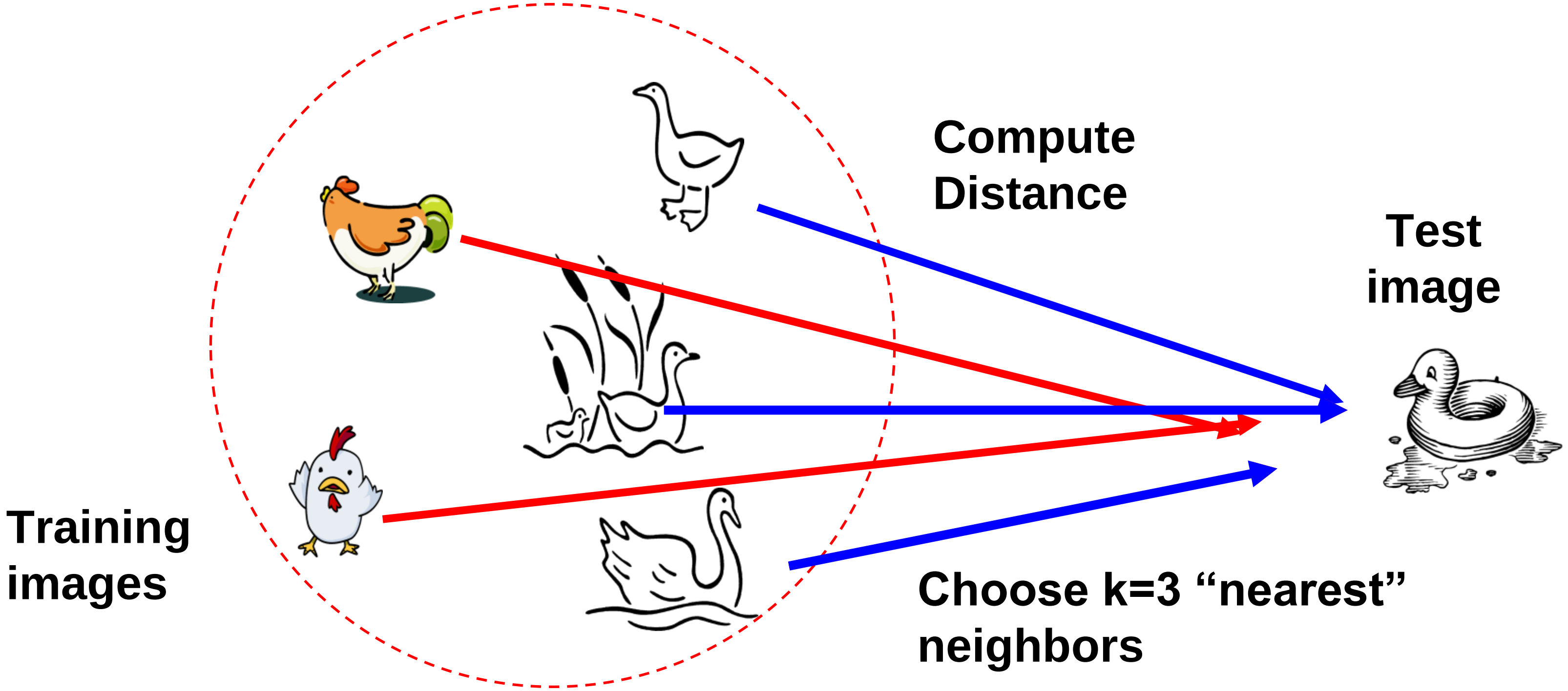
Intuition for Nearest Neighbor Classifier

Given a training dataset, simply store each image's features and their corresponding label.



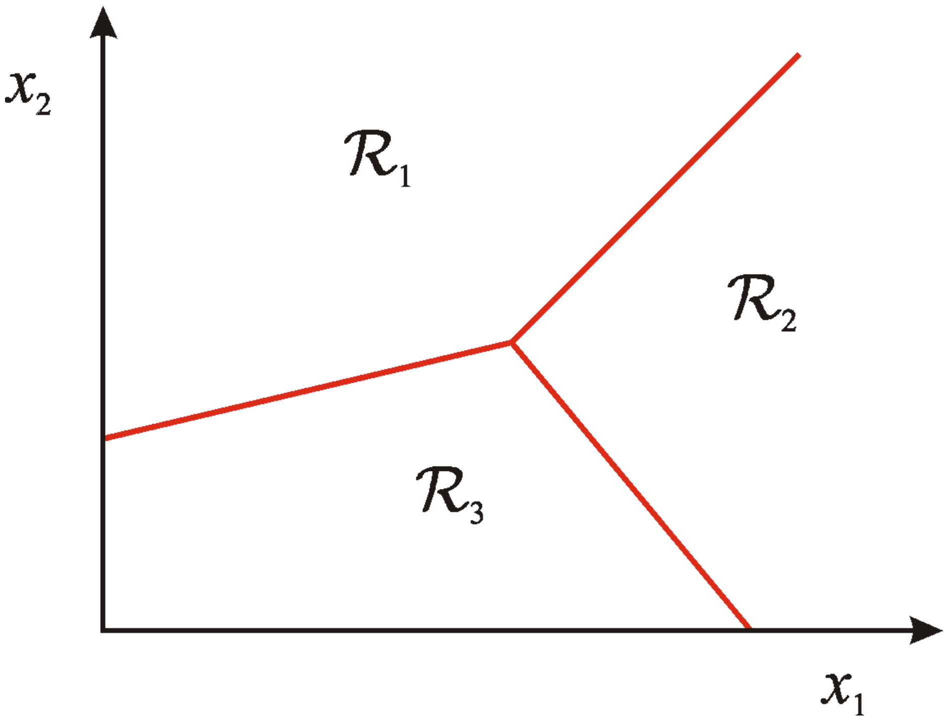
Nearest Neighbor Classifier

- Assign label of majority of K=3 nearest neighbors



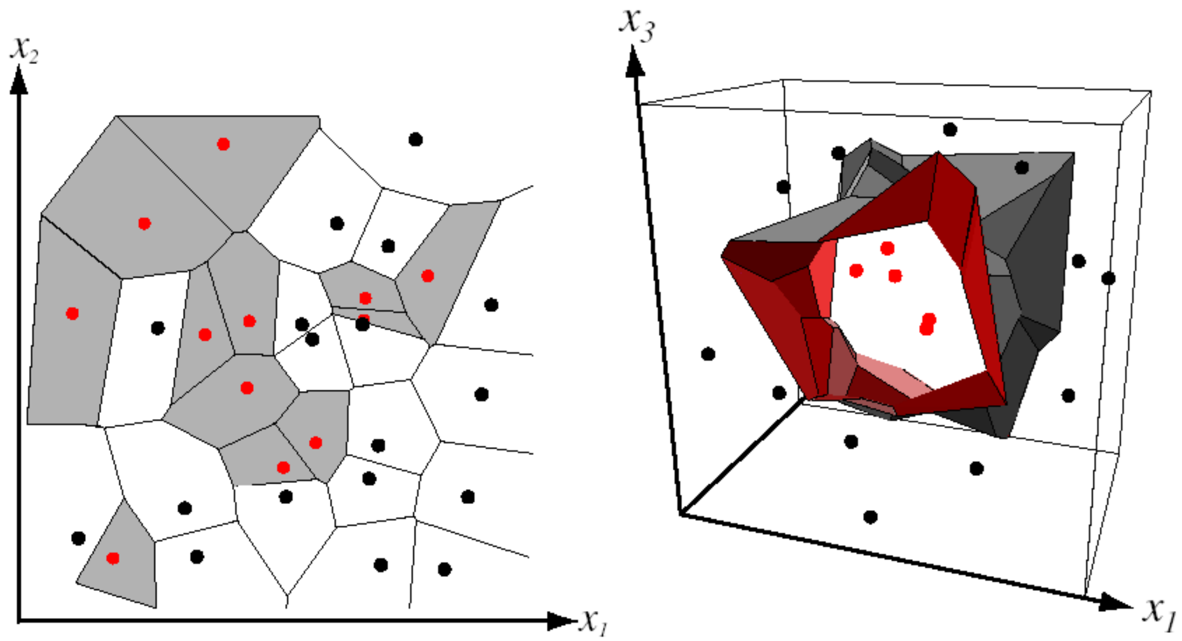
Classification

- Assign input vector to one of many classes (categories)
- **Geometric interpretation** of classifiers: A classifier divides input space into *decision regions* separated by *decision boundaries*



Nearest Neighbor Classifier

- Assign label of nearest training data point to each test data point



from Duda *et al.*

Partitioning of feature space
for two-category 2D and 3D data

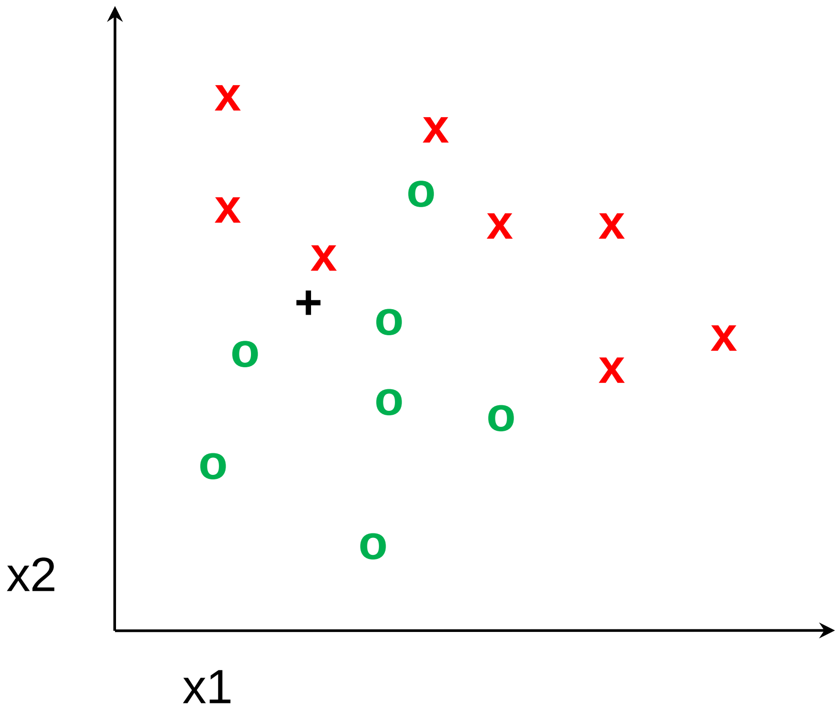
How do we find the nearest neighbors in feature space?

Distance measure (same as the ones from segmentation)

Euclidean:

$$Dist(X^n, X^m) = \sqrt{\sum_{i=1}^{D} (X_i^n - X_i^m)^2}$$

Where X^n and X^m are the n-th and m-th data points



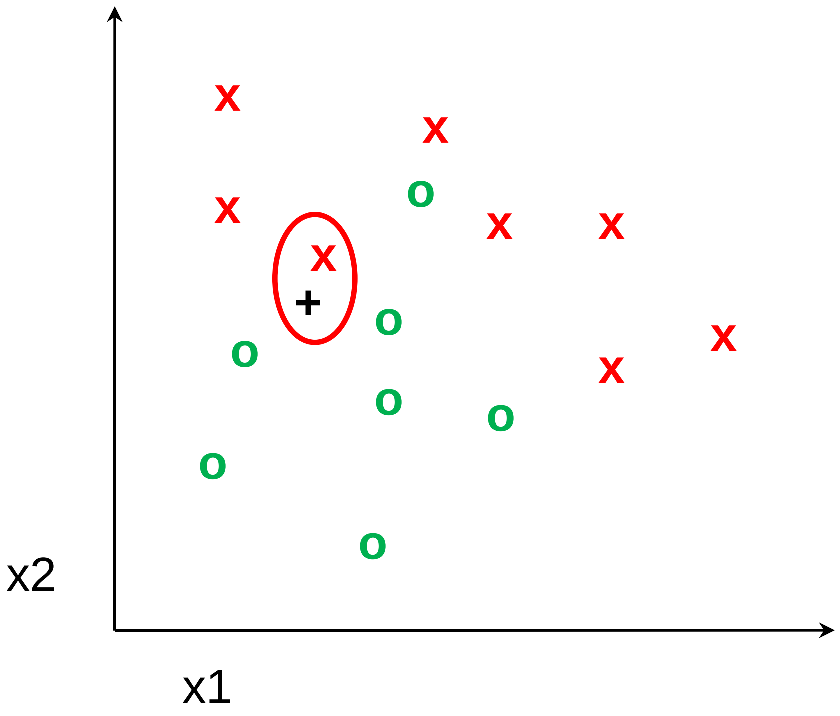
1-nearest neighbor

Distance measure (same as the ones from segmentation)

Euclidean:

$$Dist(X^n, X^m) = \sqrt{\sum_{i=1}^{D} (X_i^n - X_i^m)^2}$$

Where X^n and X^m are the n-th and m-th data points



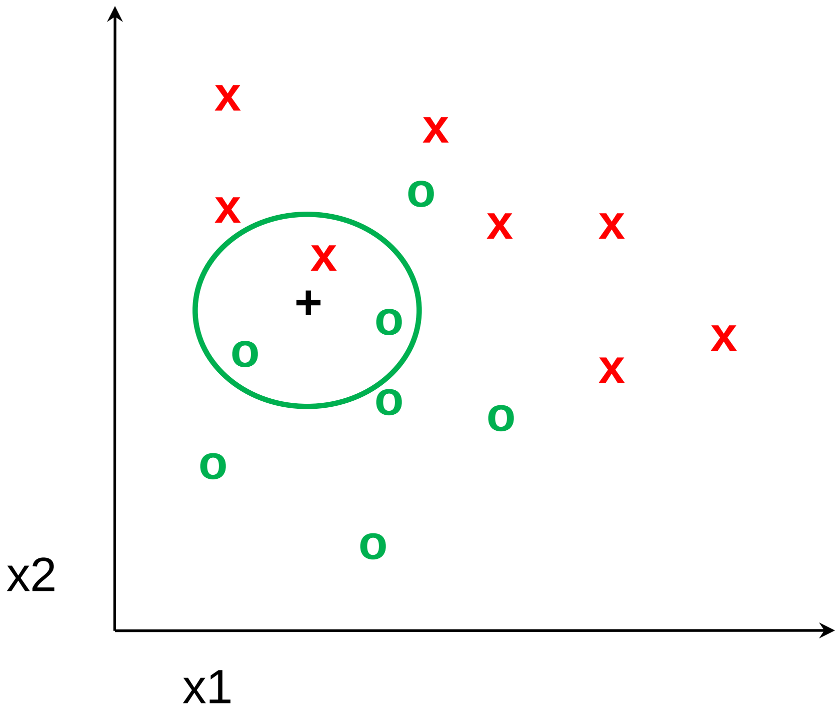
3-nearest neighbor

Distance measure (same as the ones from segmentation)

Euclidean:

$$Dist(X^n, X^m) = \sqrt{\sum_{i=1}^{D} (X_i^n - X_i^m)^2}$$

Where X^n and X^m are the n-th and m-th data points



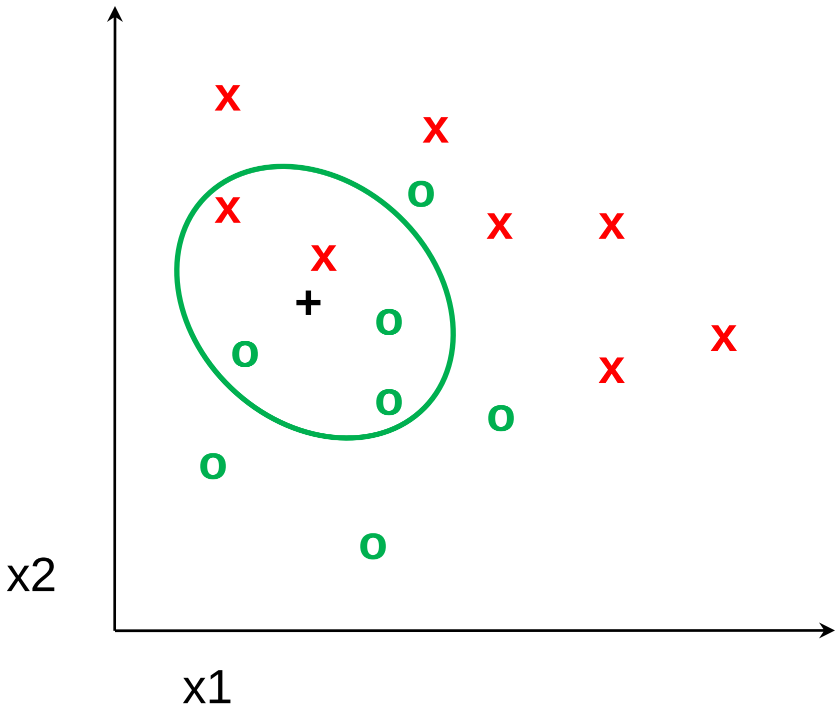
5-nearest neighbor

Distance measure (same as the ones from segmentation)

Euclidean:

$$Dist(X^n, X^m) = \sqrt{\sum_{i=1}^{D} (X_i^n - X_i^m)^2}$$

Where X^n and X^m are the n-th and m-th data points



Choosing the right features is important but dataset-dependent

	Color	$D_x D_y$	Mag-Lap	PCA Masks	PCA Gray	Cont. Greedy	Cont. DynProg	Avg.
apple	57.56%	**85.37%**	80.24%	78.78%	**88.29%**	77.07%	76.34%	77.66%
pear	66.10%	90.00%	85.37%	**99.51%**	**99.76%**	90.73%	91.71%	89.03%
tomato	**98.54%**	94.63%	**97.07%**	67.80%	76.59%	70.73%	70.24%	82.23%
cow	86.59%	82.68%	**94.39%**	75.12%	62.44%	86.83%	86.34%	82.06%
dog	34.63%	62.44%	74.39%	72.20%	66.34%	**81.95%**	**82.93%**	67.84%
horse	32.68%	58.78%	70.98%	77.80%	77.32%	**84.63%**	**84.63%**	69.55%
cup	79.76%	66.10%	77.80%	**96.10%**	**96.10%**	**99.76%**	**99.02%**	87.81%
car	62.93%	**98.29%**	77.56%	**100.0%**	**97.07%**	**99.51%**	**100.0%**	90.77%
total	64.85%	79.79%	82.23%	83.41%	82.99%	86.40%	86.40%	80.87%

Dataset: ETH-80, by B. Leibe, 2003

K-NN: a very useful algorithm

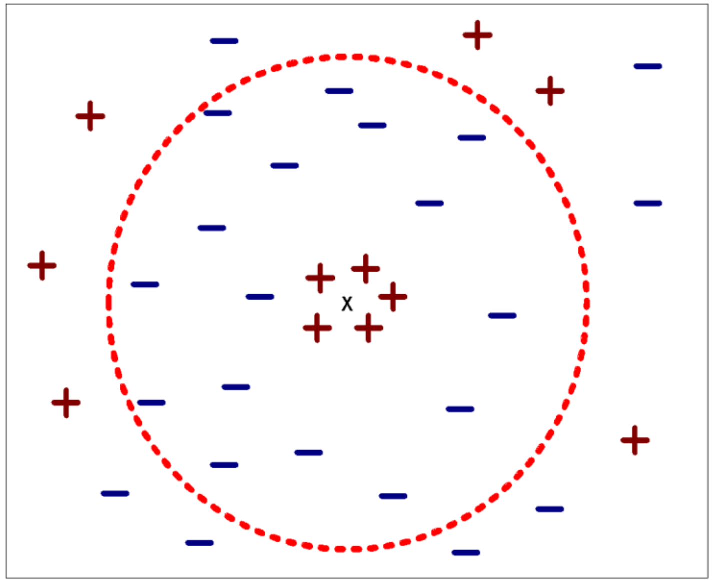
- Simple, a good one to try first
- Very flexible decision boundaries
- With infinite examples, 1-NN has a strong theoretical guarantee (out of scope for this class)

What we will learn today?

- Introduction to recognition
- A simple Object Recognition pipeline
- Choosing the right features
- A training algorithm: kNN
- Testing an algorithm
- Challenges with kNN
- Dimensionality reduction

K-NN: issues to keep in mind

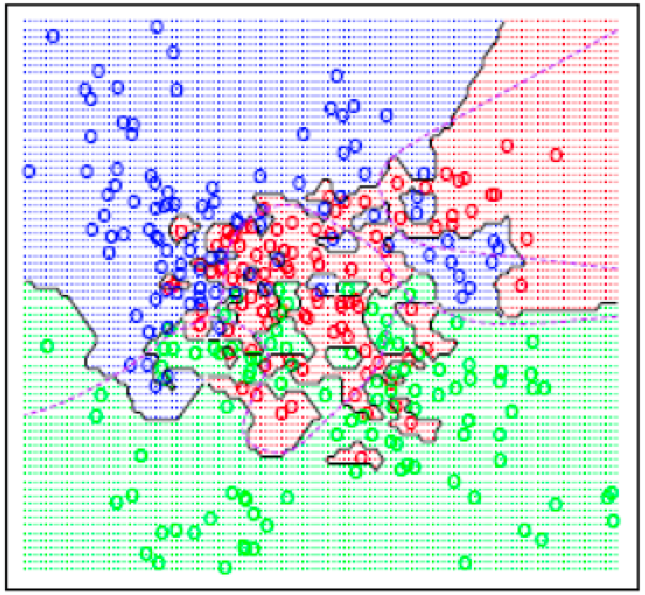
- Choosing the value of k:
  - If too small, sensitive to noise points
  - If too large, neighborhood may include points from other classes



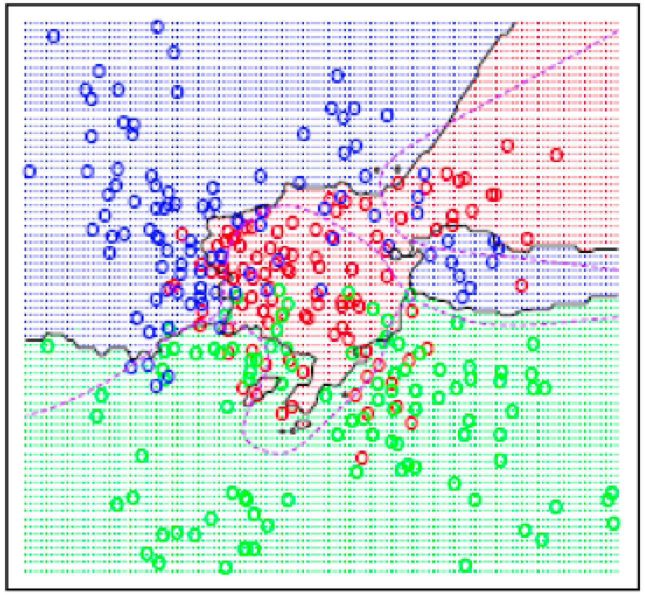
K-NN: issues to keep in mind

- Choosing the value of k:
    - If too small, sensitive to noise points
    - If too large, neighborhood may include points from other classes

K=1

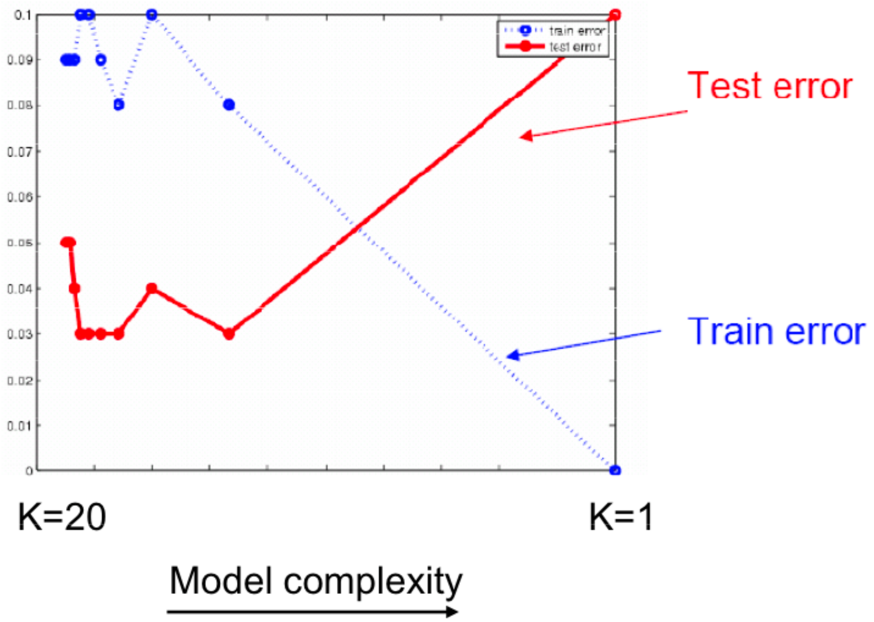


K=15



K-NN: issues to keep in mind

- Choosing the value of k:
  - If too small, sensitive to noise points
  - If too large, neighborhood may include points from other classes
  - Solution: Cross validate

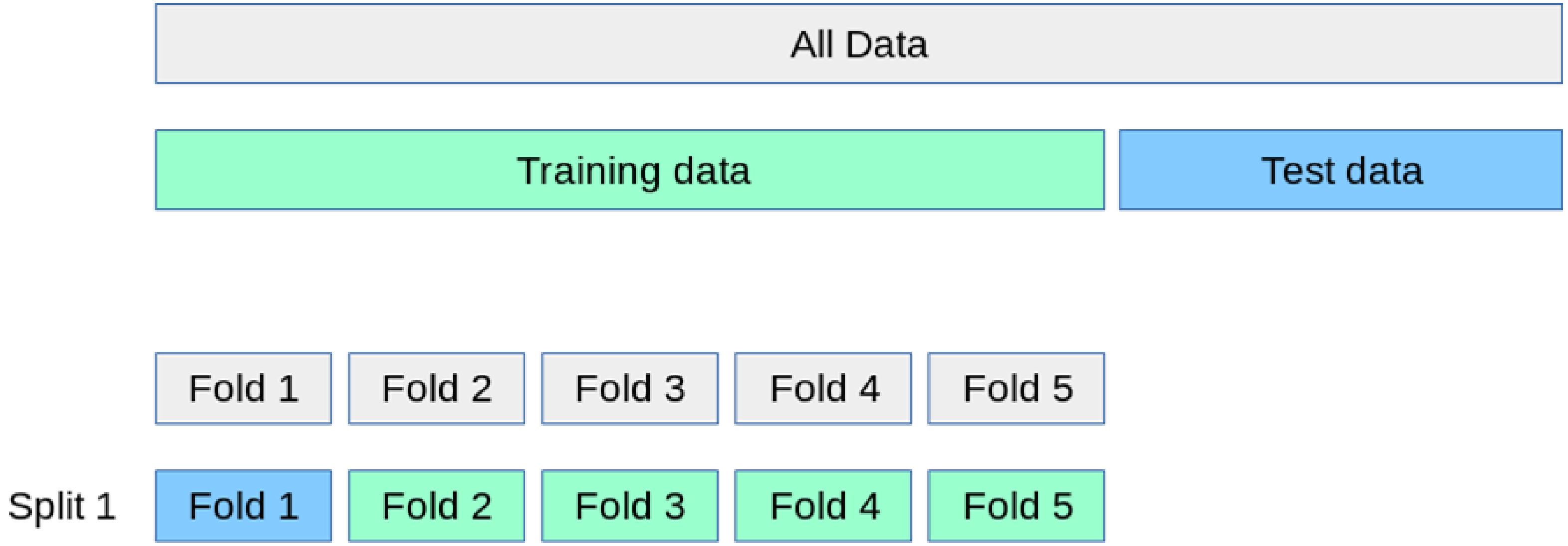


Cross validation

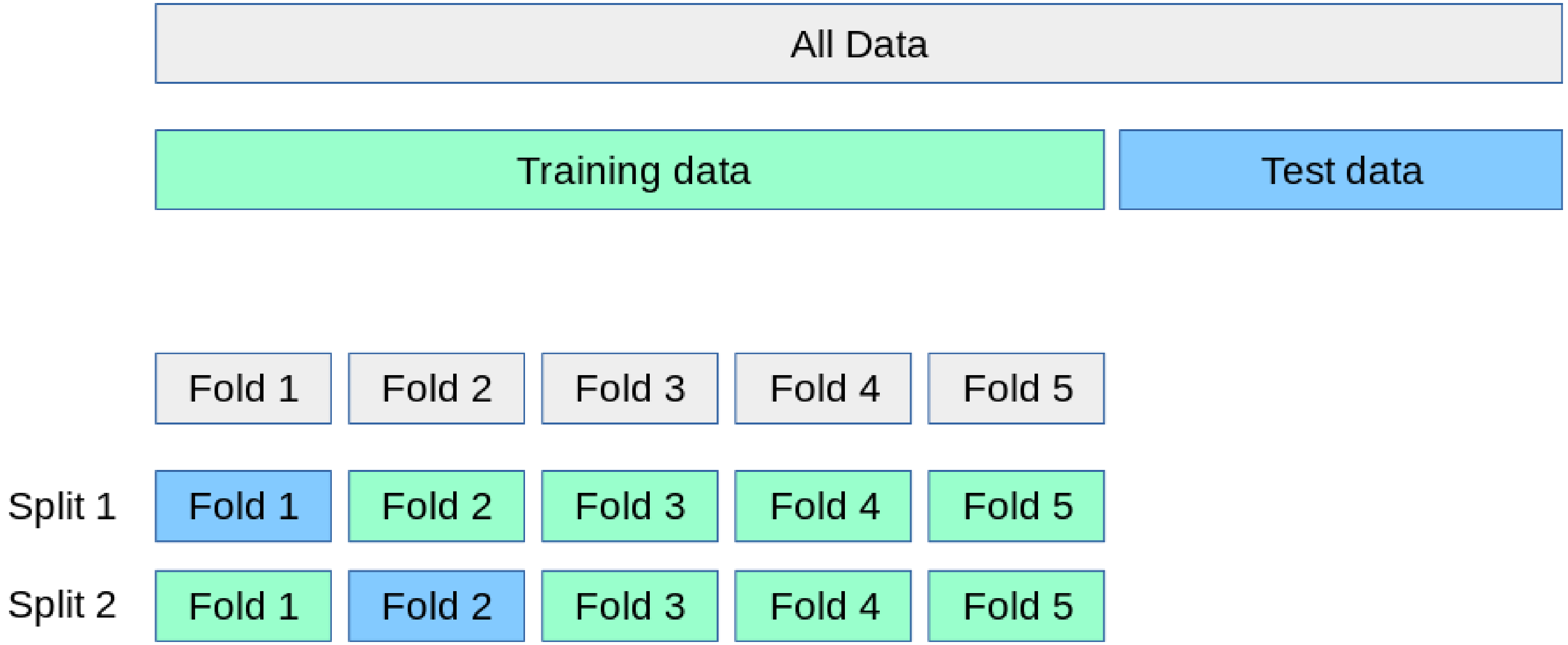
All Data

Training data | Test data

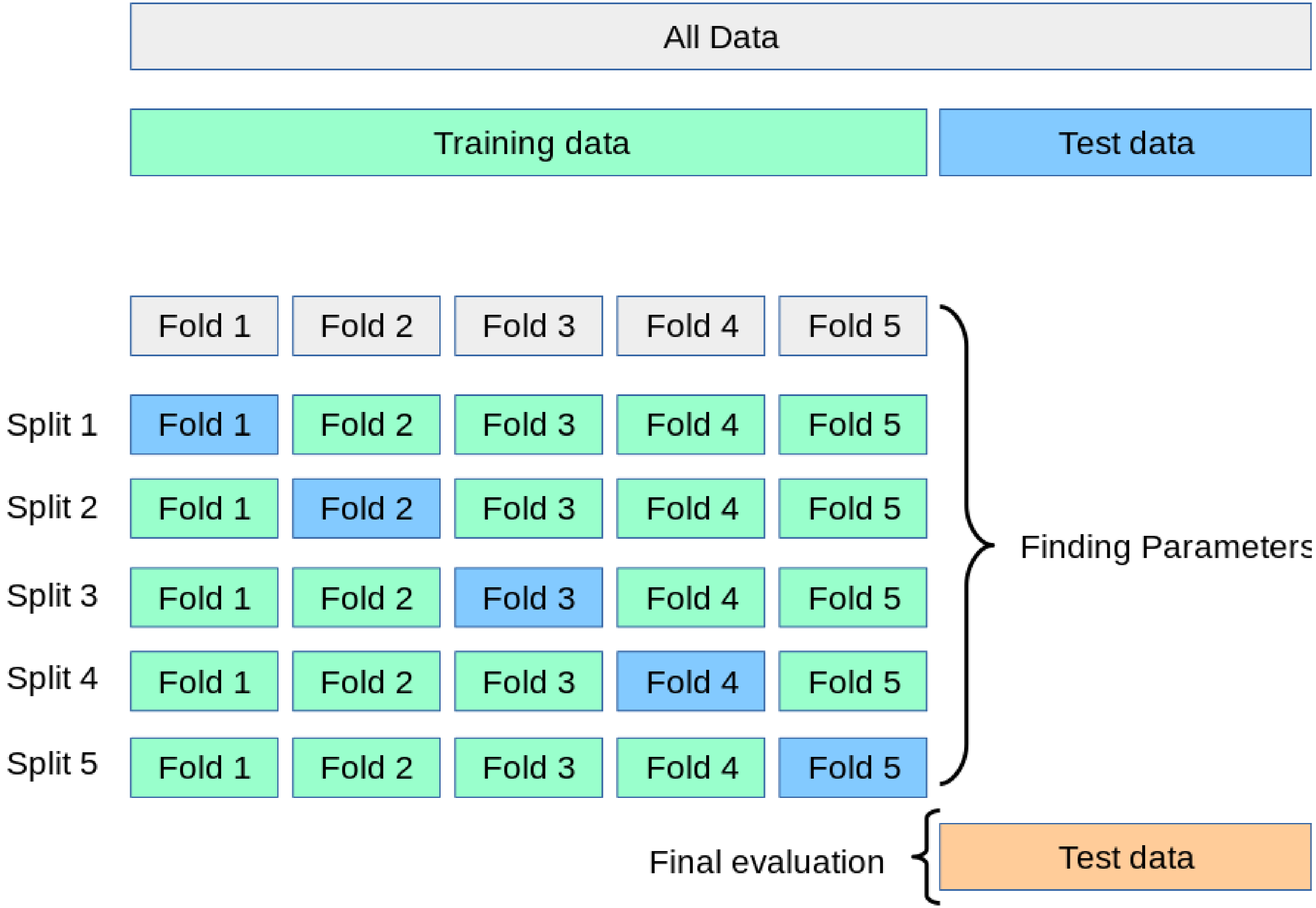
Cross validation



Cross validation



Cross validation

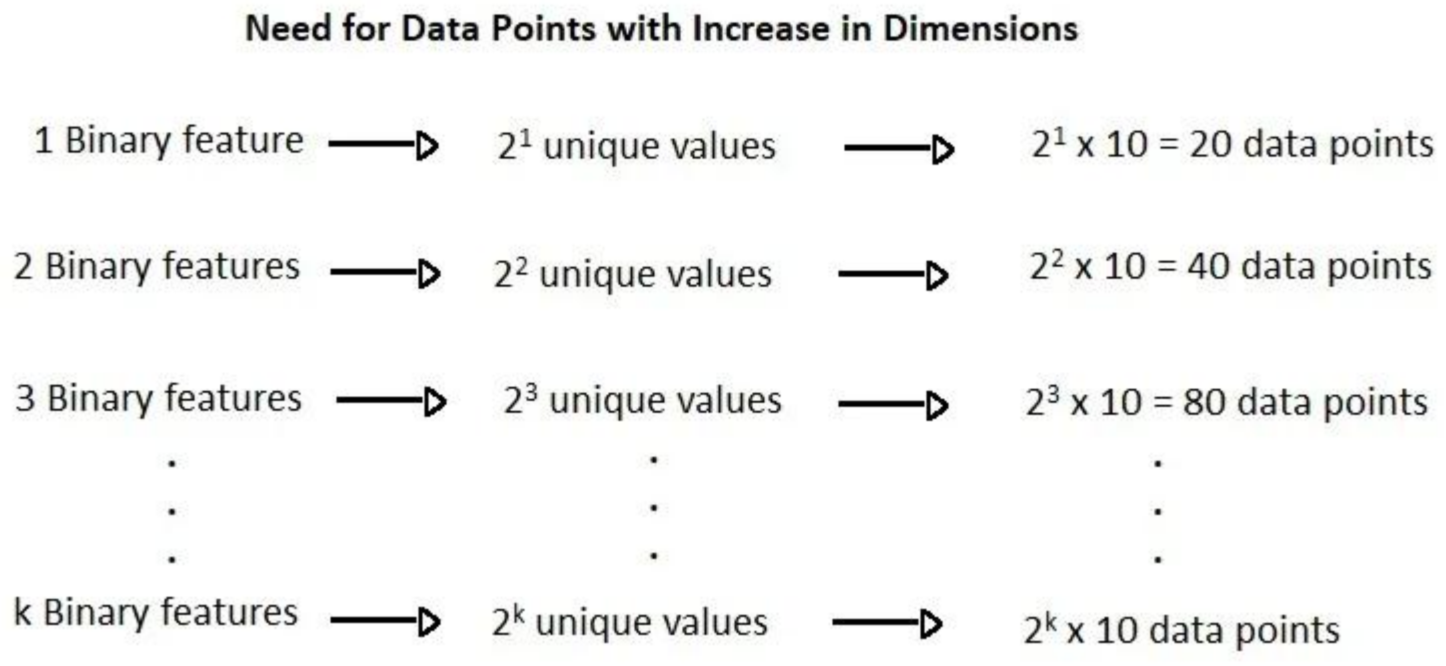


K-NN: issues to keep in mind

- Choosing the value of k:
  - If too small, sensitive to noise points
  - If too large, neighborhood may include points from other classes
  - Solution: cross validate!
- Curse of Dimensionality

Curse of dimensionality

- As the dimensionality increases, the number of data points required for good performance increases exponentially.
- Let's say that for a model to perform well, we need at least 10 data points for each combination of feature values.



K-NN: issues to keep in mind

- Choosing the value of k:
  - If too small, sensitive to noise points
  - If too large, neighborhood may include points from other classes
  - Solution: cross validate!
- Curse of Dimensionality
  - Solution: dimensionality reduction

What we will learn today

- Introduction to recognition
- A simple Object Recognition pipeline
- Choosing the right features
- A training algorithm: kNN
- Testing an algorithm
- Challenges with kNN
- Dimensionality reduction

Singular Value Decomposition (SVD)

$$\mathbf{U\Sigma V^T = A}$$

- Where **U** and **V** are rotation matrices, and **Σ** is a scaling matrix. For example:

$$\begin{array}{ccccccc} U & & \Sigma & & V^T & & A \\ \begin{bmatrix} -.40 & .916 \\ .916 & .40 \end{bmatrix} & \times & \begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix} & \times & \begin{bmatrix} -.05 & .999 \\ .999 & .05 \end{bmatrix} & = & \begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix} \end{array}$$

What is SVD actually doing for images?

$$\overset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \overset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \overset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

- Look at how the multiplication works out, left to right:
- Column 1 of **U** gets scaled by the first value from **Σ**.

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}}$$

What is SVD actually doing for images?

$$\overset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \overset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \overset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

- Look at how the multiplication works out, left to right:
- Column 1 of **U** gets scaled by the first value from **Σ**.

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}}$$

What is SVD actually doing for images?

$$\overset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \overset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \overset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

- Look at how the multiplication works out, left to right:
- Column 1 of **U** gets scaled by the first value from **Σ**.

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \overset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

- The resulting vector gets scaled by row 1 of $\mathbf{V^T}$ to produce a contribution to the columns of **A**

SVD is a type dimensionality reduction

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \qquad \overset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

$$+ \overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \qquad \overset{A_{partial}}{\begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}}$$

$$= \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- Each product of (*column i of **U***)·(*value i from **Σ***)·(*row i of **V**^{T}*) produces a component of the final **A**.

SVD is a type dimensionality reduction

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \overset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

$$\overset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \overset{A_{partial}}{\begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}}$$

- We’re building **A** as a linear combination of the columns of ***U***
- Using all columns of ***U***, we’ll rebuild the original matrix perfectly
- But, in real-world data, often we can just use the first few columns of ***U*** and we’ll get something close (e.g. the first ***A**$_{partial}$*, above)

SVD is a type dimensionality reduction

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \overset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

$$\overset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \overset{A_{partial}}{\begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}}$$

- We can call those first few columns of ***U*** the Principal Components of the data
- They show the major patterns that can be added to produce the columns of the original matrix
- The rows of **V^{T}** show how the *principal components* are mixed to produce the columns of the matrix

SVD is a type dimensionality reduction

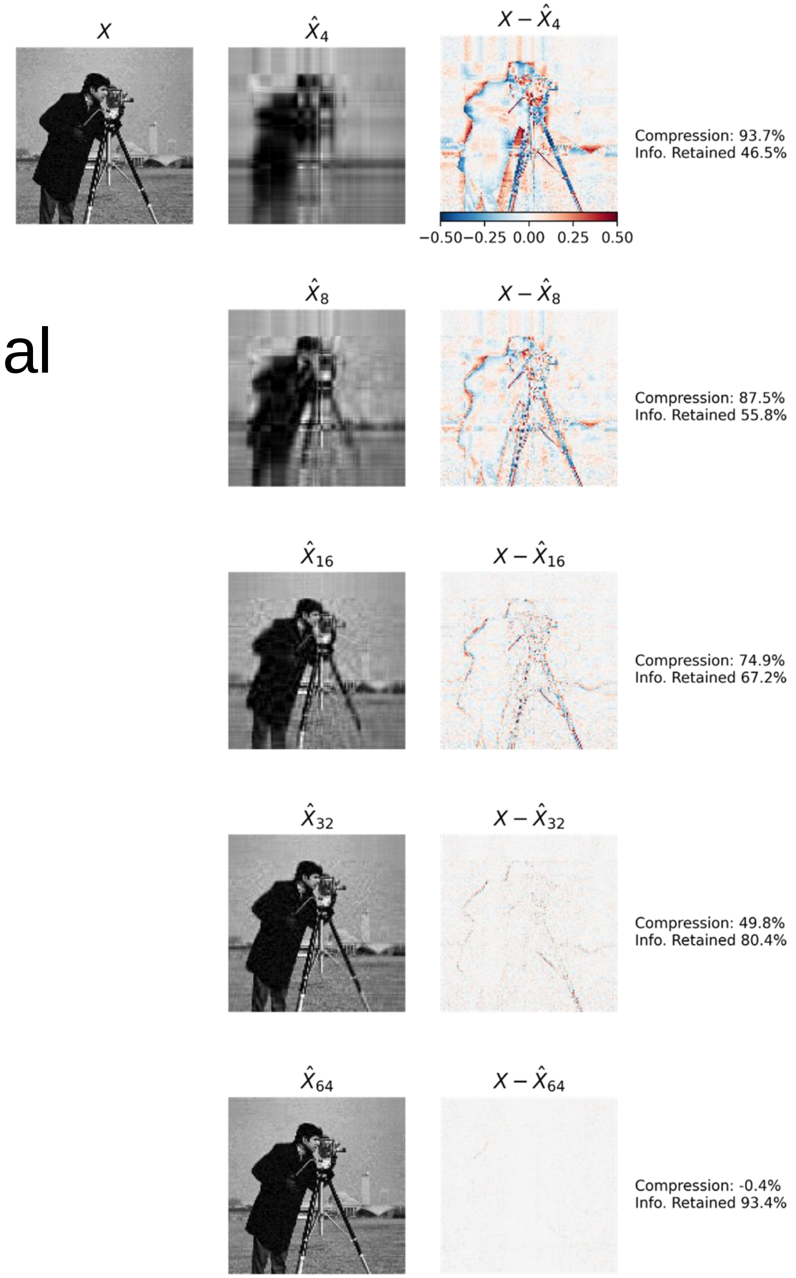
$$\overset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \overset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \overset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

We can look at **Σ** to see that the first column has a large effect

while the second column has a much smaller effect in this example

Image compression

- For this image, using **only the first 16** of 300 principal components produces a recognizable reconstruction
- Using the first 64 almost perfectly reconstructs the image

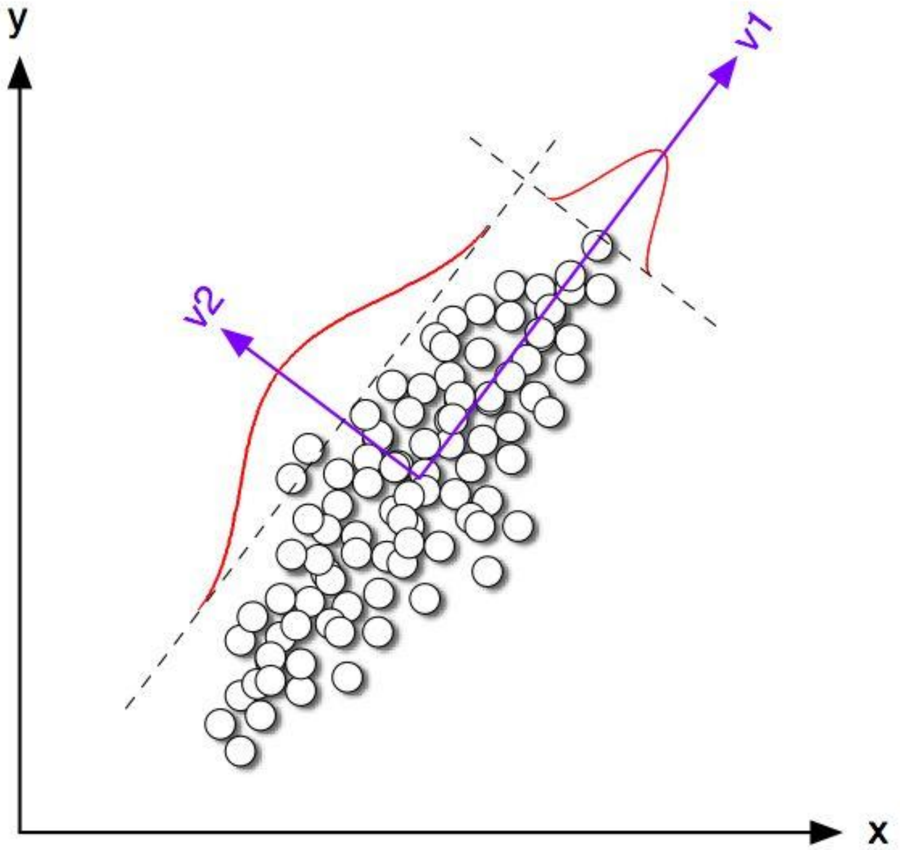


- Principal Components Analysis
    - Used HEAVILY in computer vision

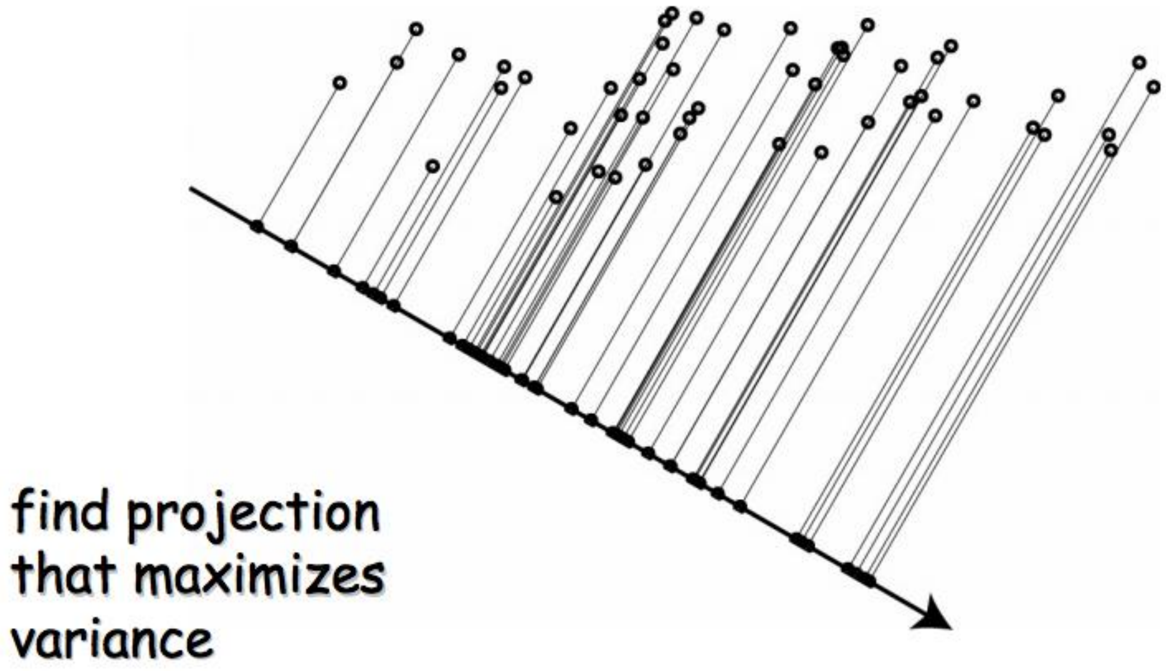
Intuition behind PCA: high dimensional data usually lives in some lower dimensional space

Covariance between the two dimensions of features is high.

Can we reduce the number of dimensions to just 1?

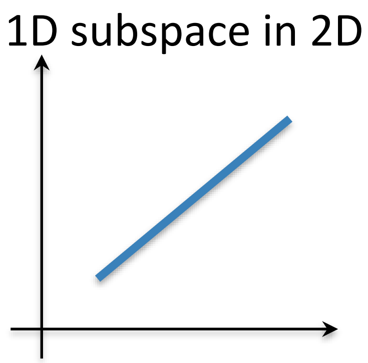


Geometric interpretation of PCA



Geometric interpretation of PCA

- Let’s say we have a set of 2D data points x. But we see that all the points lie on a line in 2D.
- So, 2 dimensions are redundant to express the data. We can express all the points with just one dimension.



PCA: Principal Component Analysis

- Given a dataset of images, can we compressed them like we can compress a single image?
  - Yes, the trick is to look into the correlation between the dimensions of the image
  - The tool for doing this is called PCA

PCA can be used to compress image RGB pixel values or also be used to compress their features!

Covariance between 2 Random Variables

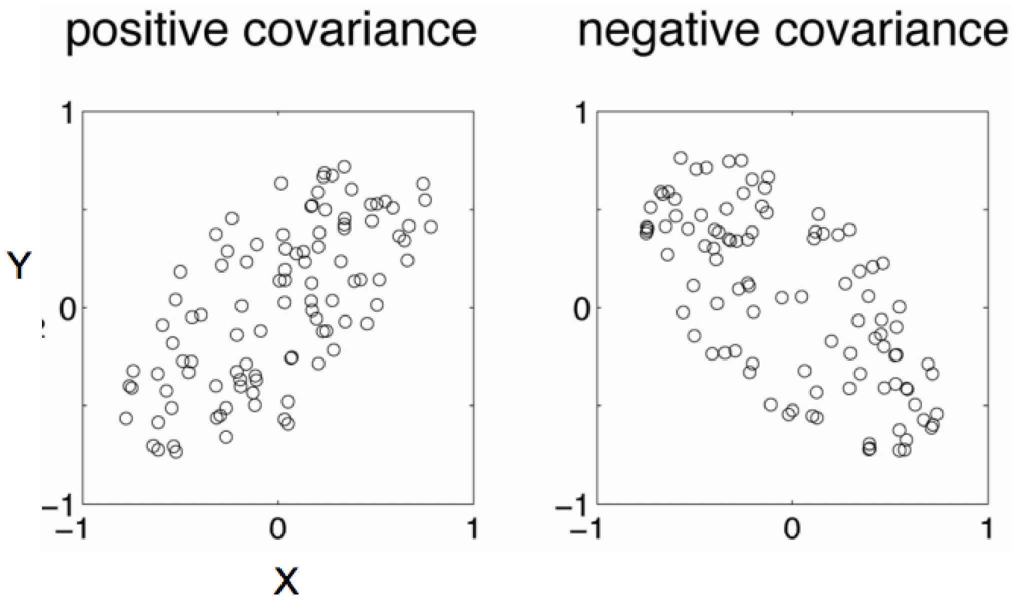
- $\mathrm{Cov}(X, Y) = 1/n \sum_{i=1,n} (x_i - E(X))\,(y_i - E(Y))$

Toy example to explain covariance

- What is covariance between dimensions?
- Let's say we have a dataset of students
  - each student is represented with 3 dimensions
  - **x**: number of hours studied for a class
  - **y**: grades obtained in that class
  - **z**: number of lectures attended
- covariance value between **x and y is say: 104.53**
  - what does this value mean?

Covariance interpretation

- ○ **x**: number of hours studied for a subject
- ○ **y**: marks obtained in that subject
- covariance value between **x and y is say: 104.53**
  - ○ what does this value mean?



Visualizing this covariance matrix

- We can represent these covariance correlation numbers in a matrix
- e.g. for 3 dimensions:

$$C = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix}$$

Variances

- Diagonal is the **variances** of x, y and z
- **cov(x,y) = cov(y,x)** hence **C is symmetrical** about the diagonal
- N-dimensional data will result in NxN covariance matrix

Covariance interpretation

- Exact value is not as important as it’s sign.
- A **positive value** of covariance indicates both dimensions increase or decrease together e.g. as the number of hours studied increases, the marks in that subject increase.
- A **negative value** indicates while one increases the other decreases, or vice-versa e.g. active social life at PSU vs performance in CS dept.
- If **covariance is zero**: the two dimensions are independent of each other e.g. heights of students vs the marks obtained in a subject

PCA by SVD

- To relate this to PCA, we consider the image (or feature) matrix

Here each x_i is an image converted to a column vector.

$$X = \begin{bmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{bmatrix}$$

So this is a dataset of images.

- The sample mean of this dataset (or in plain english, the average image) is:

$$\mu = \frac{1}{n}\sum_i x_i = \frac{1}{n}\begin{bmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{bmatrix}\begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = \frac{1}{n}X1$$

This just sums up the rows of X and divides by n to get the average.

PCA by SVD

- Center the data by subtracting the mean to each column of X
- The centered dataset matrix is

$$X_c = \begin{bmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{bmatrix} - \begin{bmatrix} | & & | \\ \mu & \dots & \mu \\ | & & | \end{bmatrix}$$

PCA by SVD

- The sample covariance matrix is

$$C = \frac{1}{n}\sum_i (x_i - \mu)(x_i - \mu)^T = \frac{1}{n}\sum_i x_i^c \left(x_i^c\right)^T$$

where x_i^c is the i^{th} column of X_c

- This can be written as

$$C = \frac{1}{n}\begin{bmatrix} | & & | \\ x_1^c & \ldots & x_n^c \\ | & & | \end{bmatrix}\begin{bmatrix} - & x_1^c & - \\ & \vdots & \\ - & x_n^c & - \end{bmatrix} = \frac{1}{n} X_c X_c^T$$

PCA by SVD

- The matrix

$$X_c^T = \begin{bmatrix} - & \mathbf{x}_1^c & - \\ & \vdots & \\ - & \mathbf{x}_n^c & - \end{bmatrix}$$

is real (n x d). Assuming n>d it has SVD decomposition

$$X_c^T = U\Sigma V^T \qquad U^T U = I \qquad V^T V = I$$

and

$$C = \frac{1}{n} X_c X_c^T$$

Calculating covariance matrix

$$
\begin{aligned}
C &= \frac{1}{n} X_c X_c^T \\
&= \frac{1}{n} U \Sigma V^T (U \Sigma V^T)^T \\
&= \frac{1}{n} U \Sigma V^T V \Sigma U^T \\
&= \frac{1}{n} U \Sigma^2 U^T
\end{aligned}
$$

PCA by SVD

$$C = \frac{1}{n} U \Sigma^2 U^T$$

- Note that U is (d x d) and orthonormal, and Σ^2 is diagonal. **This is just the eigenvalue decomposition of C**
- This means that we can calculate the eigenvectors of C using the eigenvectors of X_c
- It follows that
  - The eigenvectors of C are the columns of U
  - The eigenvalues of C are the diagonal entries of Σ^2: λ_i^2

PCA by SVD

- In summary, computation of PCA by SVD
- Given X with one image (or feature) per column
  - Create the centered data matrix

$$X_c = \begin{bmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{bmatrix} - \begin{bmatrix} | & & | \\ \mu & \dots & \mu \\ | & & | \end{bmatrix}$$

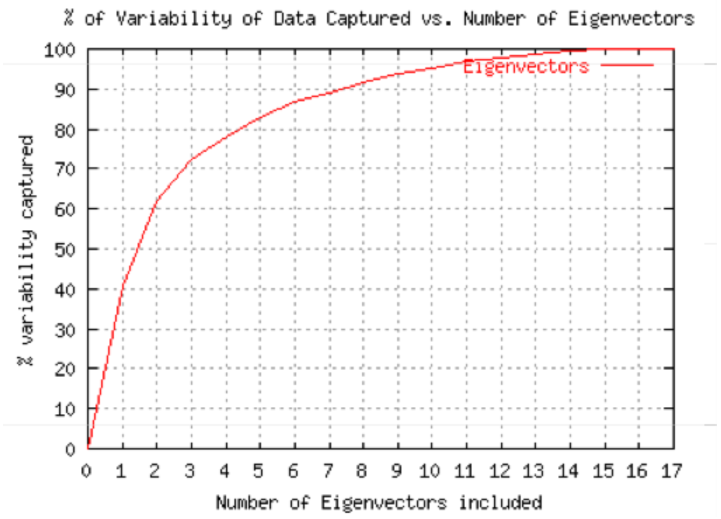
  - Compute its SVD

$$X_c^T = U \Sigma V^T$$

  - Principal components of the covariance matrix C are columns of U

To compress an image dataset, pick the largest eigenvalues and their corresponding eigenvectors

- Pick the eigenvectors that explain p% of the image data variability
  - Can be done by plotting the ratio r_k as a function of k



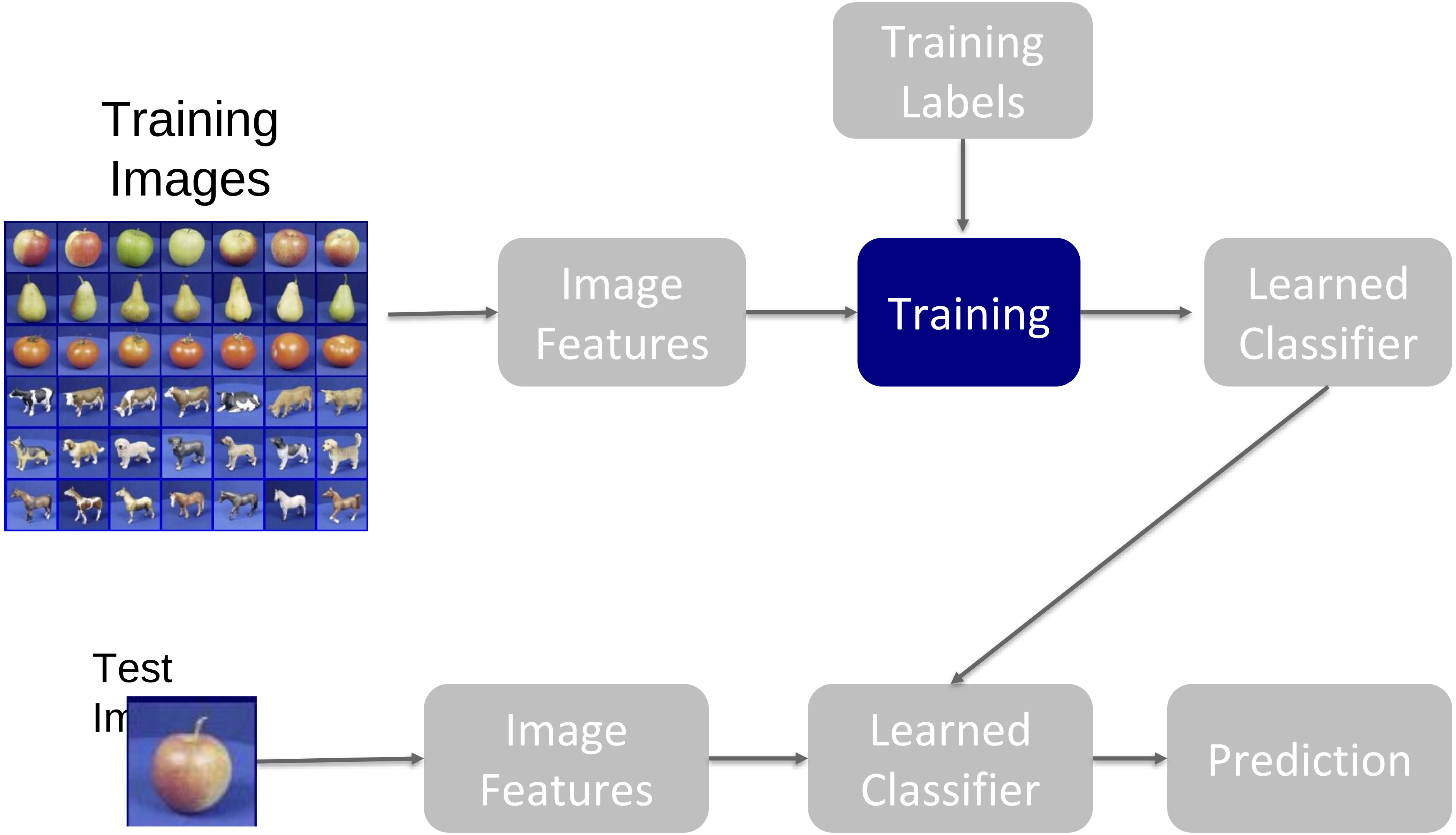
$$r_k = \sum_{i=1}^{\boxed{k}} \lambda_i^2 \Big/ \sum_{i=1}^{\boxed{n}} \lambda_i^2$$

  - E.g. we need k=3 eigenvectors to cover 70% of the variability of this dataset

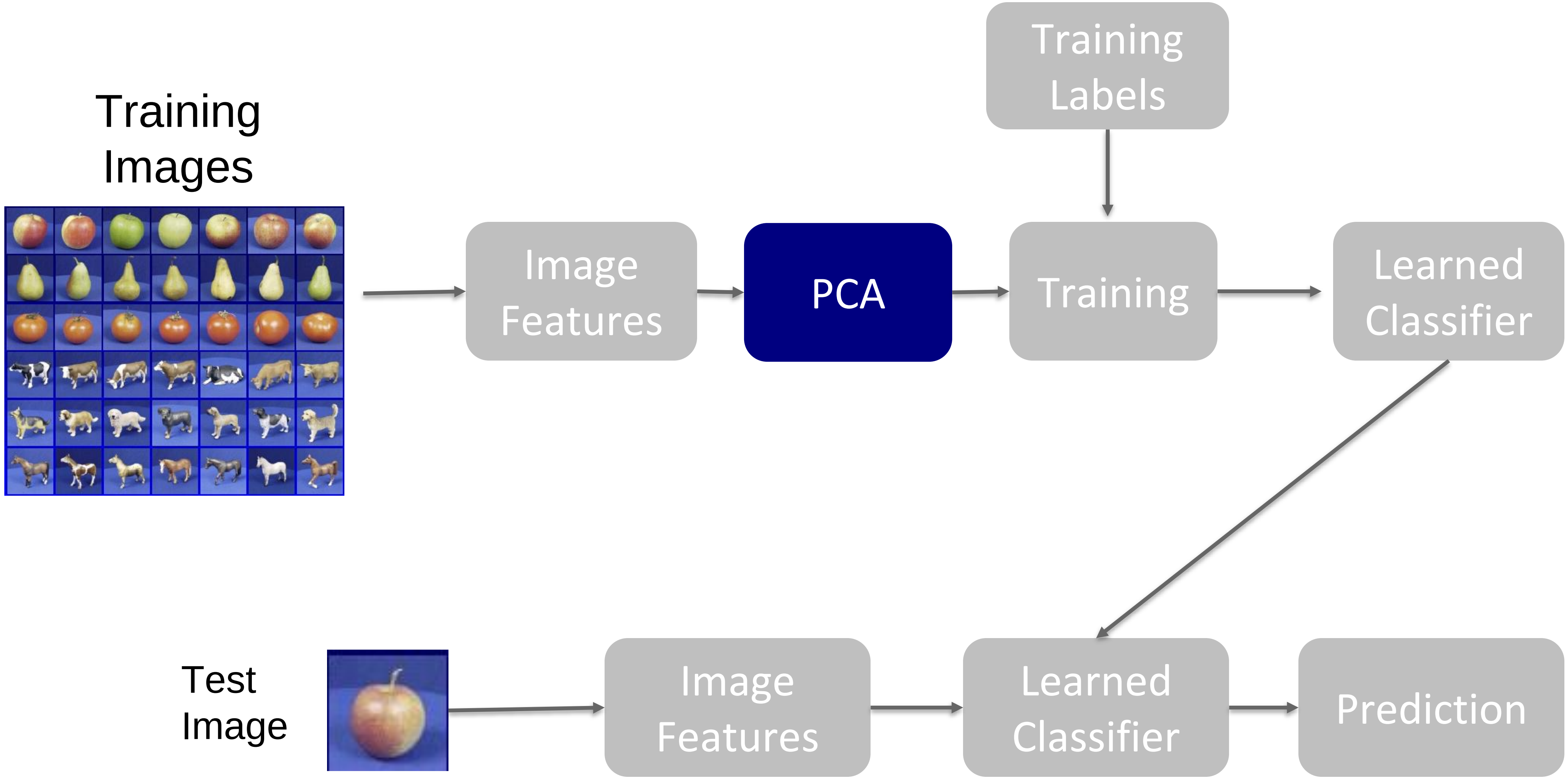
What exactly is the covariance

- Variance and Covariance are a measure of the **“spread”** of a set of points around their center of mass (mean)
- **Variance** – measure of the deviation from the mean for points in one dimension e.g. heights
- **Covariance** as a measure of how much each of the dimensions vary from the mean with respect to each other.
- Covariance is measured between 2 dimensions to see if there is a relationship between the 2 dimensions e.g. number of hours studied & marks obtained.
- The covariance between one dimension and itself is the variance

What happens with PCA during training?



What happens with PCA during training?



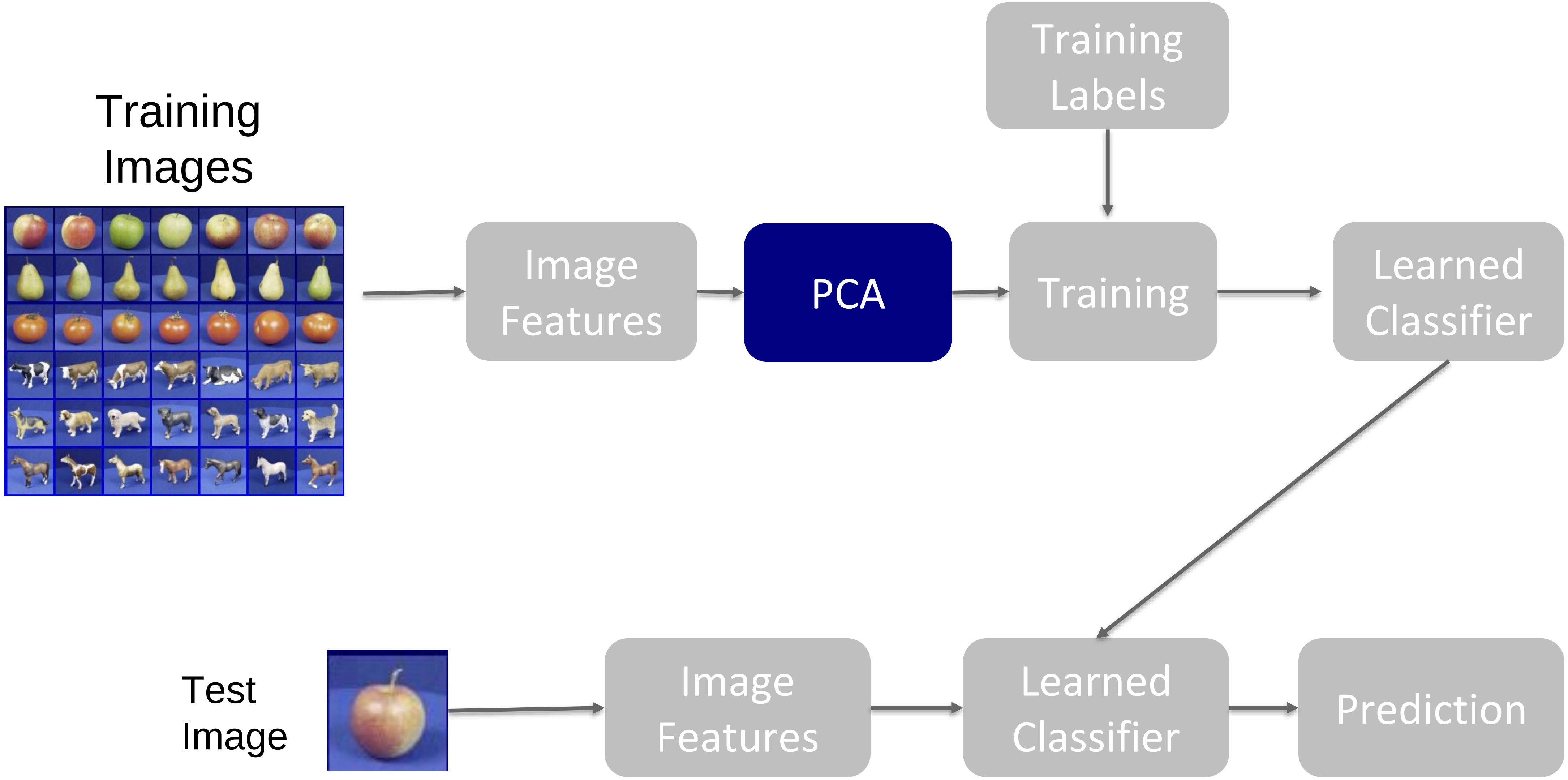
PCA during training

Let's say that we choose k top eigenvalues and their corresponding eigenvectors: $[u_1, \ldots, u_k]$

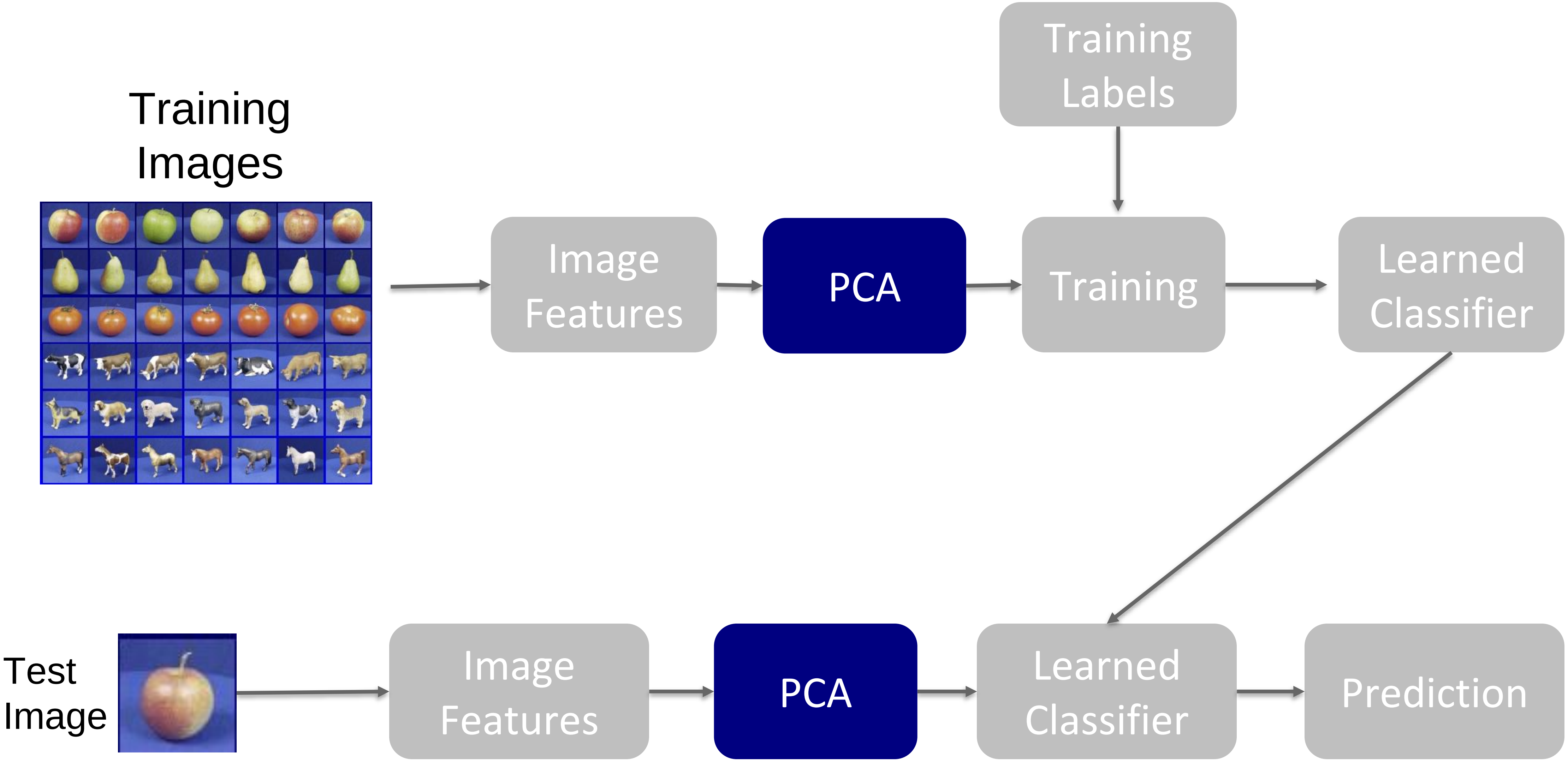
Replace all image features x with:

$$\hat{x} = \begin{bmatrix} u_1^T x \\ u_2^T x \\ \ldots \\ u_k^T x \end{bmatrix}$$

What happens with PCA during testing?



What happens with PCA during testing?

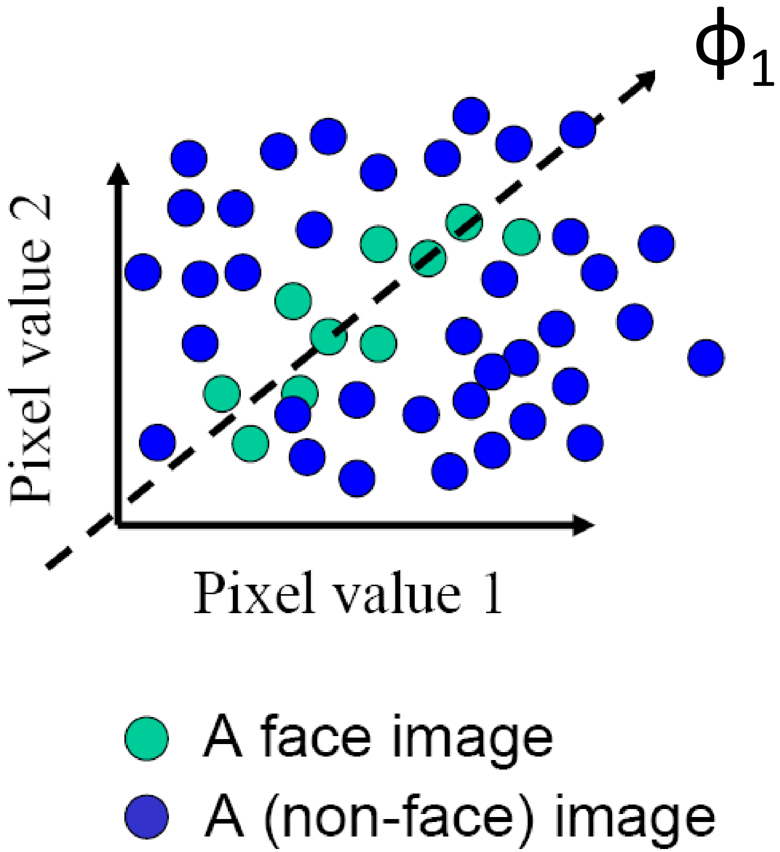


How PCA was originally used in vision: To identify celebrities using their faces

- An image is a point in a high dimensional space
  - In grayscale, an N x M image is a point in R^{NM}
  - E.g. 100x100 images lives in a 10,000-dimensional space

100x100 images can contain many things other than faces!

The Space of Faces



- However, relatively few high dimensional vectors correspond to valid face images
- We want to effectively model the subspace of face images

This is where PCA comes in

Slide credit: Chuck Dyer, Steve Seitz, Nishino

Eigenfaces: an algorithm using PCA to reduce the space of faces

- Assume that most face images lie on a low-dimensional subspace determined by the first k (k<<d) eigenvectors of a dataset of faces
- To demonstrate the effectiveness of PCA for images, they called each eigenvector of a dataset “eigenfaces”
- Represent all face images in the dataset as linear combinations of eigenfaces

M. Turk and A. Pentland, Face Recognition using Eigenfaces, CVPR 1991

Training images: $\mathbf{x}_1,\ldots,\mathbf{x}_N$

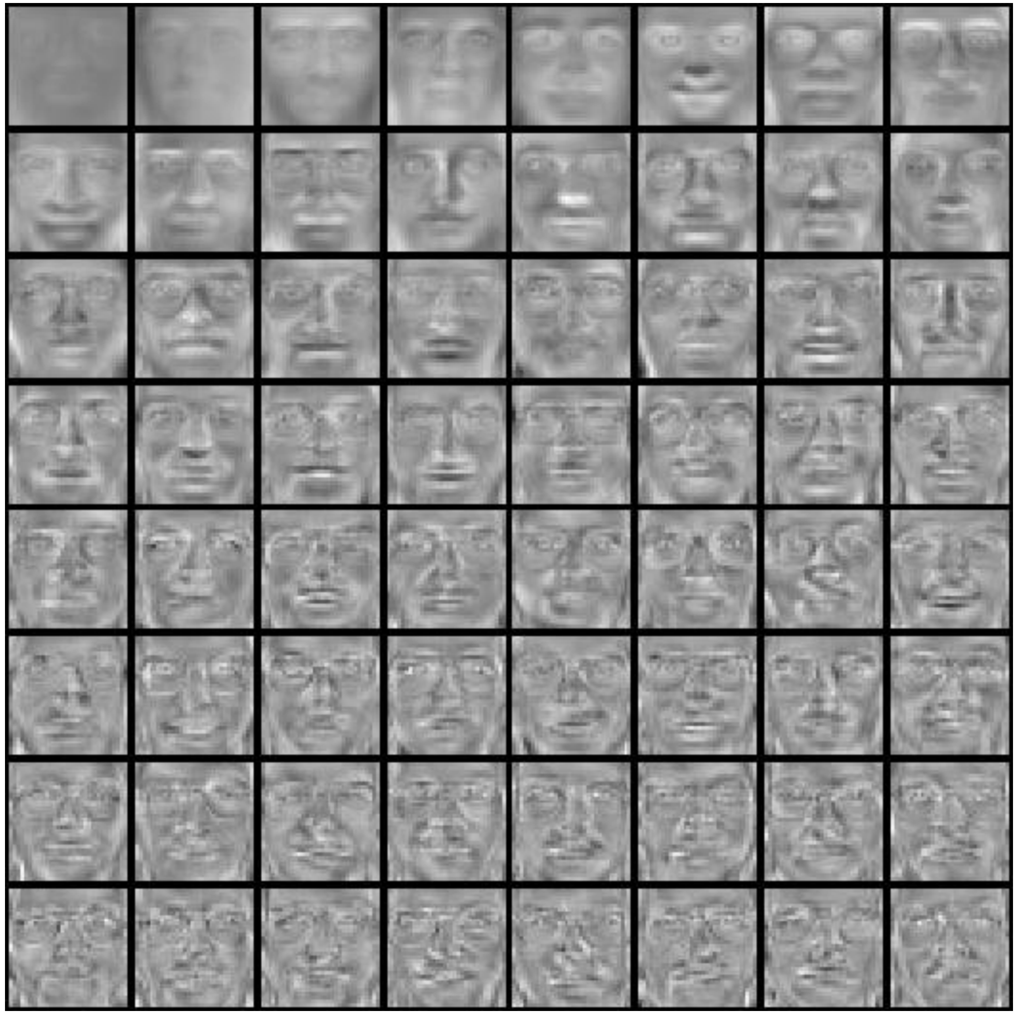
Each 100x100 image is going to be represented as a 10,000-dimensional vector

$$X = \begin{bmatrix} | & & | \\ x_1 & \ldots & x_n \\ | & & | \end{bmatrix}$$

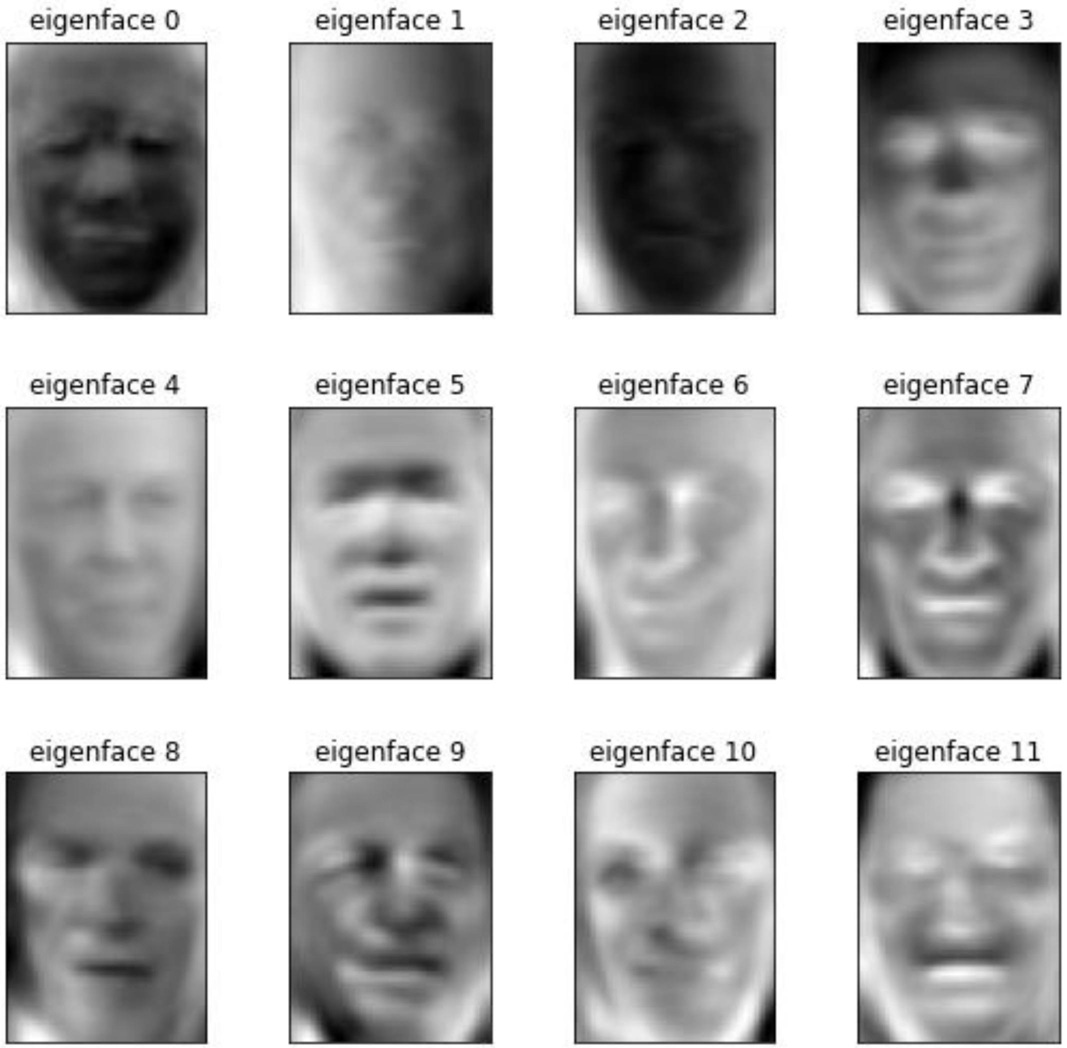
Top eigenvectors: $U_1, \ldots, U_k$

Mean: μ

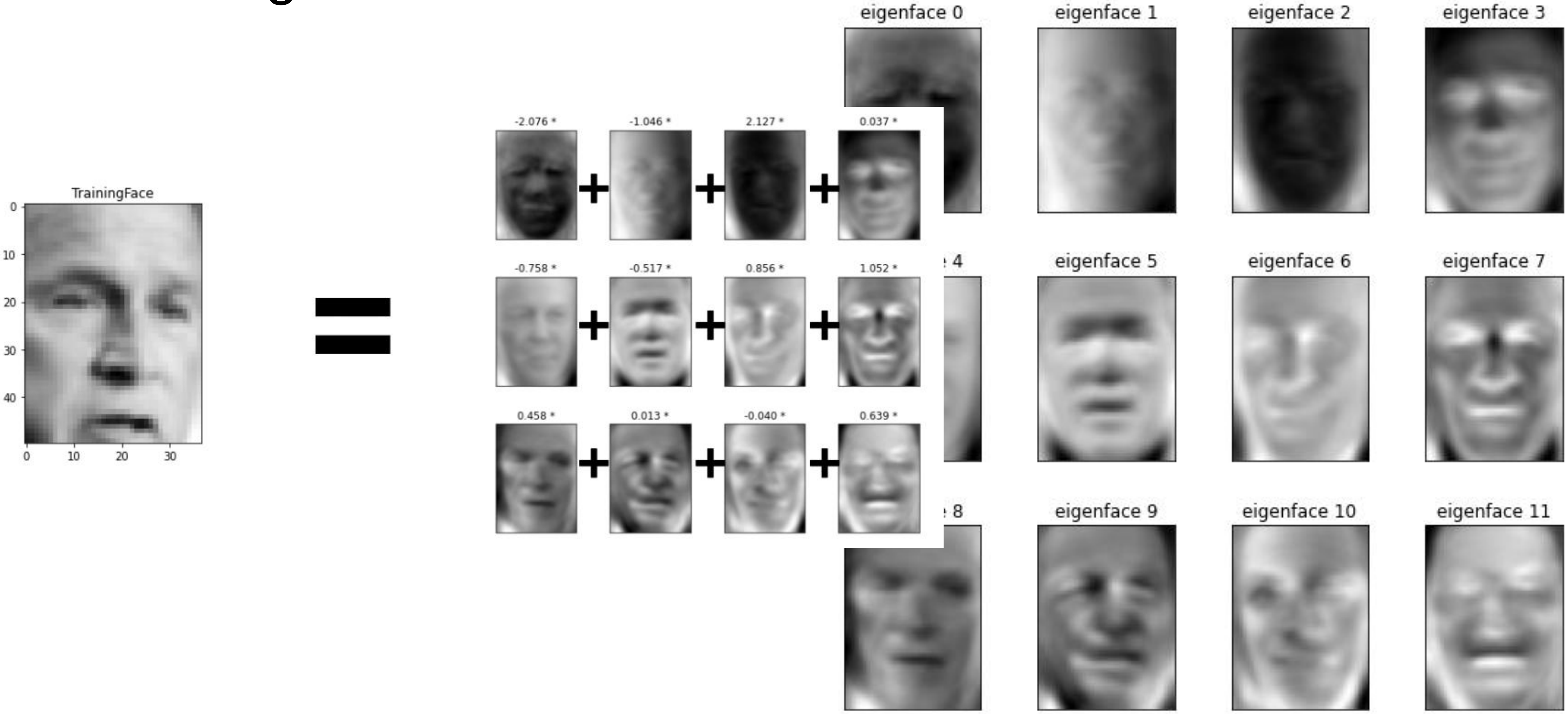
$$\mu = \frac{1}{n}\sum_i x_i$$



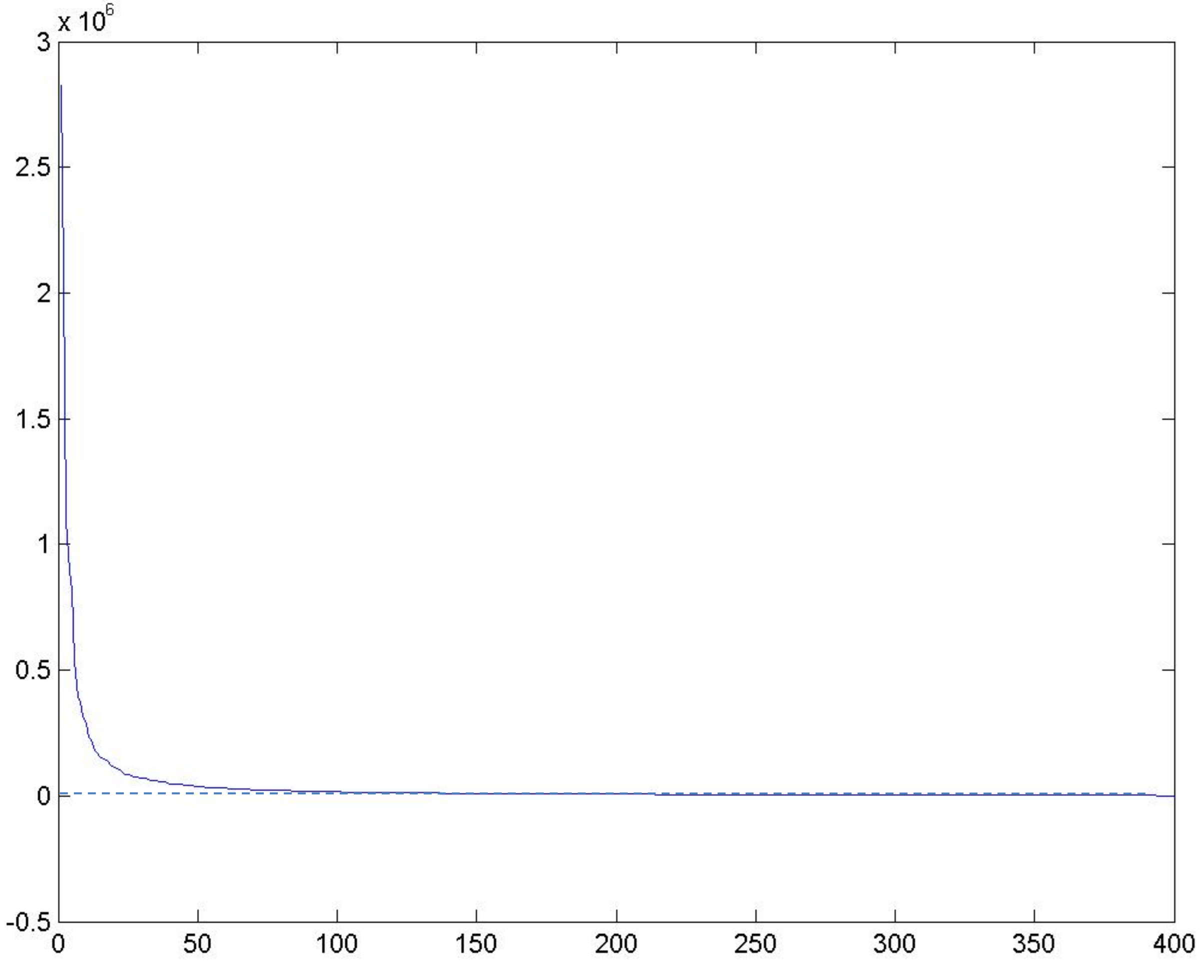
Calculate its SVD and visualize its top eigenvectors



Every image can be reconstructed as a linear combination of these eigenvectors



Error rate when reconstructing a face decreases as you use more eigenvectors



Reconstruction and Errors

K = 4

K = 200

K = 400

- Fewer eigenfaces result in more information loss, and hence less discrimination between faces.

Using PCA for classifying faces

- Training
  1. Place all training images $x_1, x_2, \ldots, x_N$ into a matrix
  2. Compute average face
  3. Compute the difference image (the <u>centered data matrix</u>)

$$X_c = \begin{bmatrix} | & & | \\ x_1 & \ldots & x_n \\ | & & | \end{bmatrix} - \begin{bmatrix} | & & | \\ \mu & \ldots & \mu \\ | & & | \end{bmatrix}$$

  4. Use SVD to find the eigenvectors of the covariance matrix

$$X_c^T = U\Sigma V^T$$

  1. Keep the top-K eigenvalues and their eigenvectors
  2. Compute each training image x_i ‘s new projected features: $\hat{x} = \begin{bmatrix} u_1^T x \\ u_2^T x \\ \ldots \\ u_k^T x \end{bmatrix}$

Using PCA for classifying faces

- Testing
  1. Given a test image x_{test}
  2. Project x into this new space into eigenface space:

$$\hat{x}_{test} = \begin{bmatrix} u_1^T x_{test} \\ u_2^T x_{test} \\ \dots \\ u_k^T x_{test} \end{bmatrix}$$

  1. Run your classifier on this new space.
     - For example, use k-NN using distance measures (Euclidean) in this new space

Offline Training Phase:
Input a set I of M labeled training images and produce
a basis set B and a vector of coefficents for each image.

$I = \{I_1, I_2, \ldots, I_M\}$ is the set of training images. (input)
$B = \{F_1, F_2, \ldots, F_m\}$ is the set of basis vectors. (output)
$A_j = [a_{j1}, a_{j2}, \ldots, a_{jm}]$ is the vector of coefficients for image I_j. (output)

1. $I_{mean} = mean(I)$.
2. $\Phi = \{\Phi_i | \Phi_i = I_i - I_{mean}\}$, the set of difference images
3. Σ_Φ = the covariance matrix obtained from Φ.
4. Use the principal components method to compute eigenvectors and eigenvalues of Σ_Φ. (see text)
5. Construct the vector **B** as the basis set by selecting the most significant m eigenvectors; start from the largest eigenvalue and continue in decreasing order of the eigenvalues to select the corresponding eigenvectors.
6. Represent each training image I_j by a linear combination of the basis vectors: $I_j^m = a_{j1}F_1 + a_{j2}F_2 + \ldots + a_{jm}F_m$

Online Recogniton Phase:
Input the set of basis vectors B, the database of coefficient sets $\{A_j\}$, and a test image I_u. Output the class label of I_u.

1. Compute vector of coefficients $A_u = [a_{u1}, a_{u2}, \ldots, a_{um}]$ for I_u;
2. Find the h nearest neighbors of vector A_u in the set $\{A_j\}$;
3. Decide the class of I_u from the labels of the h nearest neighbors (possibly reject in case neighbors are far or inconsistent in labels);

Algorithm 4: Recognition-by-Appearance using a Basis of Principal Components.

Shortcomings

- Requires carefully curated training data:
  - All faces centered in frame
  - All faces have to be the same size
  - Some sensitivity to angle (ideally all faces are facing front)
- Alternative:
  - “Learn” one set of PCA vectors for each angle
  - Use the one with lowest error
- Method is completely knowledge free
  - (sometimes this is good!)
  - Doesn’t know that faces 2D projections of 3D heads
  - But it also makes no effort to preserve what makes a “face” a “face”

Summary for Eigenface

Pros

- Non-iterative, globally optimal solution

Cons:

- PCA projection is **optimal for reconstruction** from a low dimensional basis, but **may NOT be optimal for recognition**
- Is there a better dimensionality reduction?
- There is LDA.

What we have learned today?

- Introduction to recognition
- A simple Object Recognition pipeline
- Choosing the right features
- A training algorithm: kNN
- Testing an algorithm
- Challenges with kNN
- Dimensionality reduction

Next lecture

BOW and Detection