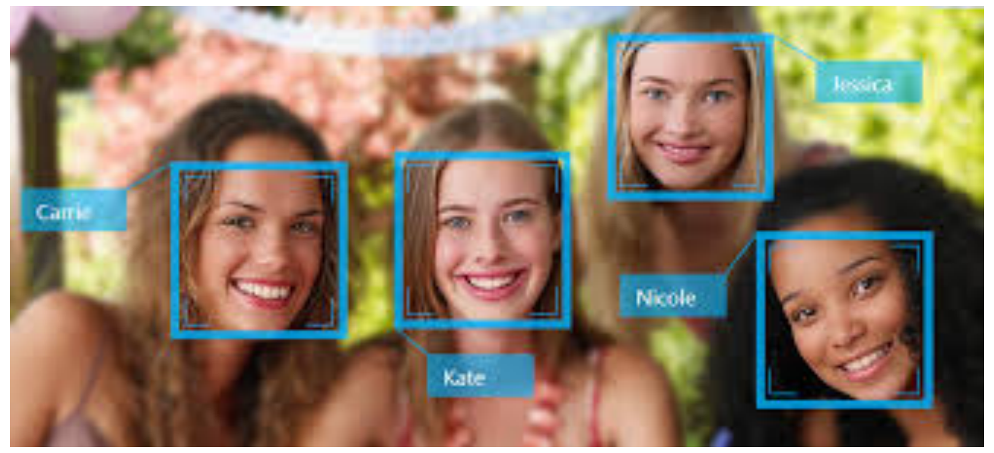
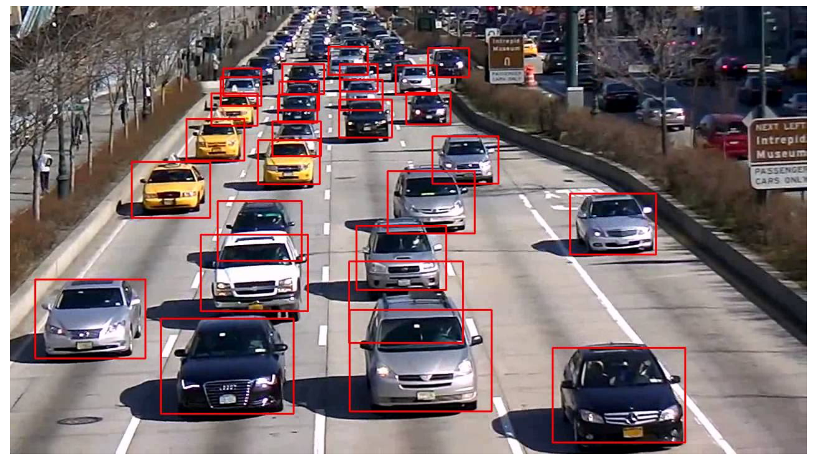
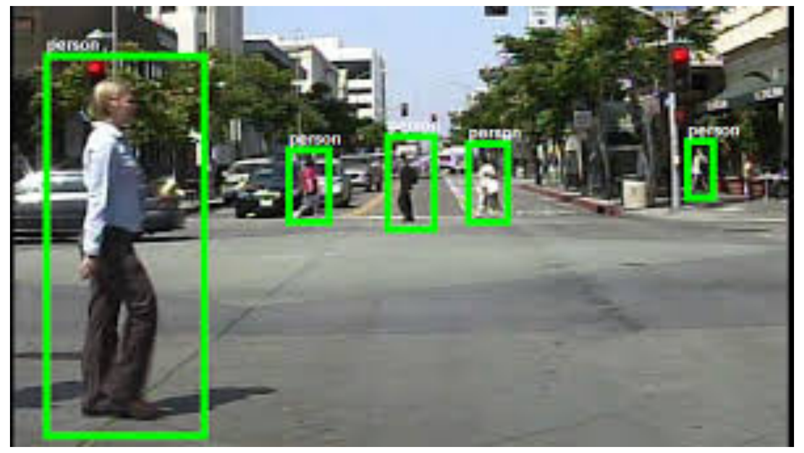
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 19: Classification (part2)

Menu for Today

Topics:

— Scene Classification

— Bag of Words Representation

— Decision Tree

— Boosting

Redings:

— **Today's** Lecture: Forsyth & Ponce (2nd ed.) 16.1.3, 16.1.4, 16.1.9

— **Next** Lecture: Forsyth & Ponce (2nd ed.) 17.1–17.2

Reminders:

Lecture 18: Re-cap (Image Classification)

Classify images containing single **objects**, the same techniques can be applied to classify natural **scenes** (e.g. beach, forest, harbour, library).

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

→ cat

Lecture 18: Image Classification

Representation of Images

— Image pixels directly

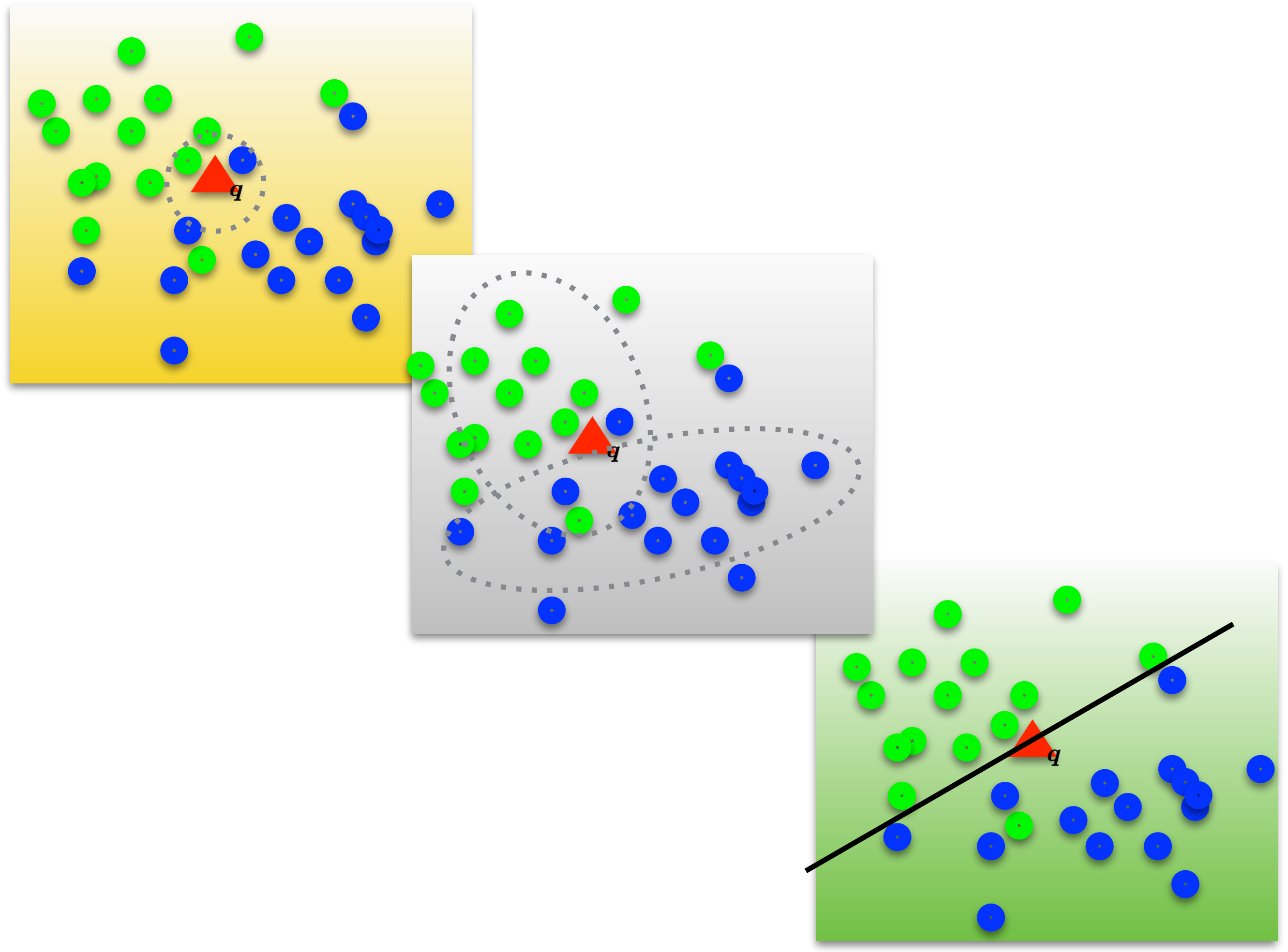
— Bag of Words

Classification Algorithms

— Bayes' Classifier

— Nearest Neighbor Classifier

— SVM Classifier



Lecture 18: Re-cap (Vector Space Model)

Many algorithms for image classification accumulate evidence on the basis of **visual words**.

To classify a text document (e.g. as an article on sports, entertainment, business, politics) we might find patterns in the occurrences of certain words.

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

Dictionary Learning:
Learn Visual Words using clustering

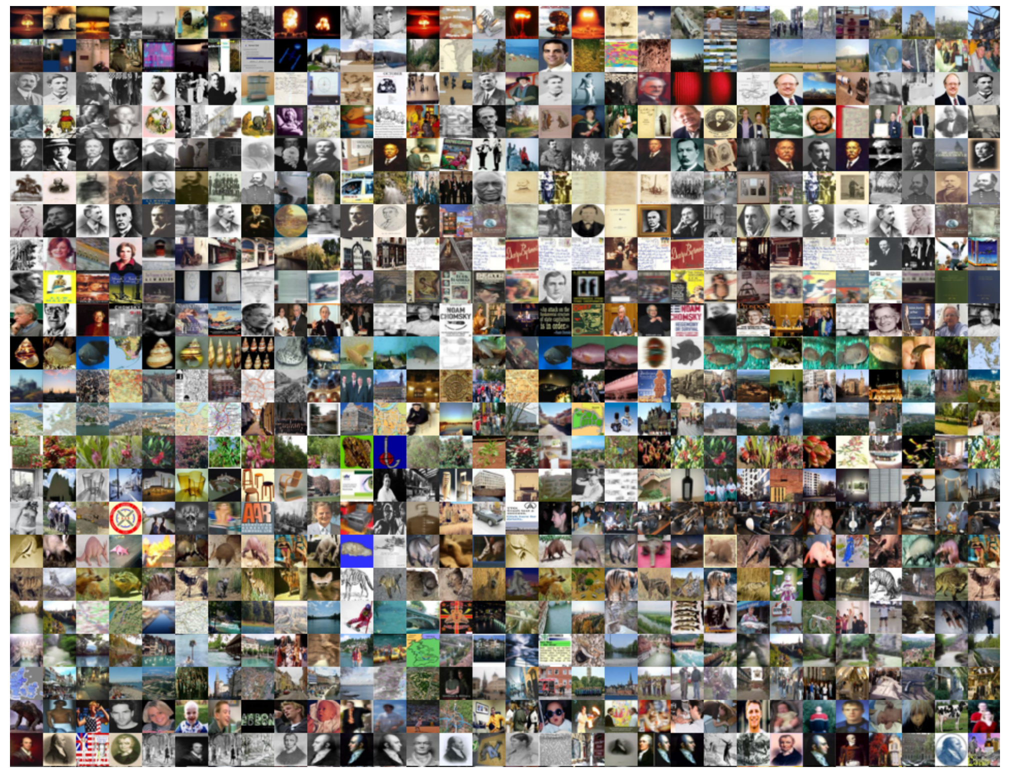
Encode:
build Bags-of-Words (BOW) vectors
for each image

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)

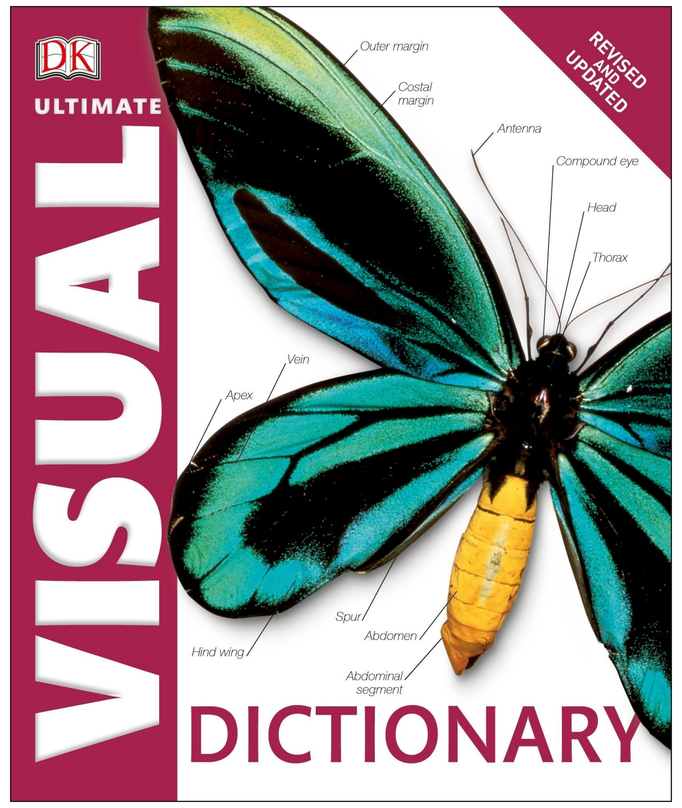


→

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

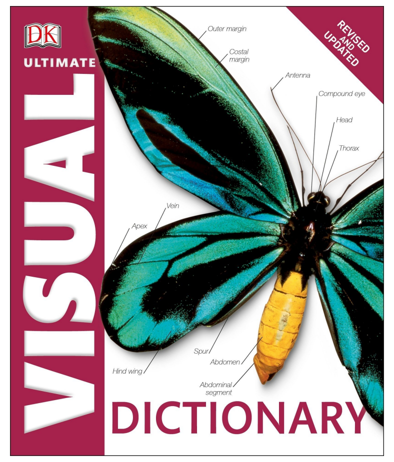
Input: training images, dictionary

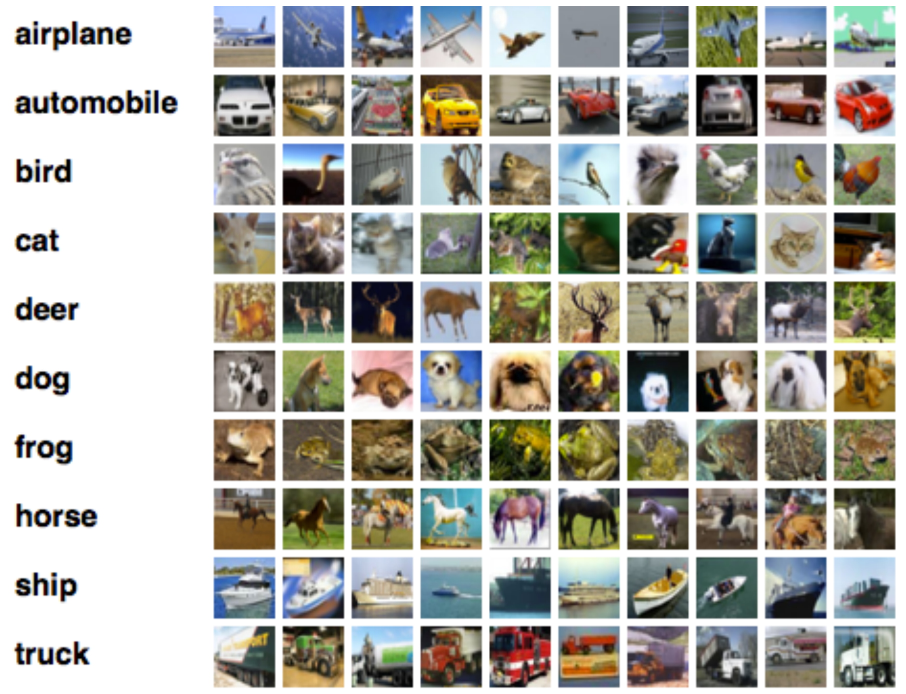
→

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

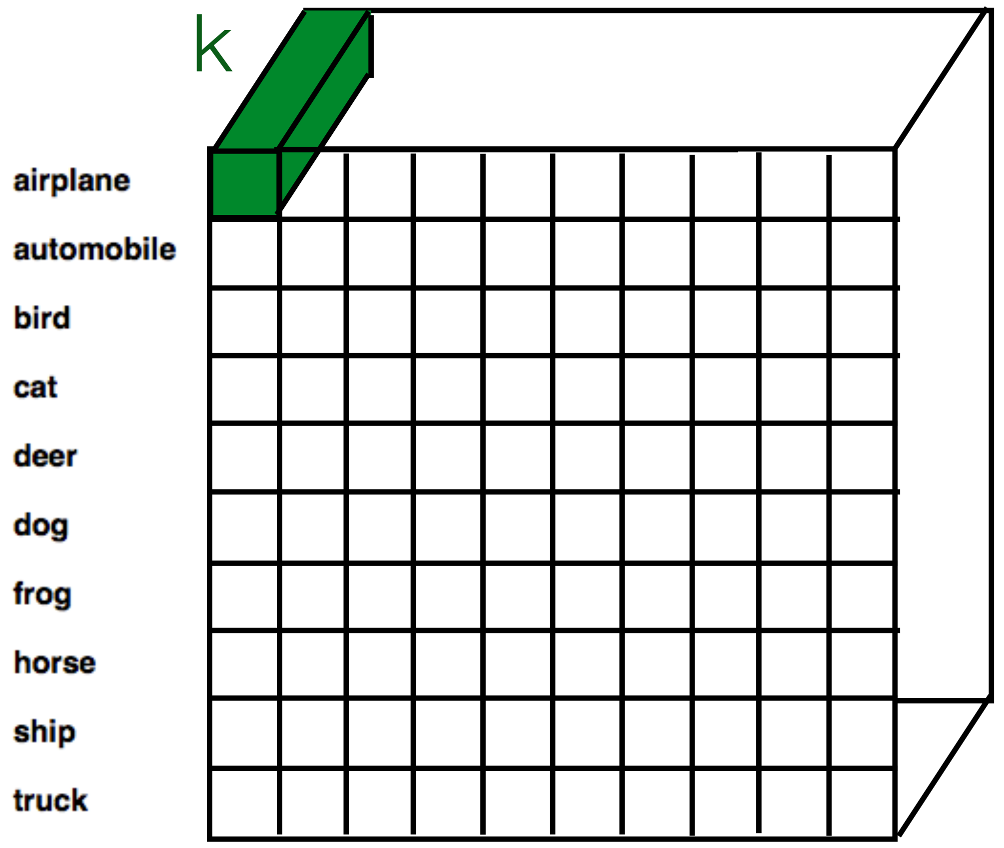
→

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

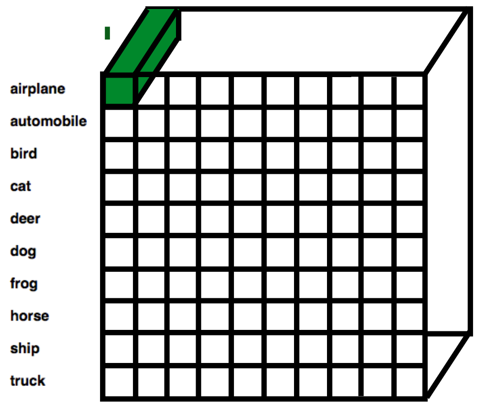
→ **Encode**:
build Bags-of-Words (BOW) vectors
for each image →

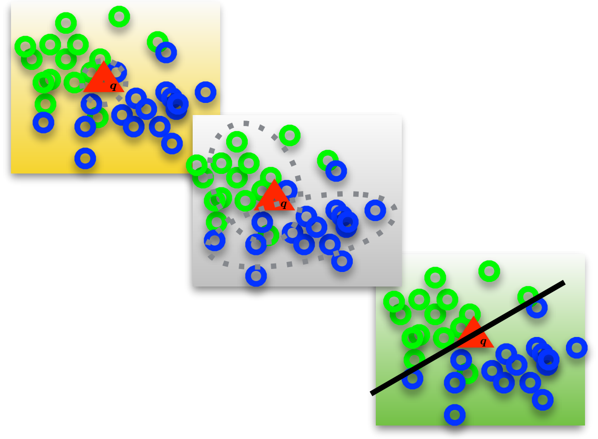
Output: histogram representation for each training image

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)

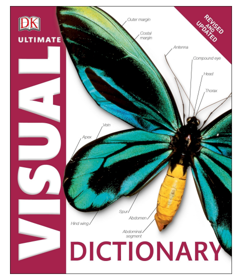
→

Dictionary Learning:
Learn Visual Words using clustering

→

Output: dictionary of visual words

Input: test image, dictionary

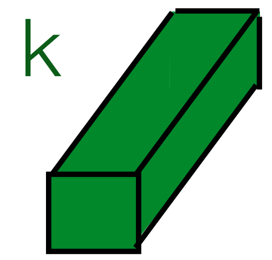


→

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)

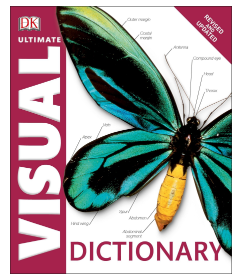
→

Dictionary Learning:
Learn Visual Words using clustering

→

Output: dictionary of visual words

Input: test image, dictionary



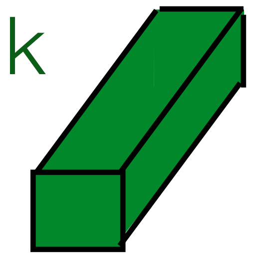
→

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

→

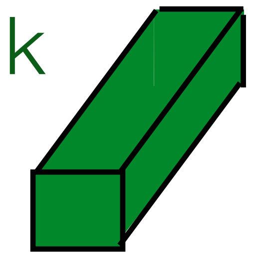
Output: histogram representation for test image

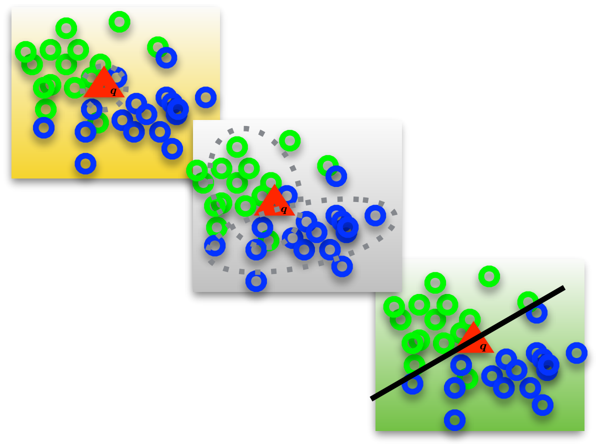
k



Input: histogram representation for test image, trained classifier

k





→

Classify:
Test data using BOWs

→

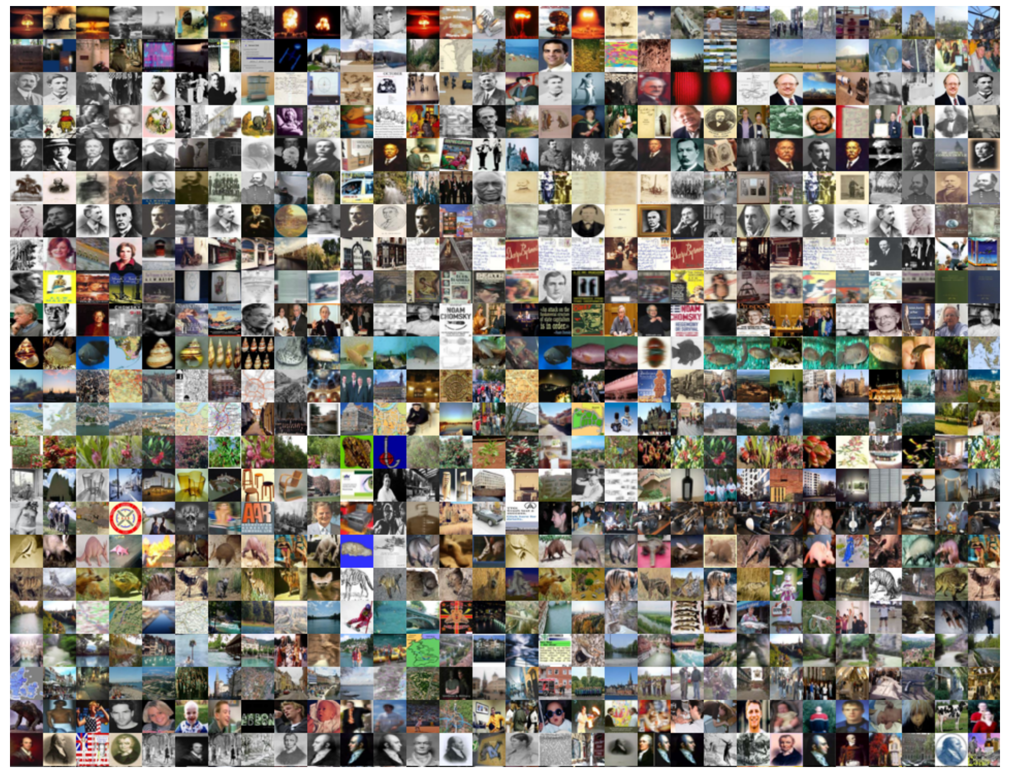
Output: prediction for test image

airplane

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

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

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

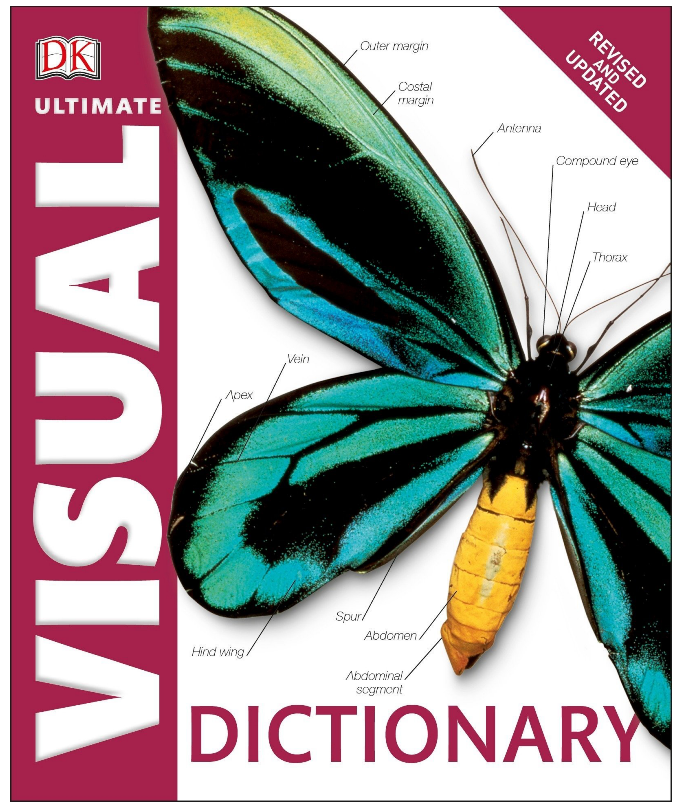


→

Dictionary Learning:
Learn Visual Words using clustering

→

Output: dictionary of visual words

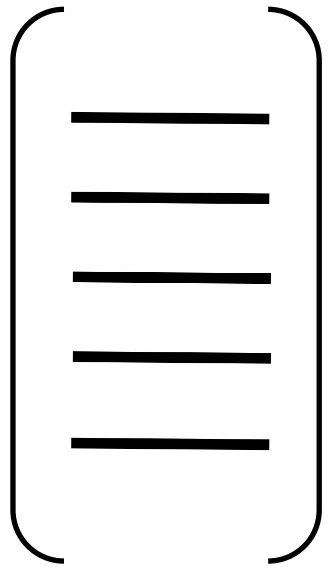


Encode:
build Bags-of-Words (BOW) vectors
for each image

Classify:
Train data using BOWs

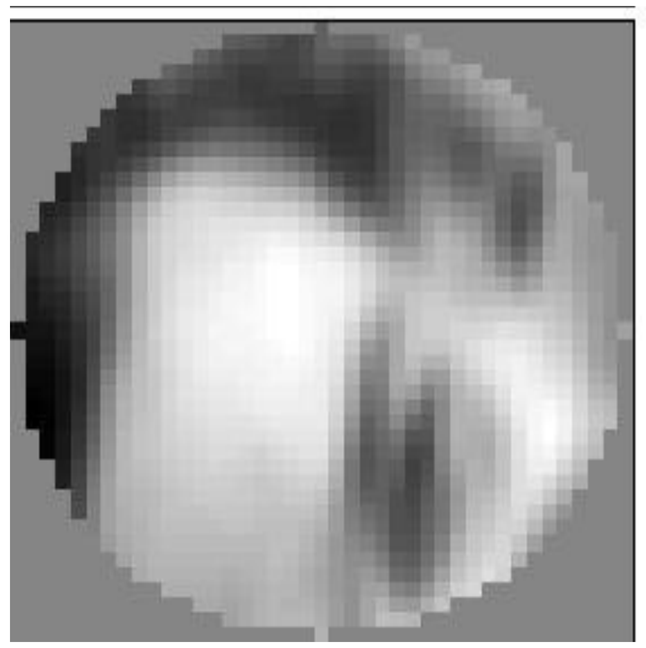
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Extracting **SIFT** Patches

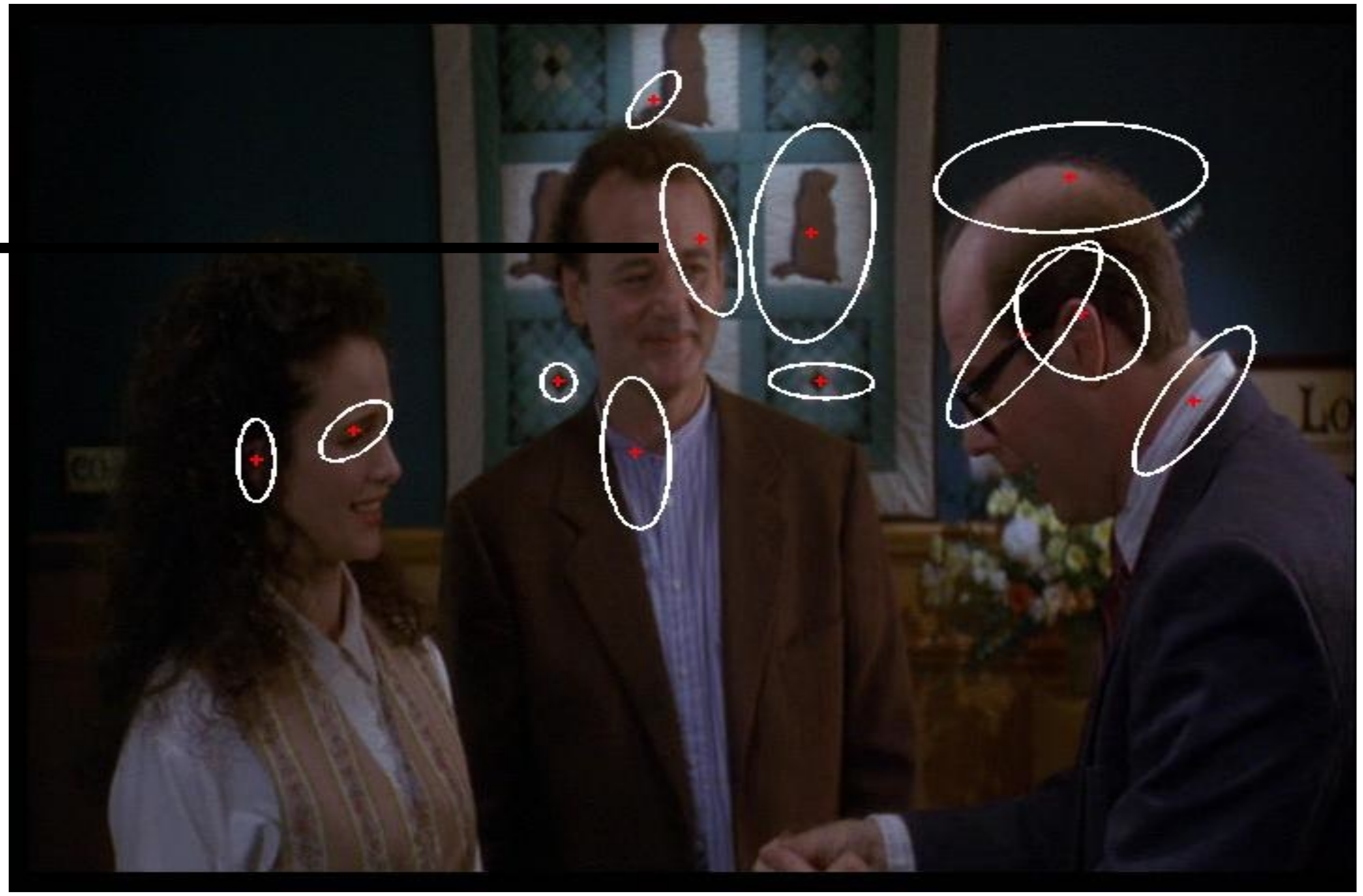


Compute SIFT descriptor

[Lowe'99]



Normalize patch



Detect patches

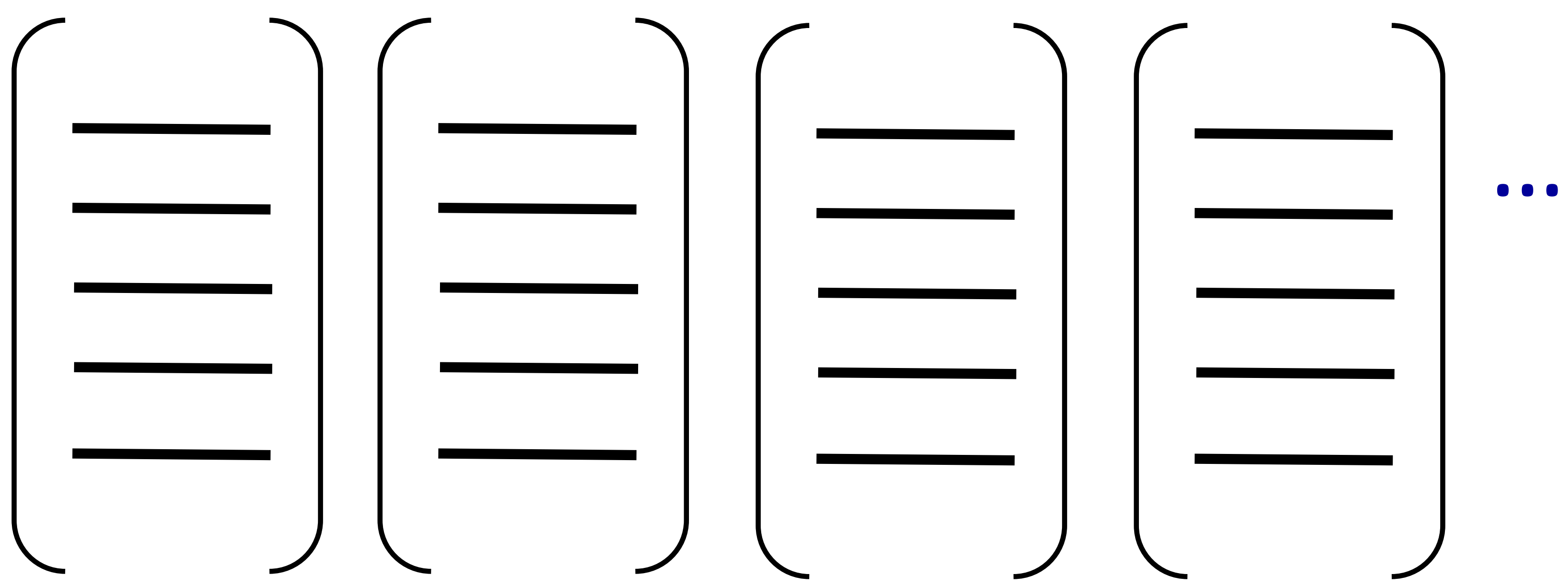
[Mikojaczyk and Schmid '02]

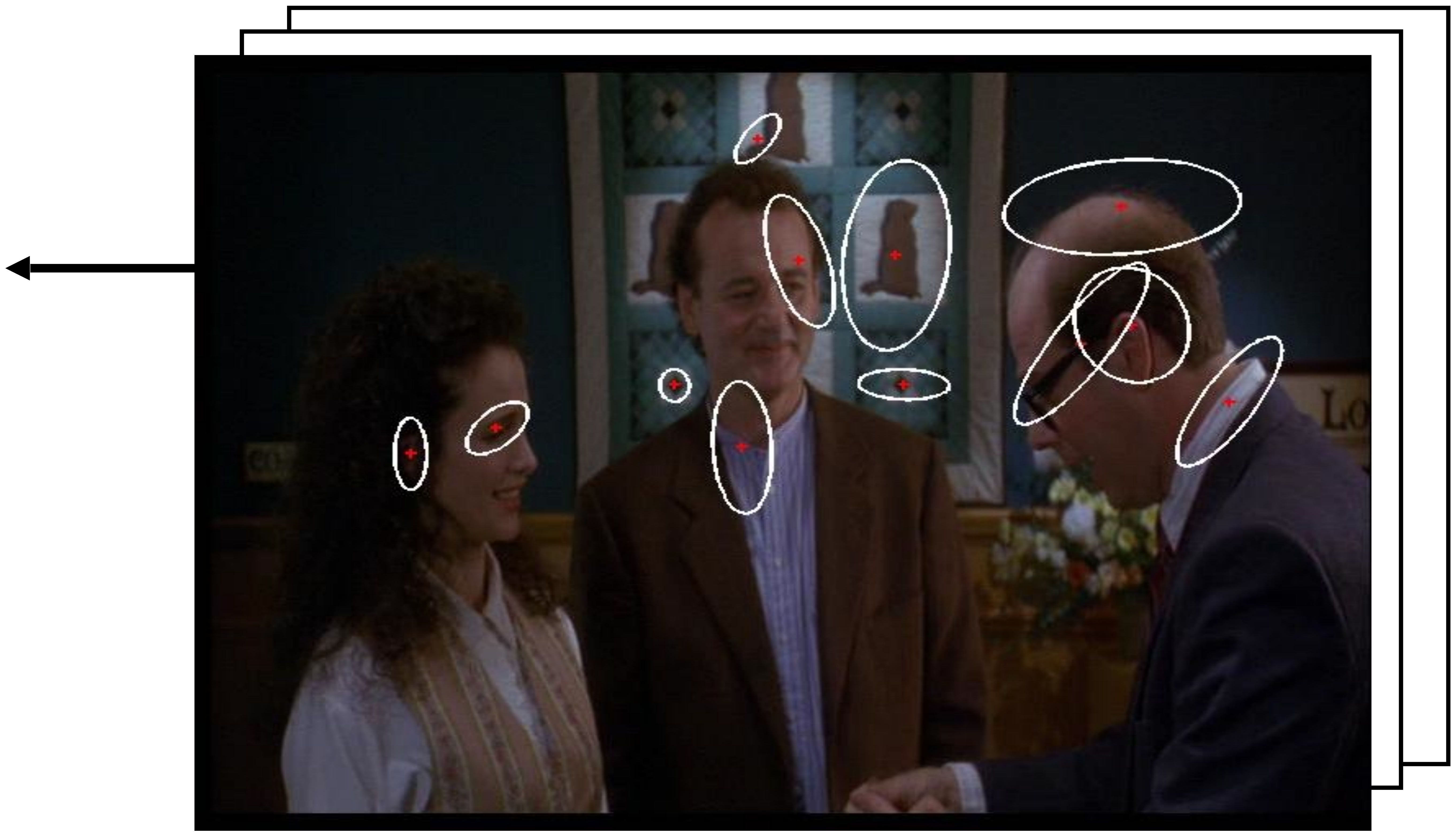
[Mata, Chum, Urban & Pajdla, '02]

[Sivic & Zisserman, '03]

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

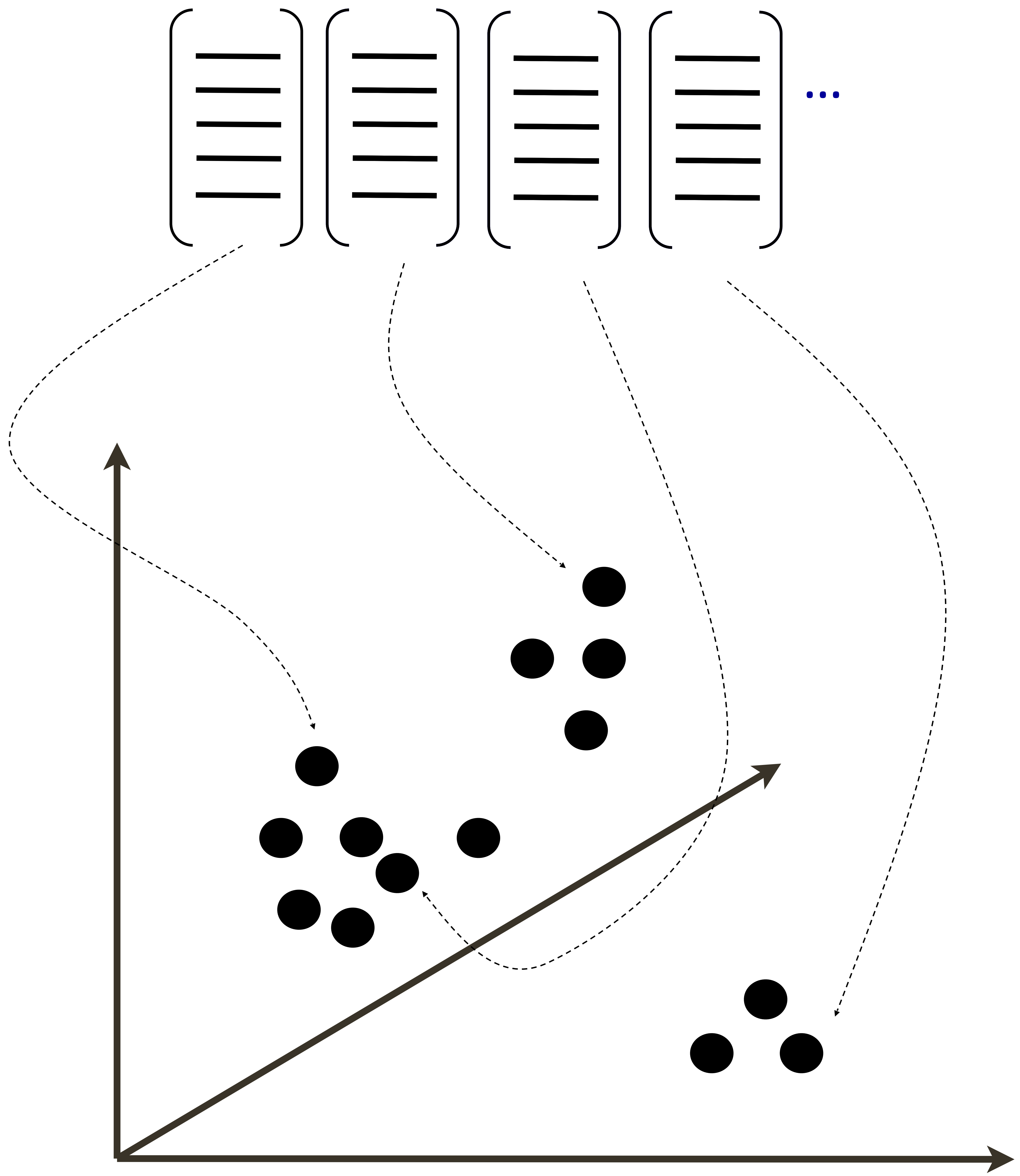
Extracting **SIFT** Patches





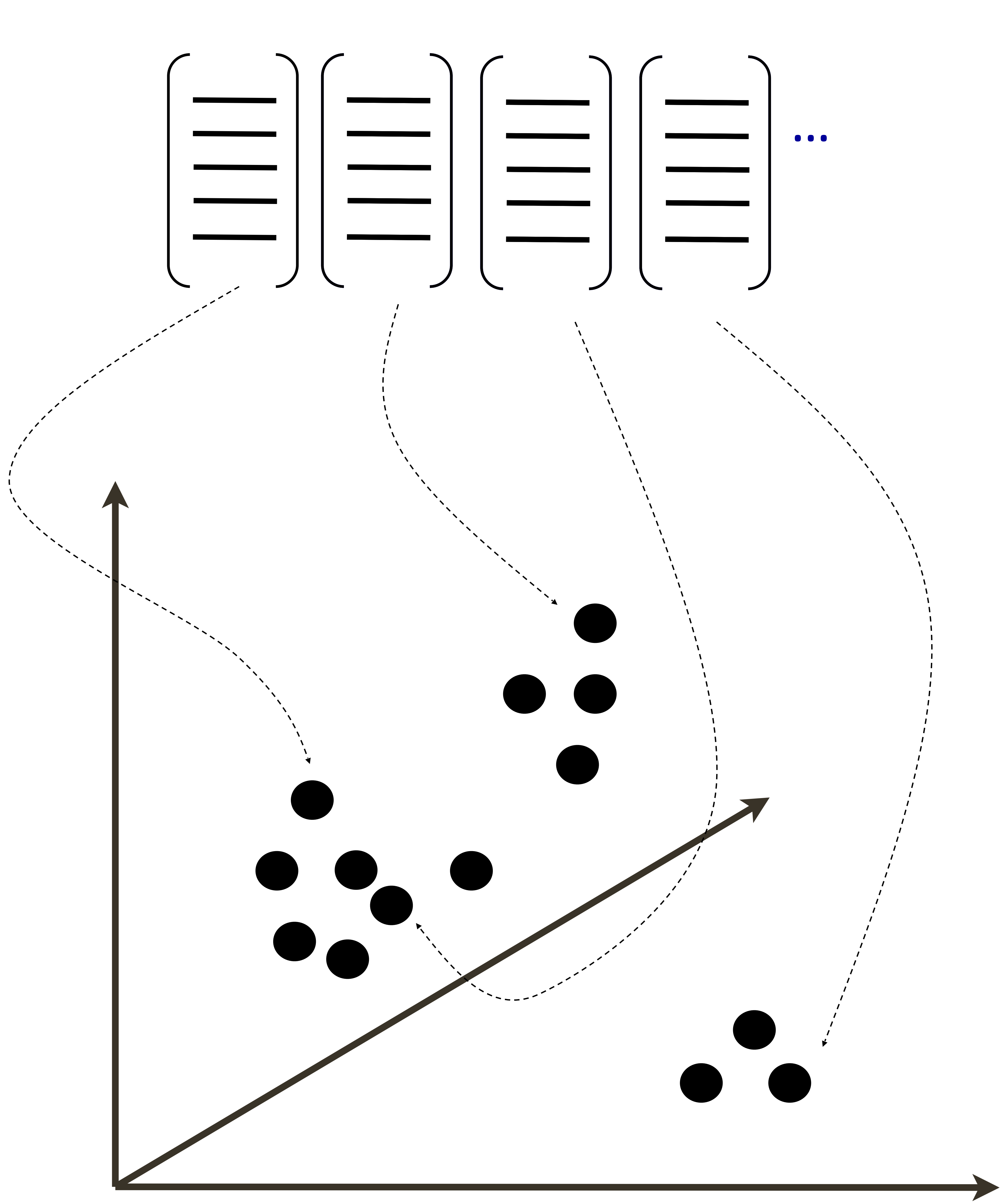
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

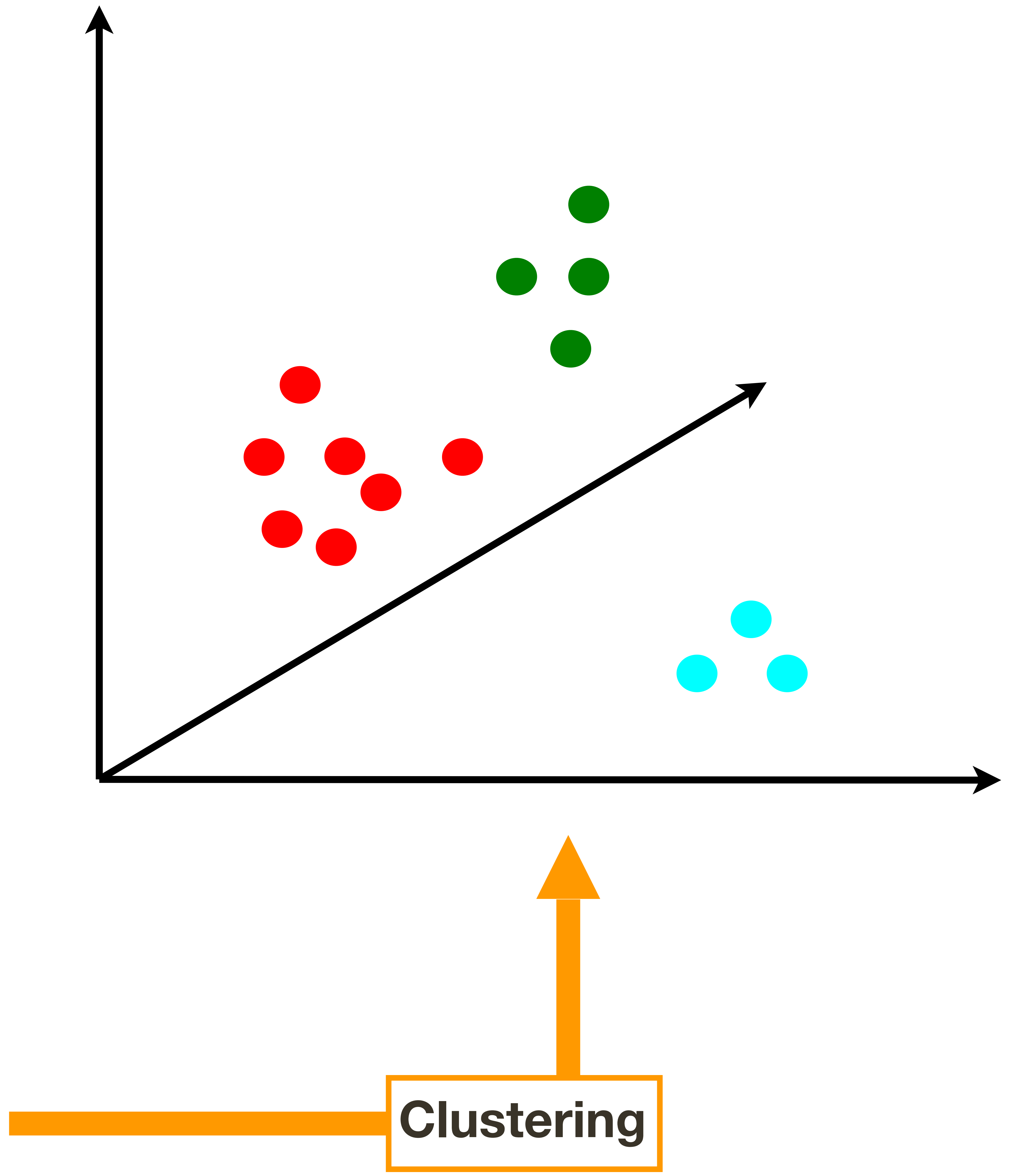
Creating **Dictionary**



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

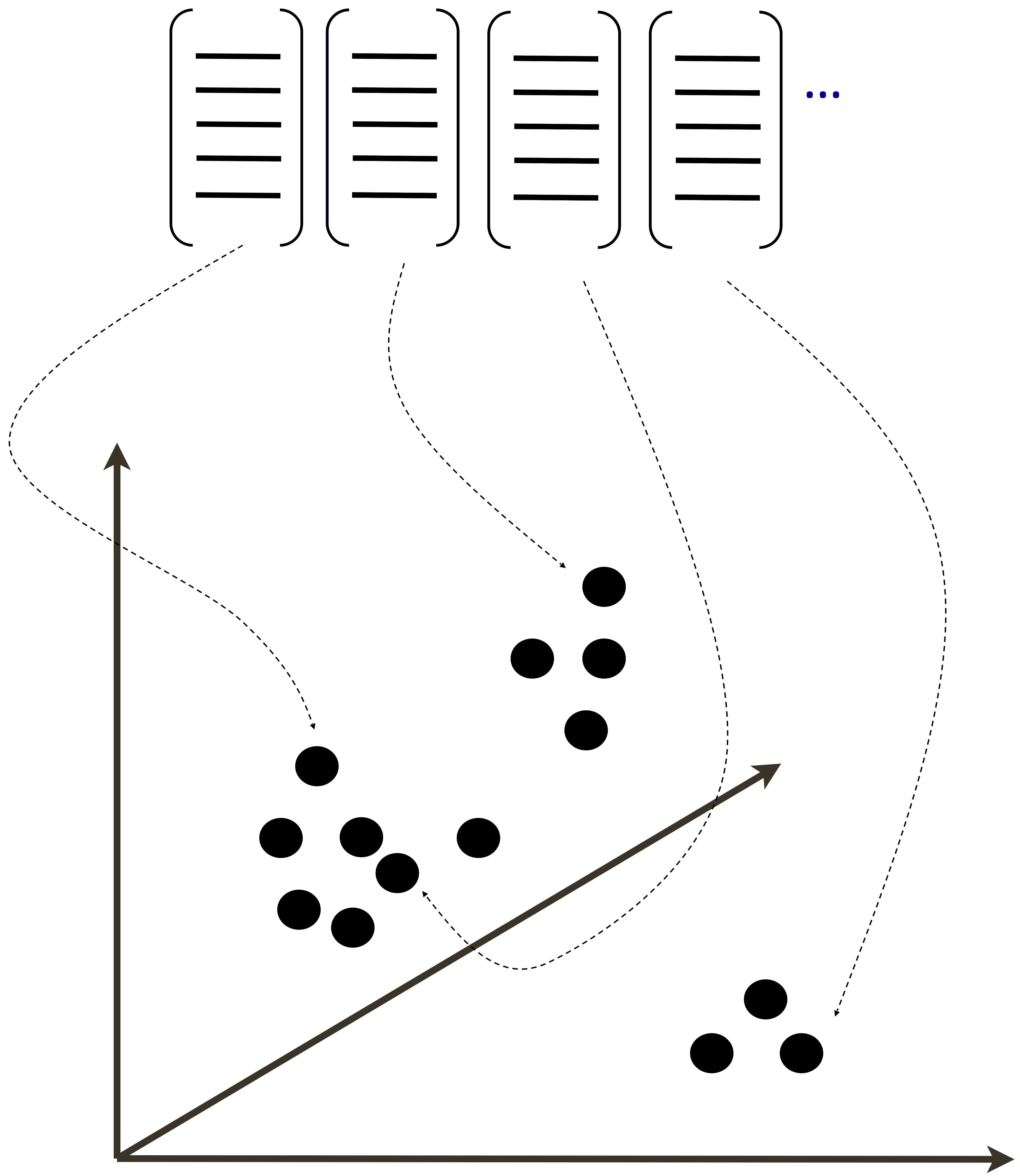
Creating **Dictionary**

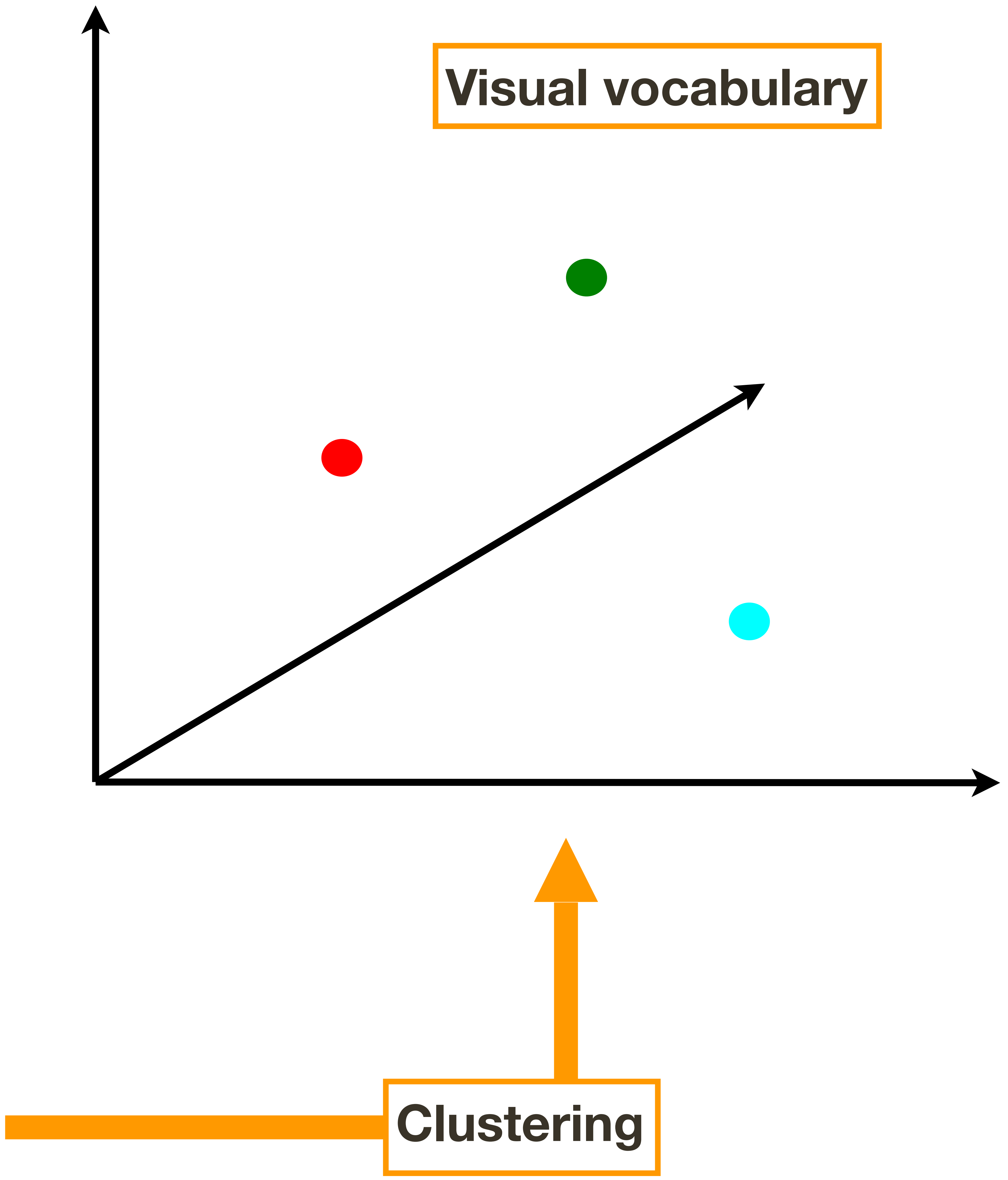




Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

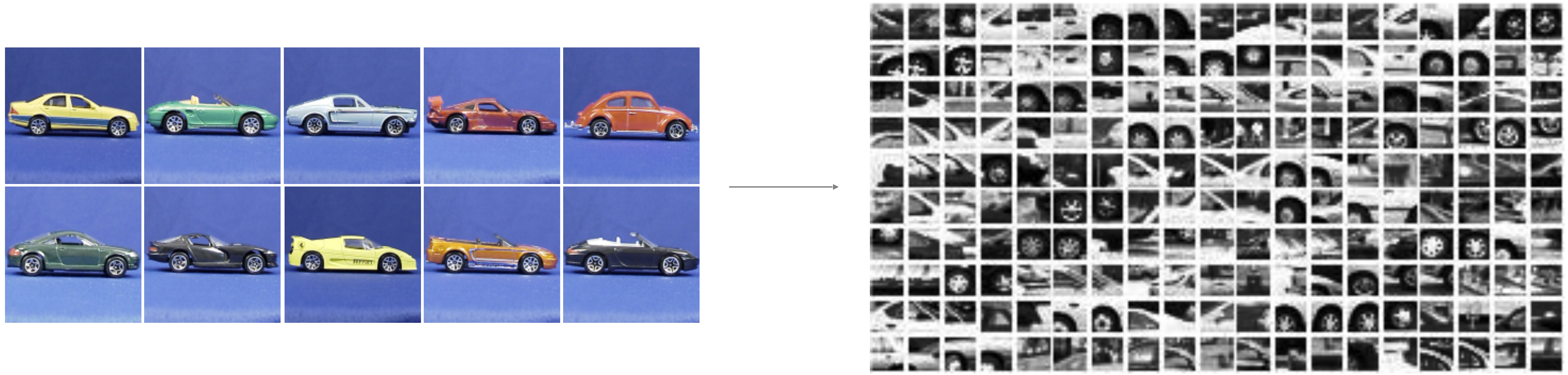
Creating **Dictionary**

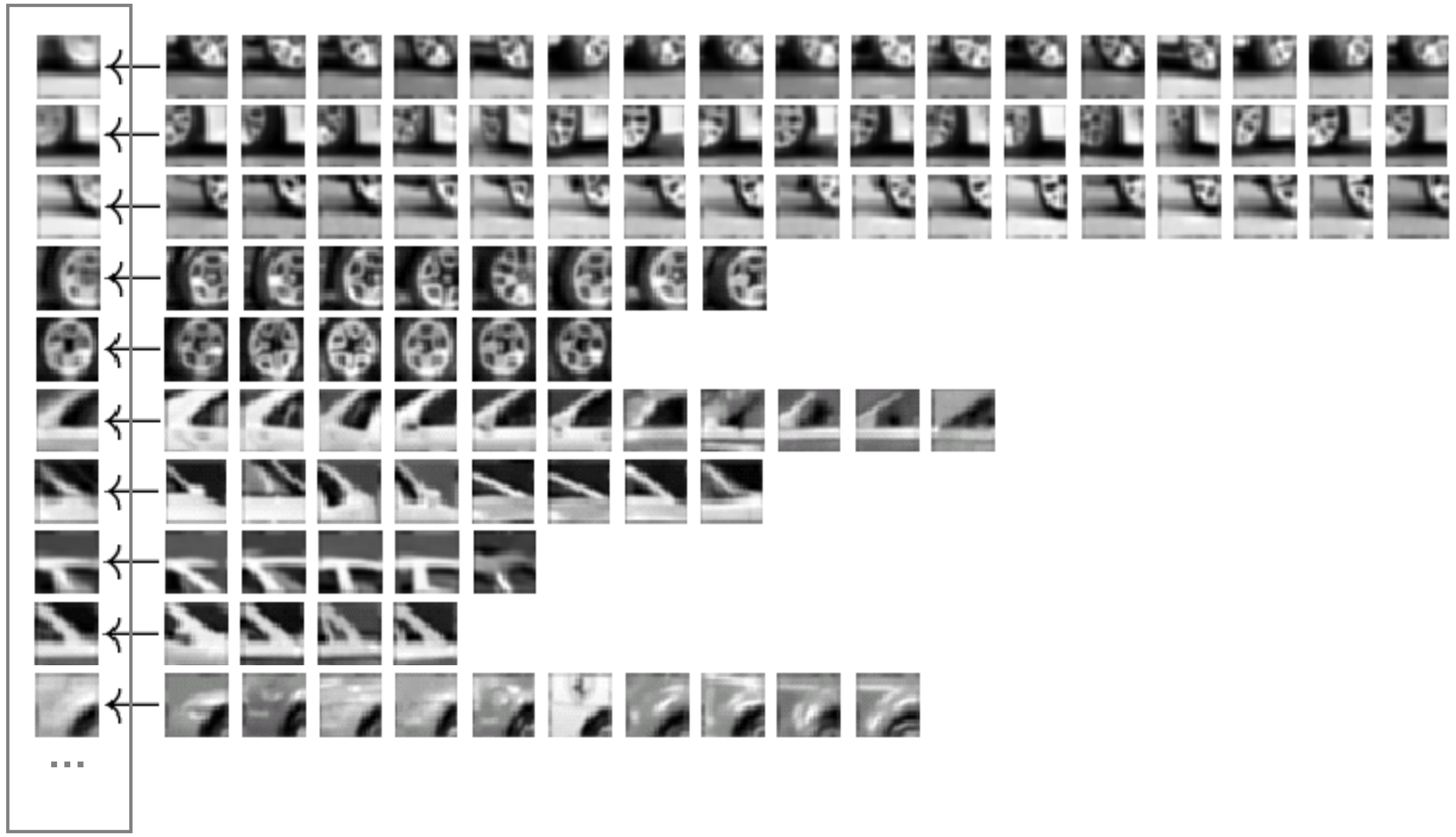




Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

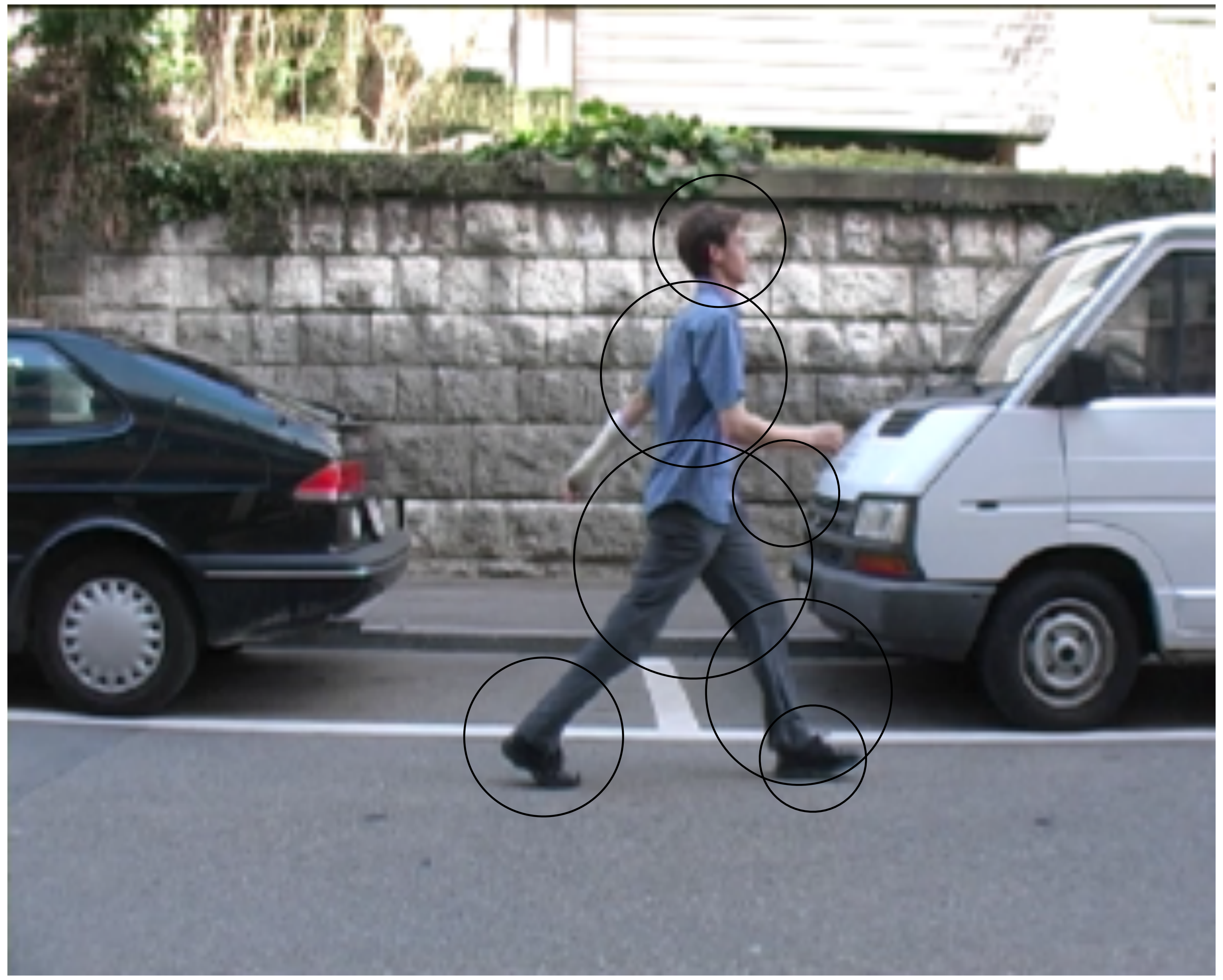
Example **Visual Dictionary**

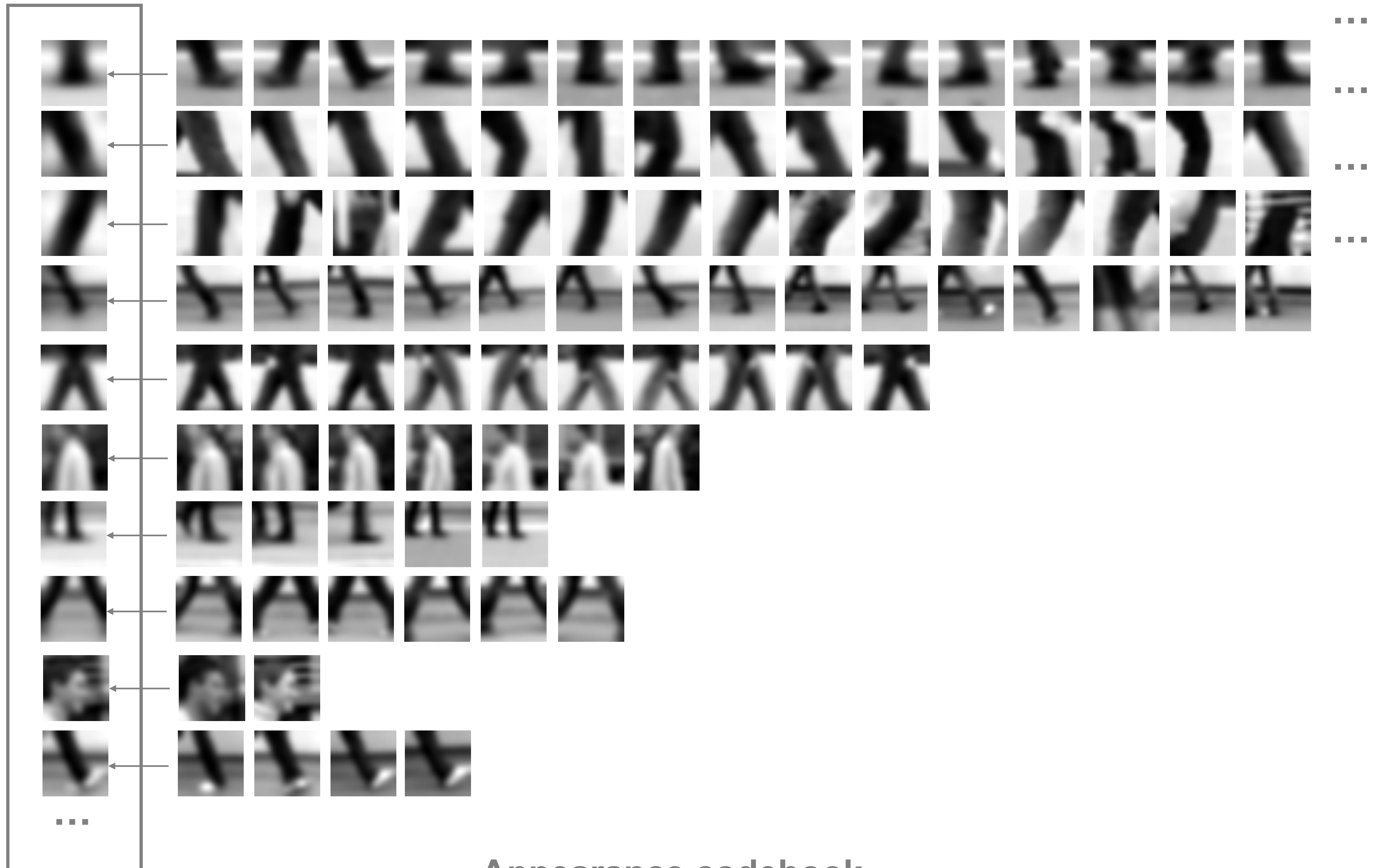




Source: B. Leibe

Example **Visual Dictionary**



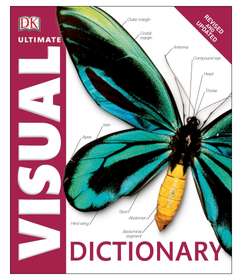


Source: B. Leibe

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

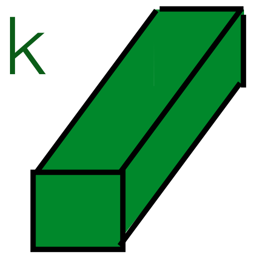
Input: large collection of images
(they don't even need to be training images)
→
Dictionary Learning:
Learn Visual Words using clustering
→
Output: dictionary of visual words

Input: test image, dictionary



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

k

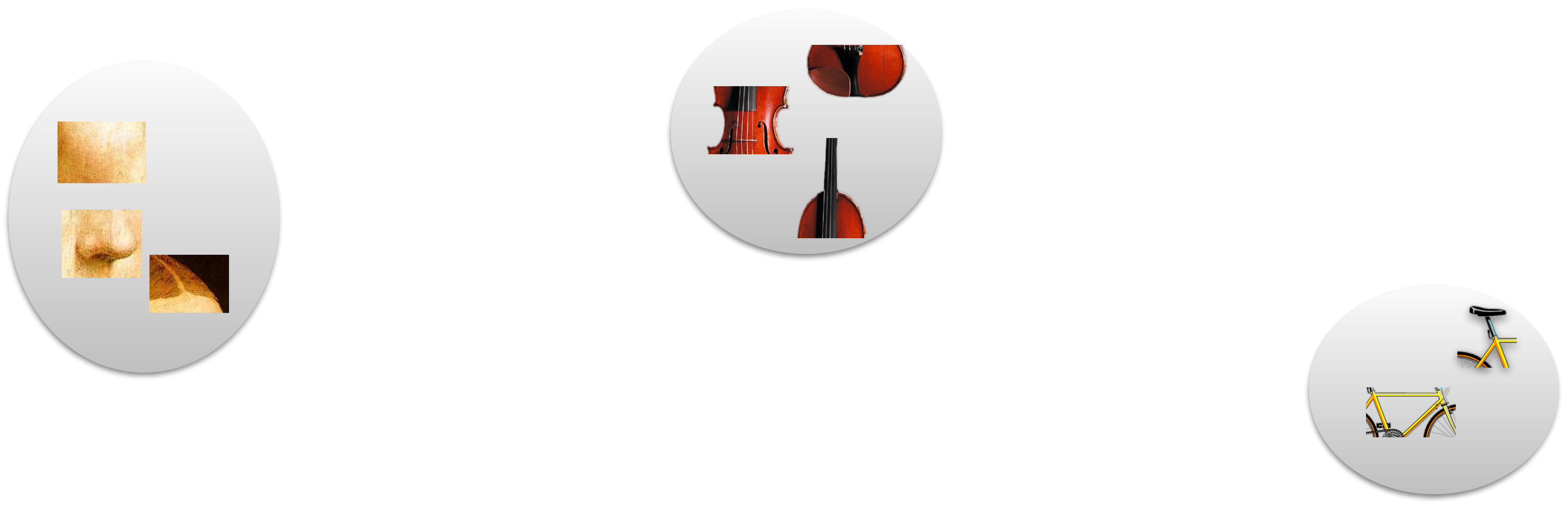


Classify:
Test data using BOWs

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

2. Encode: build Bag-of-Words (BOW) vectors for each image

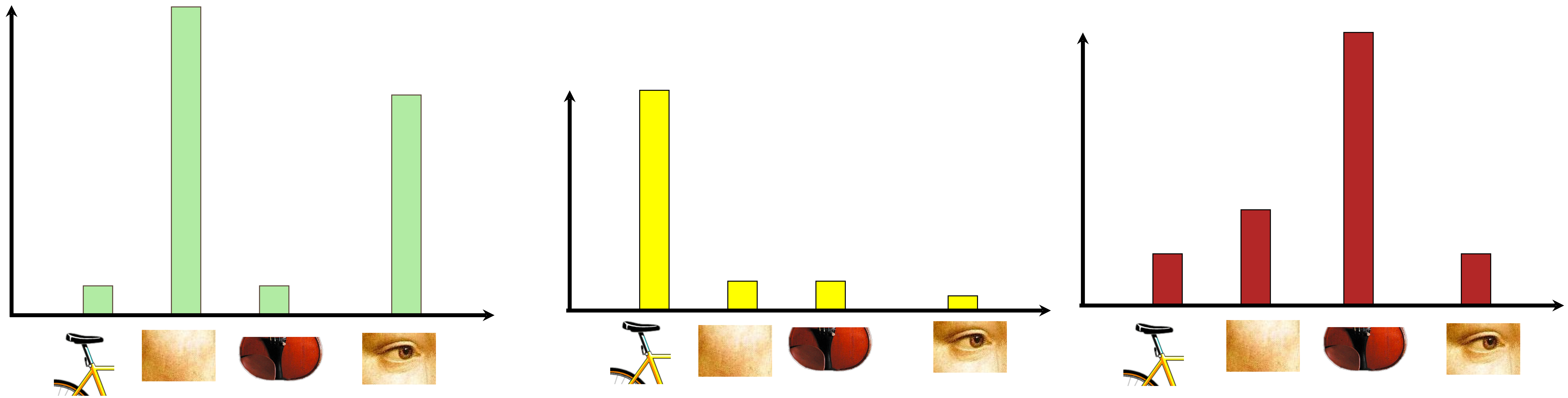
1. **Quantization**: image features gets associated to a visual word (nearest cluster center)



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

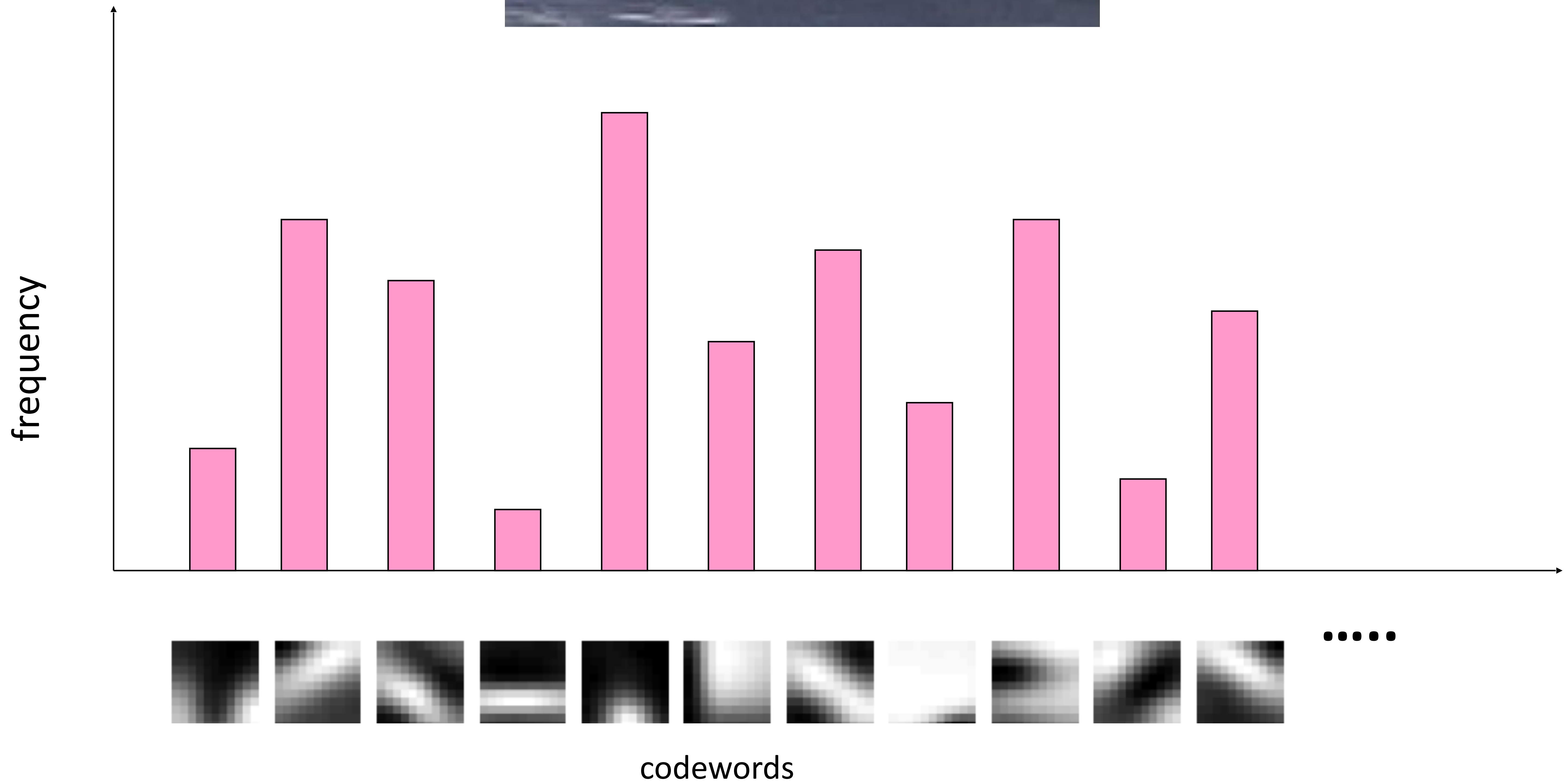
2. Encode: build Bag-of-Words (BOW) vectors for each image

2. **Histogram**: count the number of visual word occurrences



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

2. **Encode:** build Bag-of-Words (BOW) vectors for each image



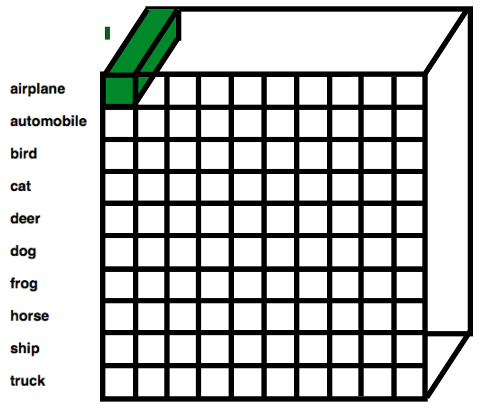
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

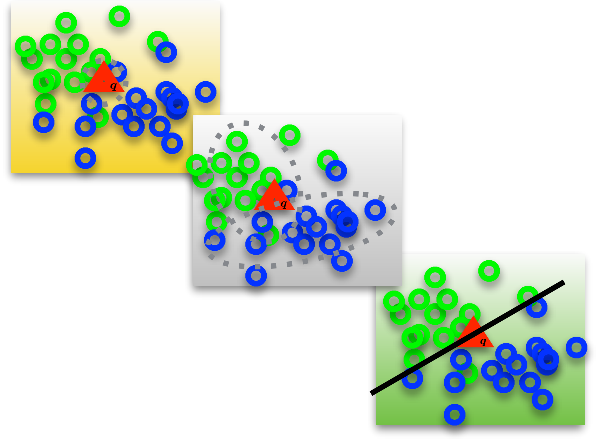
Standard **Bag-of-Words** Pipeline (for image classification) — **Training**

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

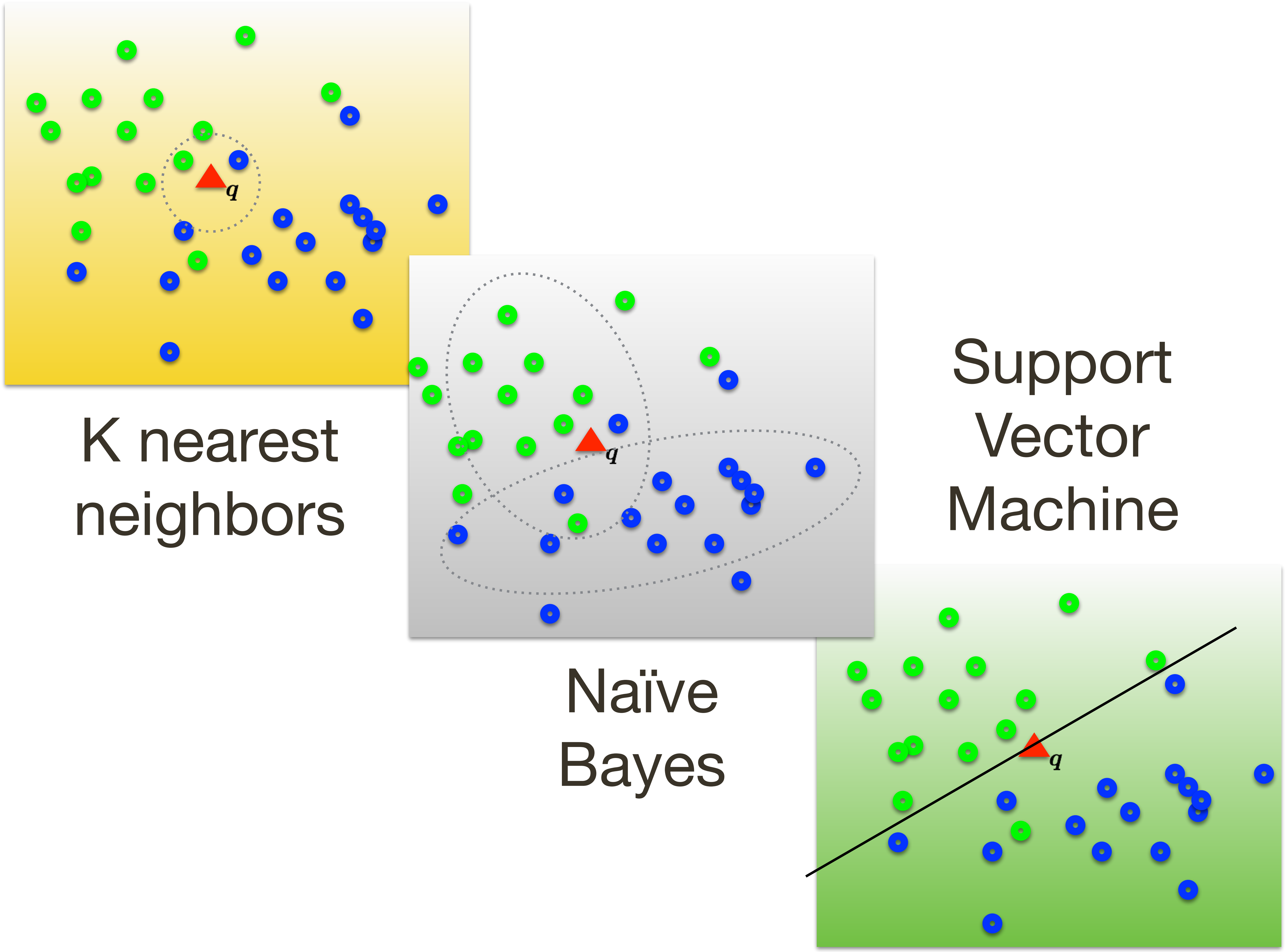
Input: histogram representation for
each training image + labels
→ **Classify**:
Train data using BOWs
→ **Output**: parameters if the classifier





Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

3. Classify: Train and text classifier 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)

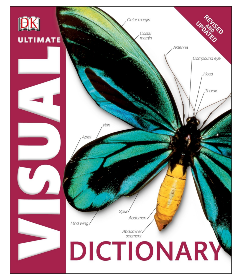
→

Dictionary Learning:
Learn Visual Words using clustering

→

Output: dictionary of visual words

Input: test image, dictionary



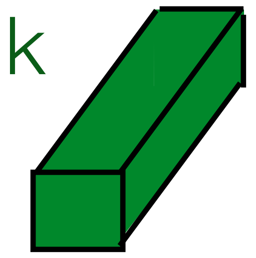
→

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

→

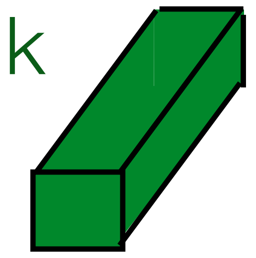
Output: histogram representation for test image

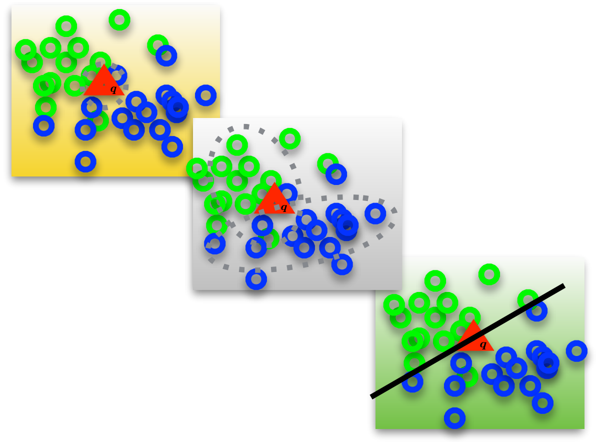
k



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)

Inference **Bag-of-Words** Representation

Algorithm:

Initialize an empty K-bin histogram, where K is the number of codewords
Extract local descriptors (e.g. SIFT) from the image
For each local descriptor **x**

Map (Quantize) **x** to its closest codeword → **c**(**x**)

Increment the histogram bin for **c**(**x**)

Return histogram

We can then classify the histogram using a trained classifier, e.g. a support vector machine or k-Nearest Neighbor classifier

Spatial Pyramid

The bag of words representation does not preserve any spatial information

The **spatial pyramid** is one way to incorporate spatial information into the image descriptor.

A spatial pyramid partitions the image and counts codewords within each grid box; this is performed at multiple levels

Spatial Pyramid

Compute Bag-of-Words histograms for each quadrant and then concatenate them

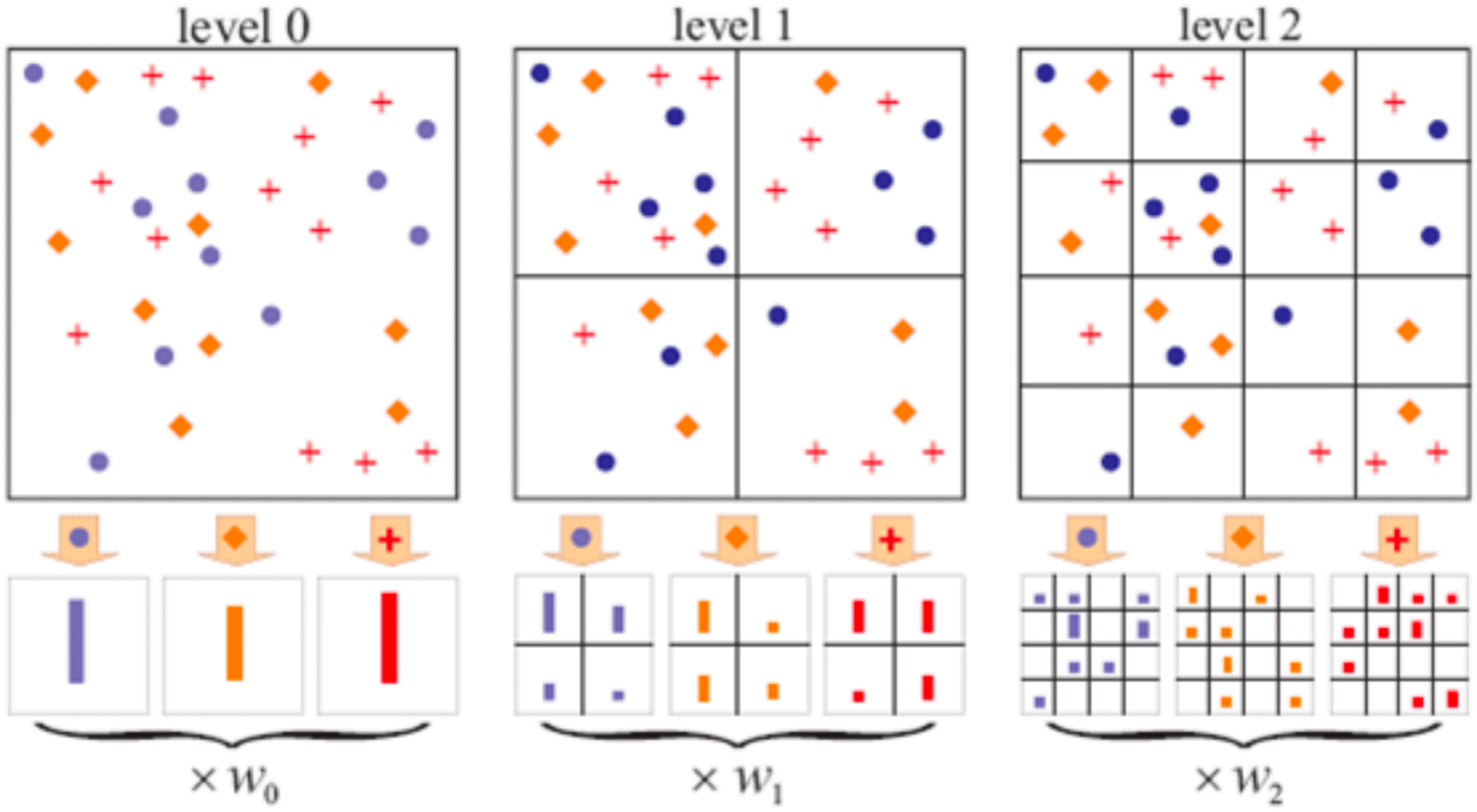


Fig. 16.8 in Forsyth & Ponce (2nd ed.).
Original credit: Lazebnik et al., 2006

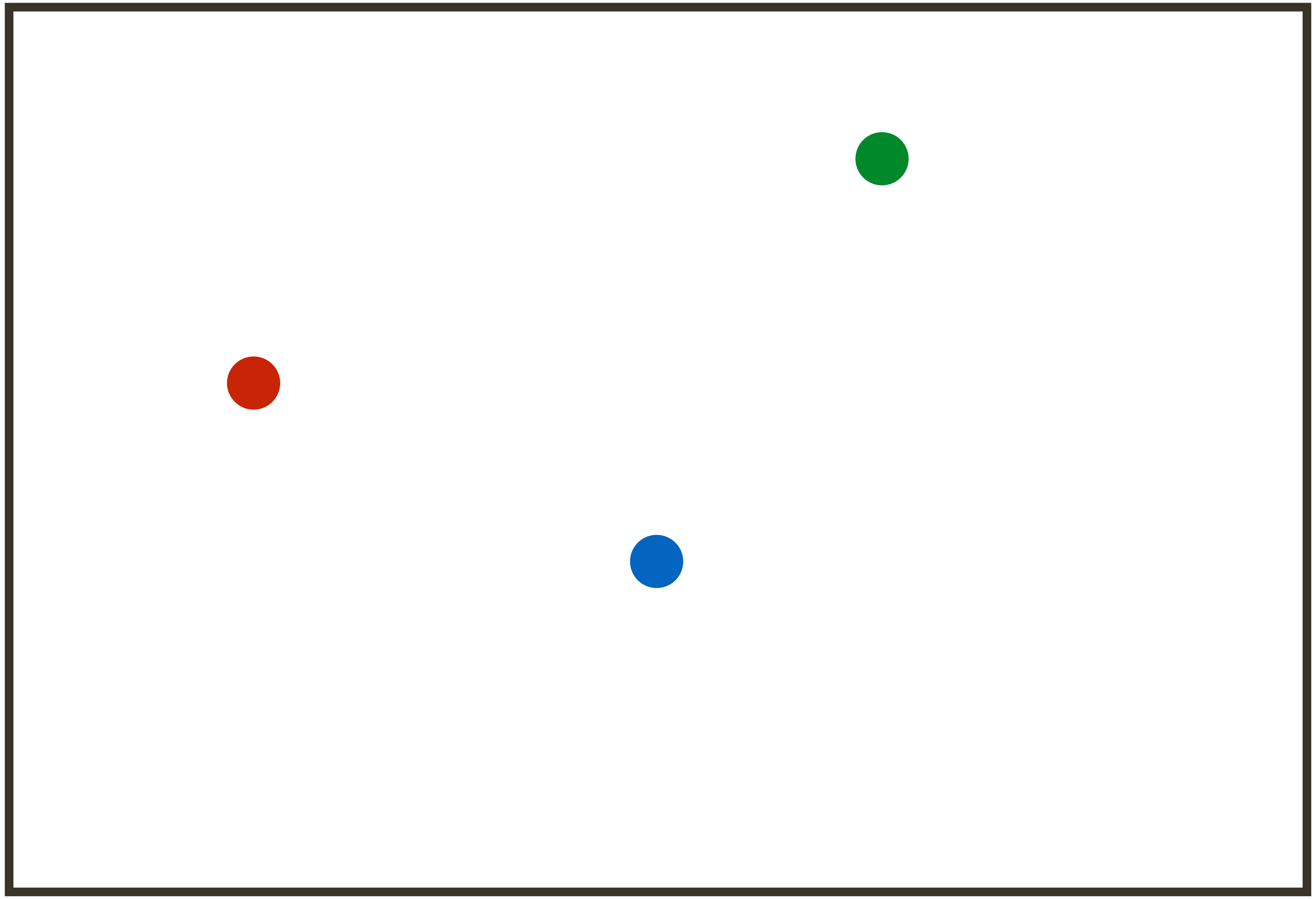
VLAD (Vector of Locally Aggregated Descriptors)

There are more advanced ways to ‘count’ visual words than incrementing its histogram bin

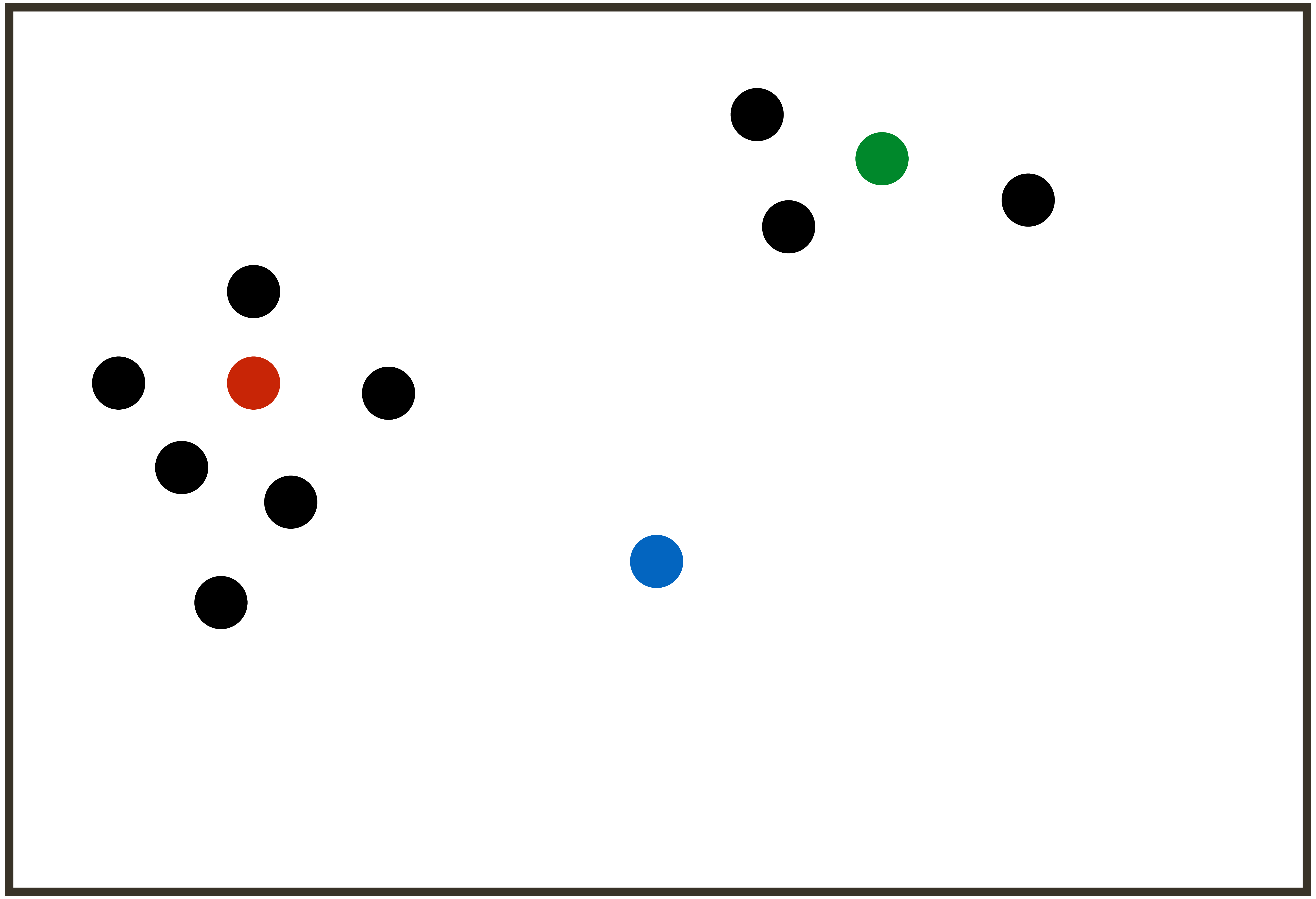
For example, it might be useful to describe how local descriptors are quantized to their visual words

In the VLAD representation, instead of incrementing the histogram bin by one, we increment it by the **residual** vector $\boldsymbol{x - c(x)}$

Example: VLAD

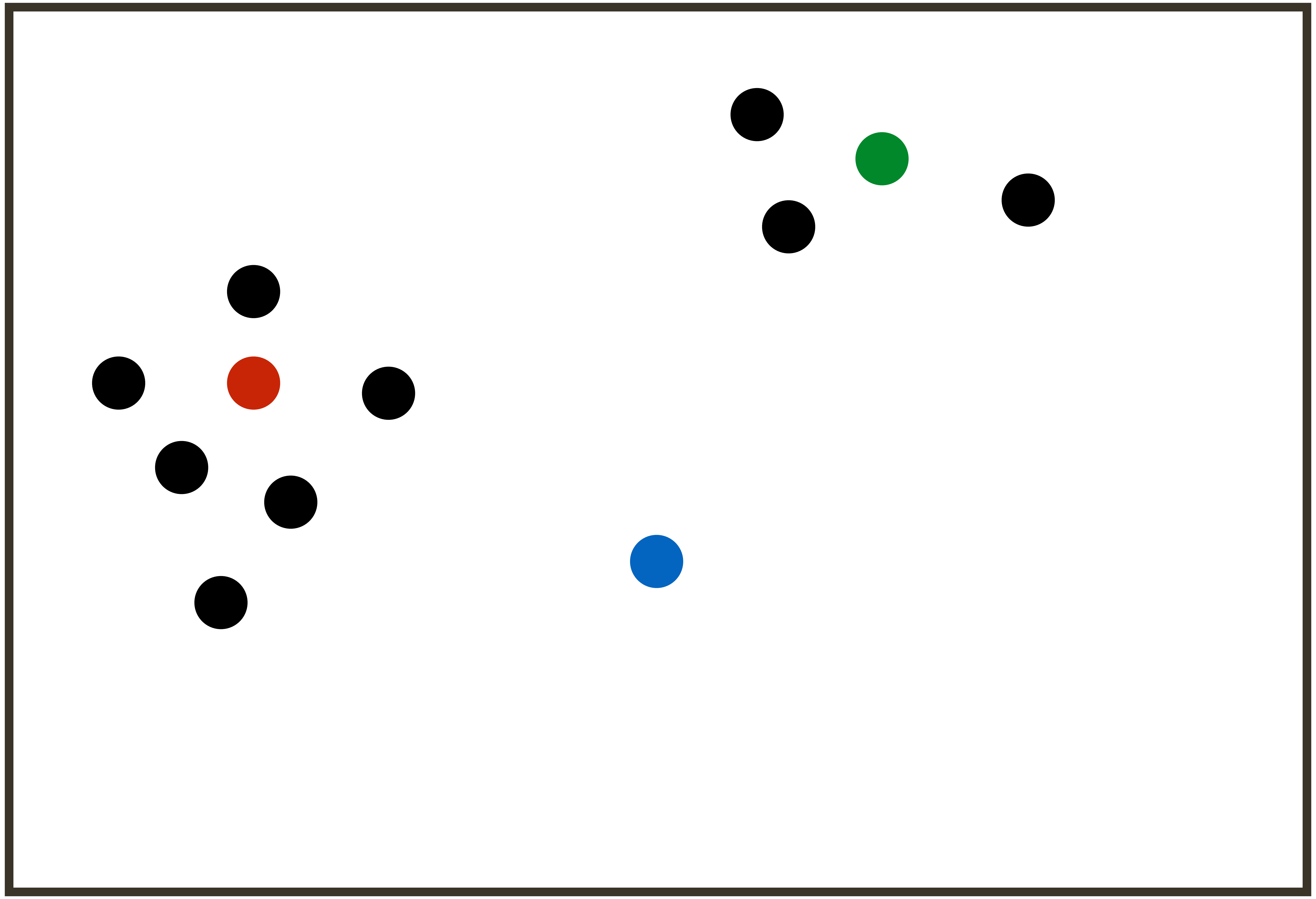


Example: VLAD

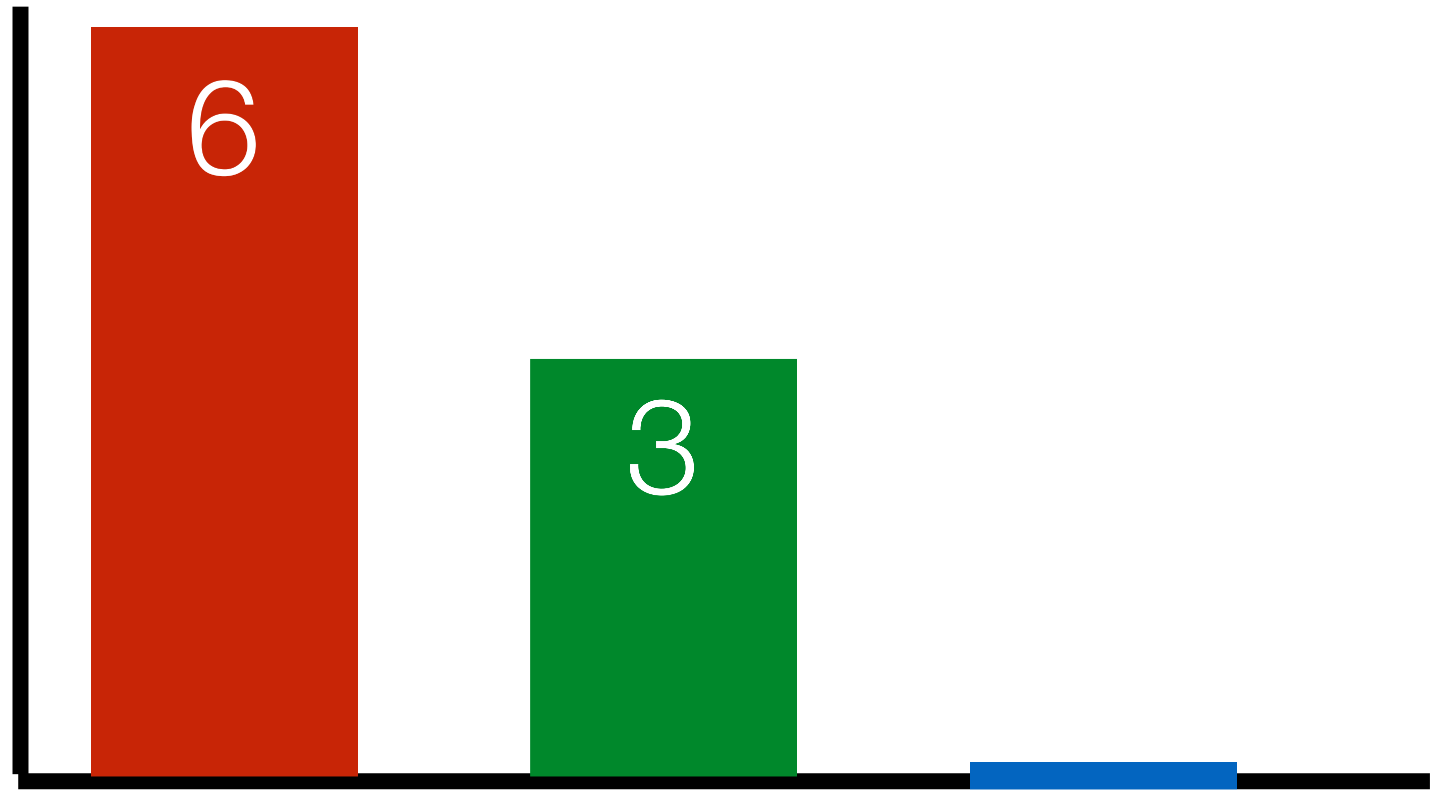


Bag of Word

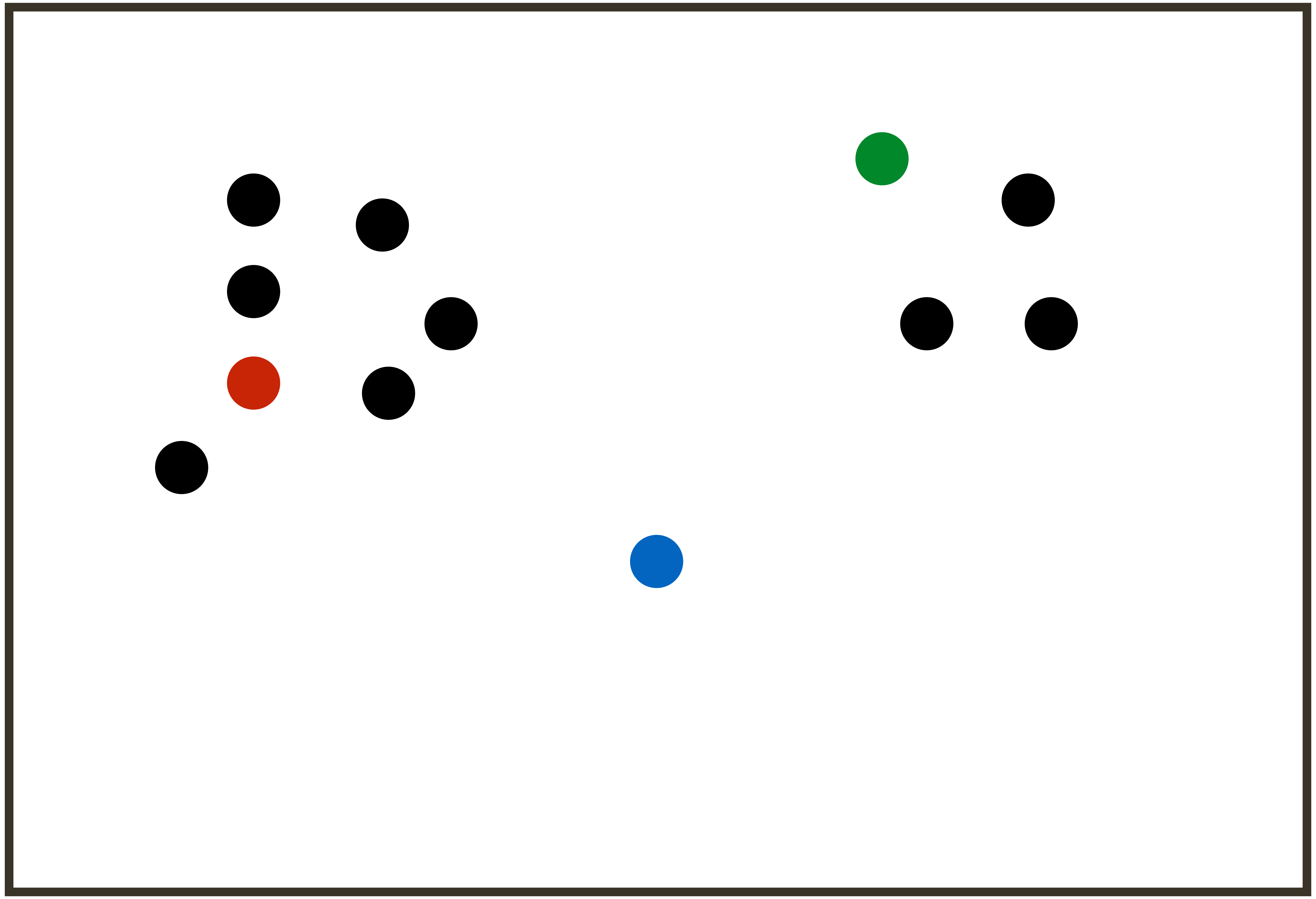
Example: VLAD



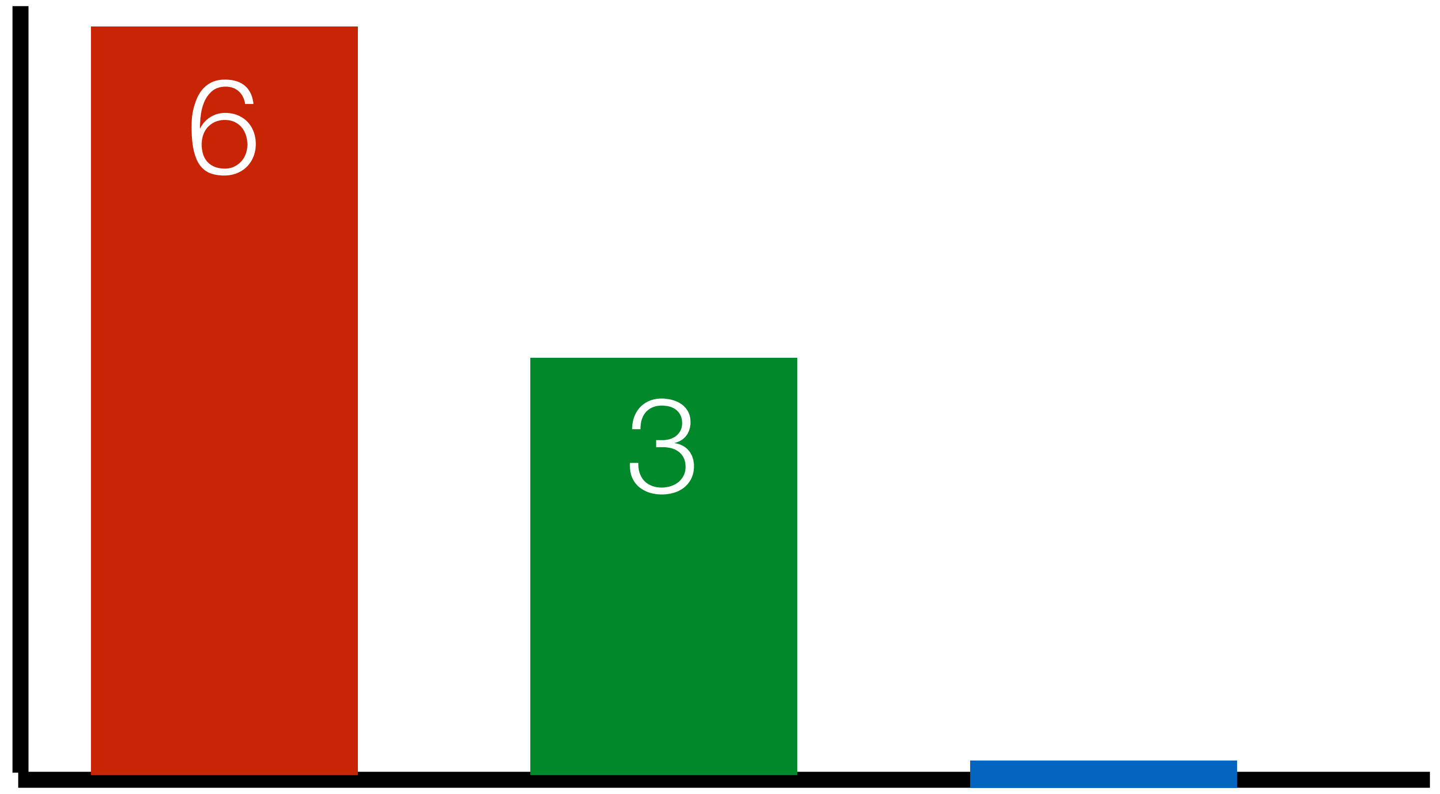
Bag of Word



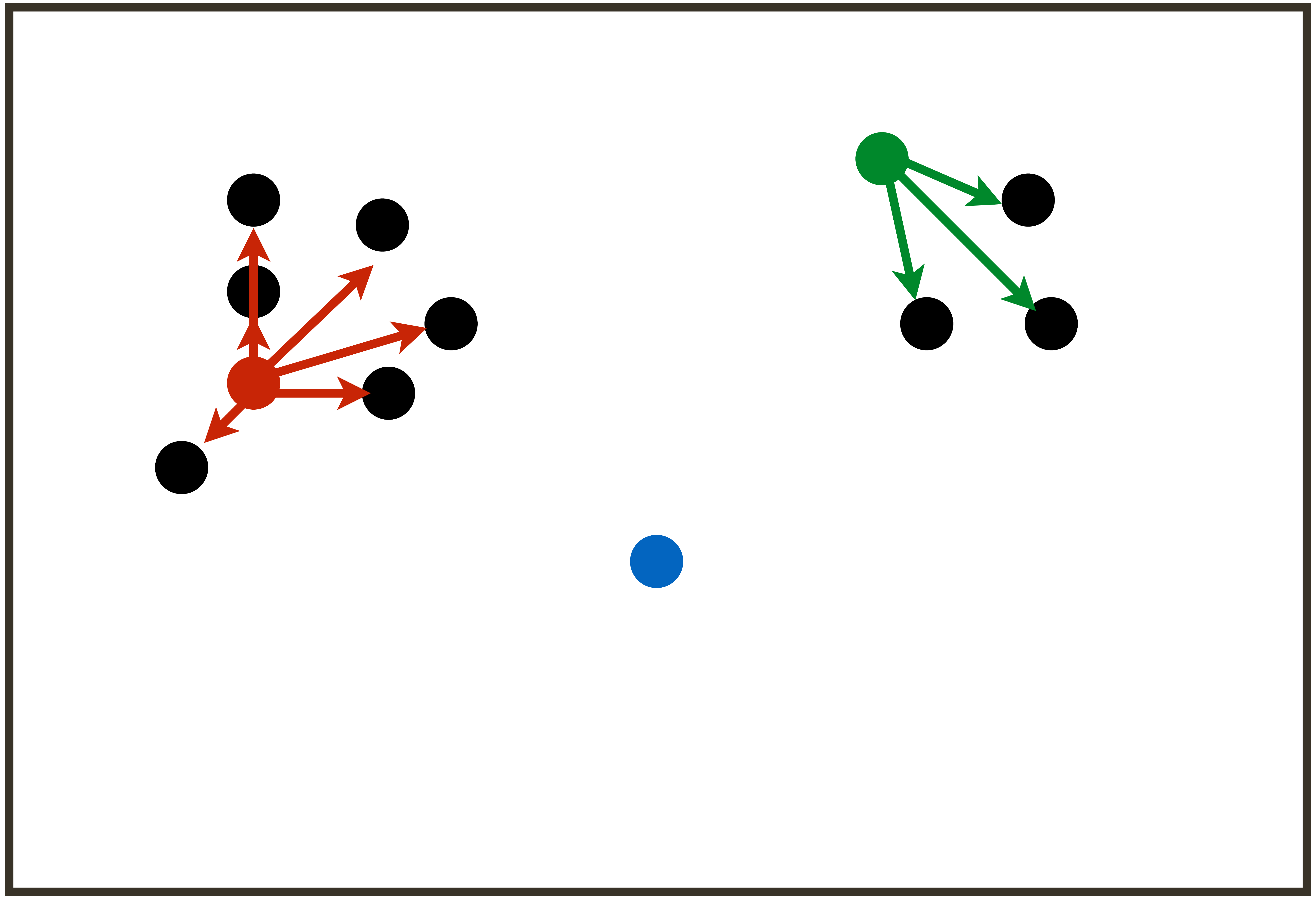
Example: VLAD



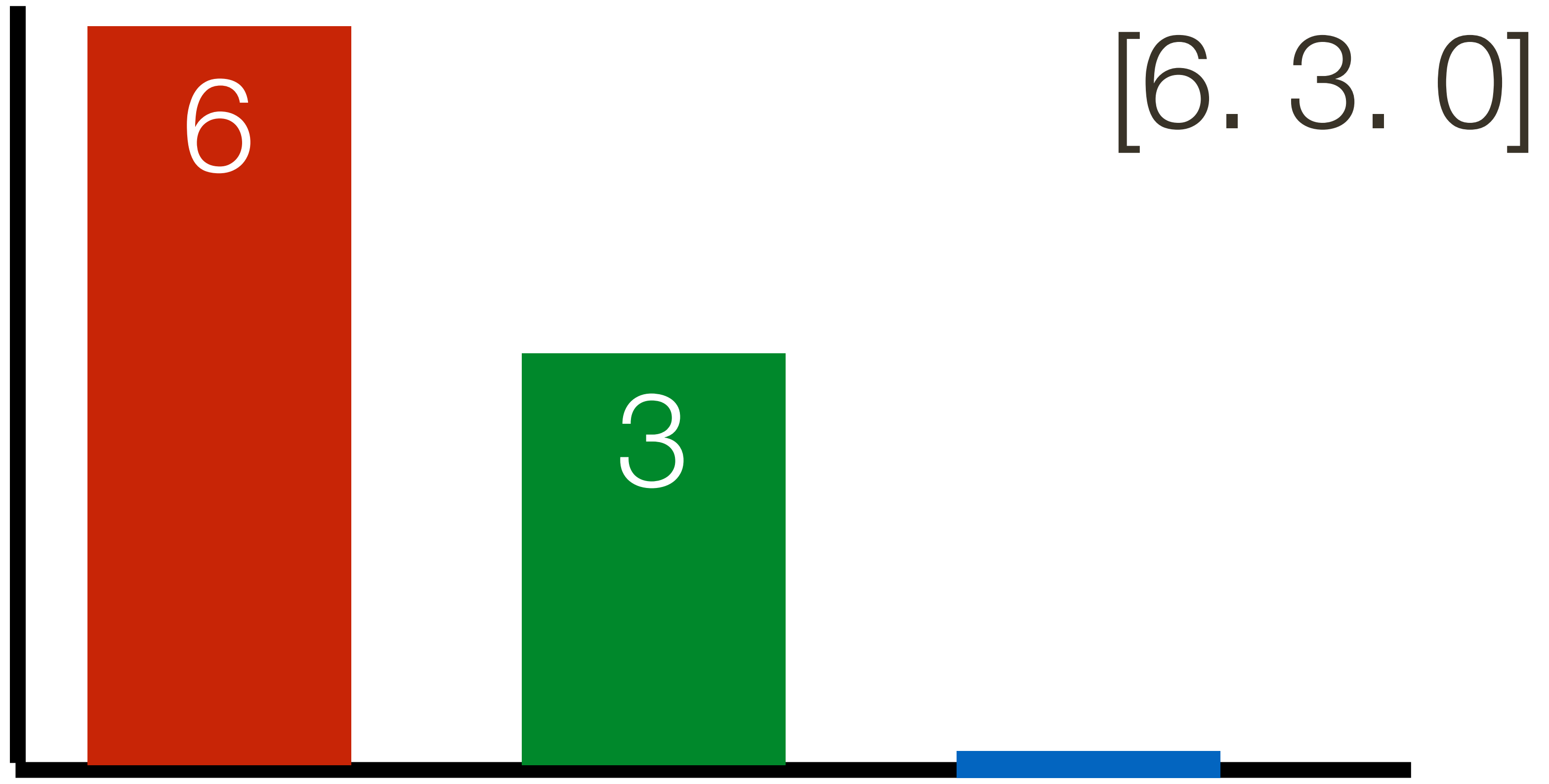
Bag of Word



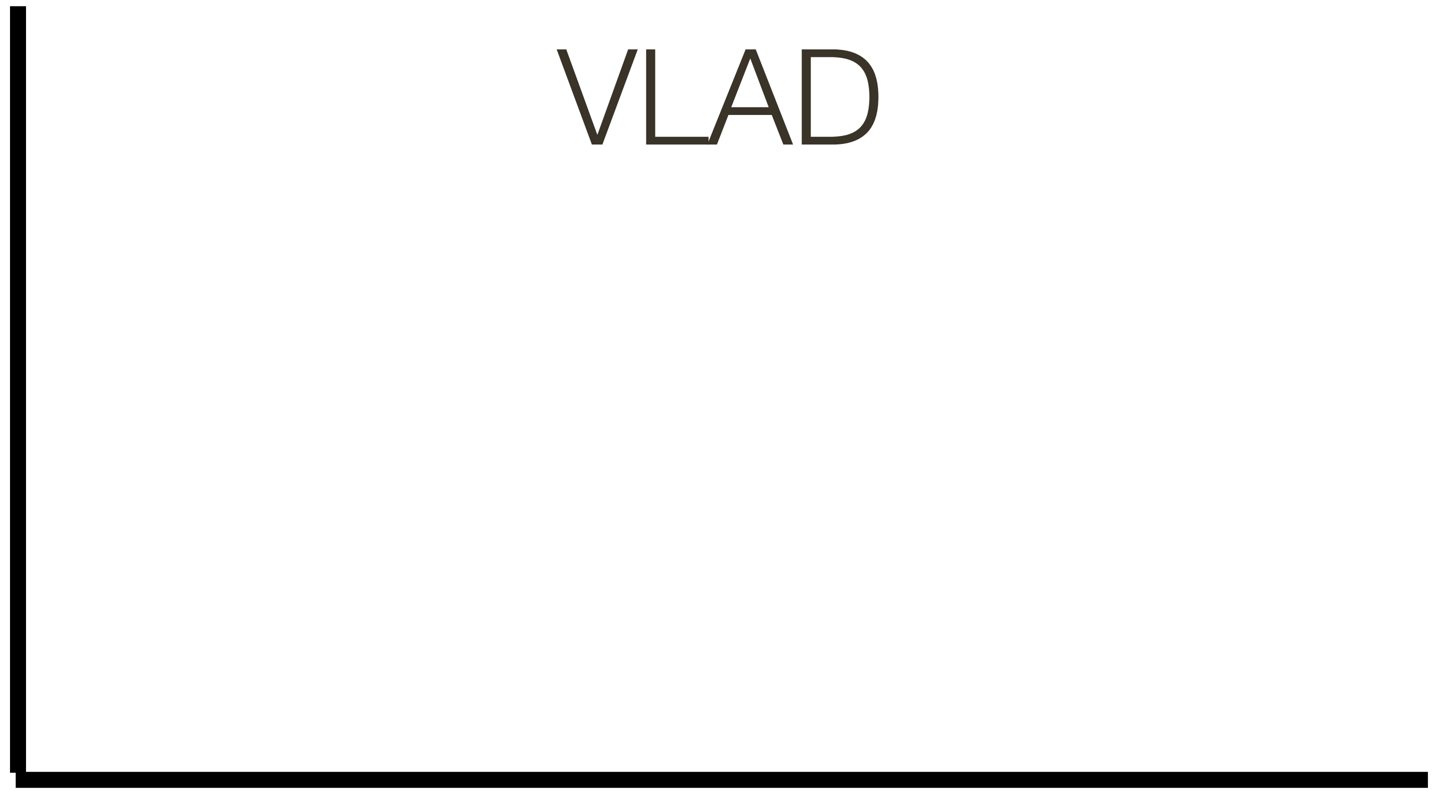
Example: VLAD



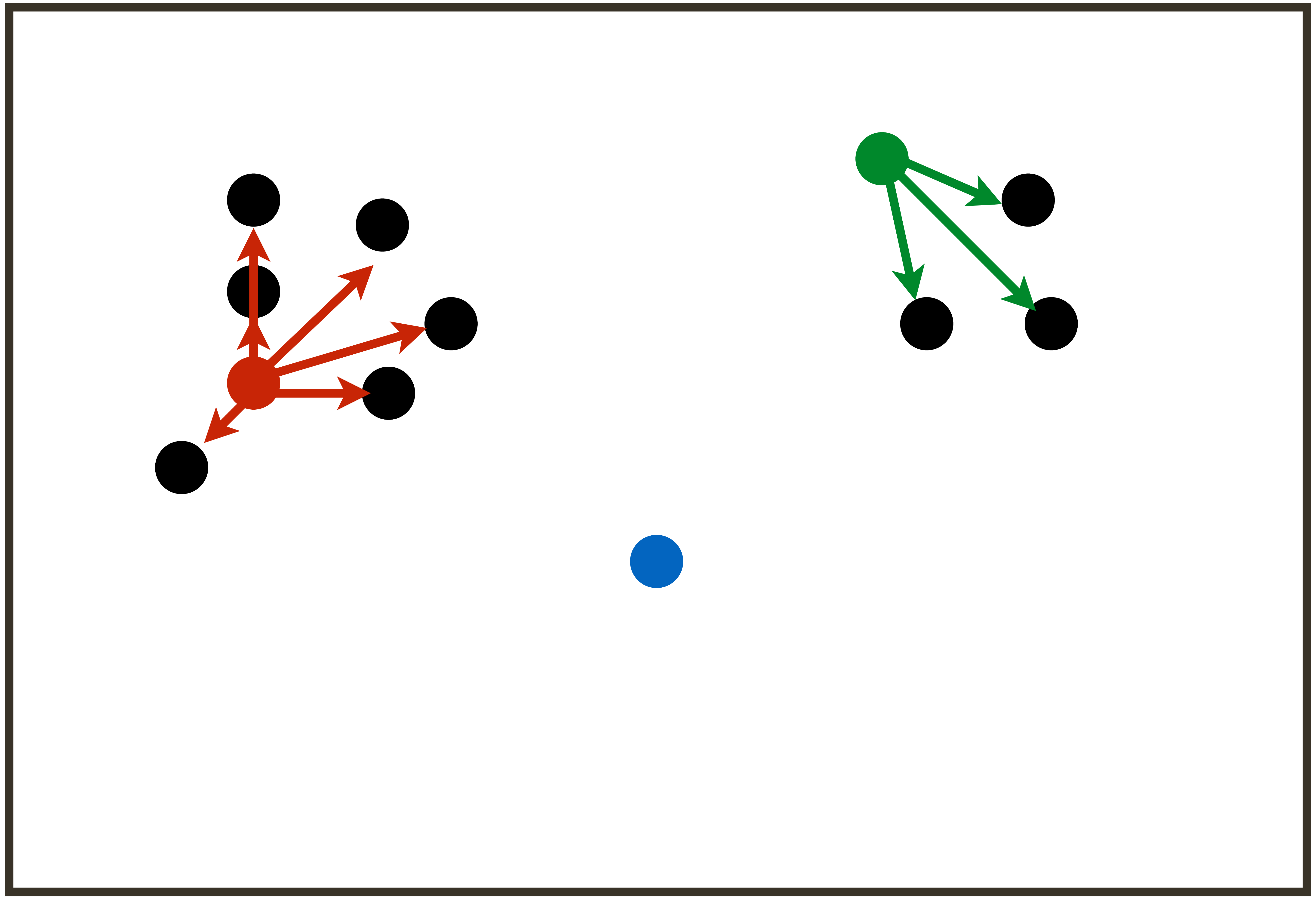
Bag of Word



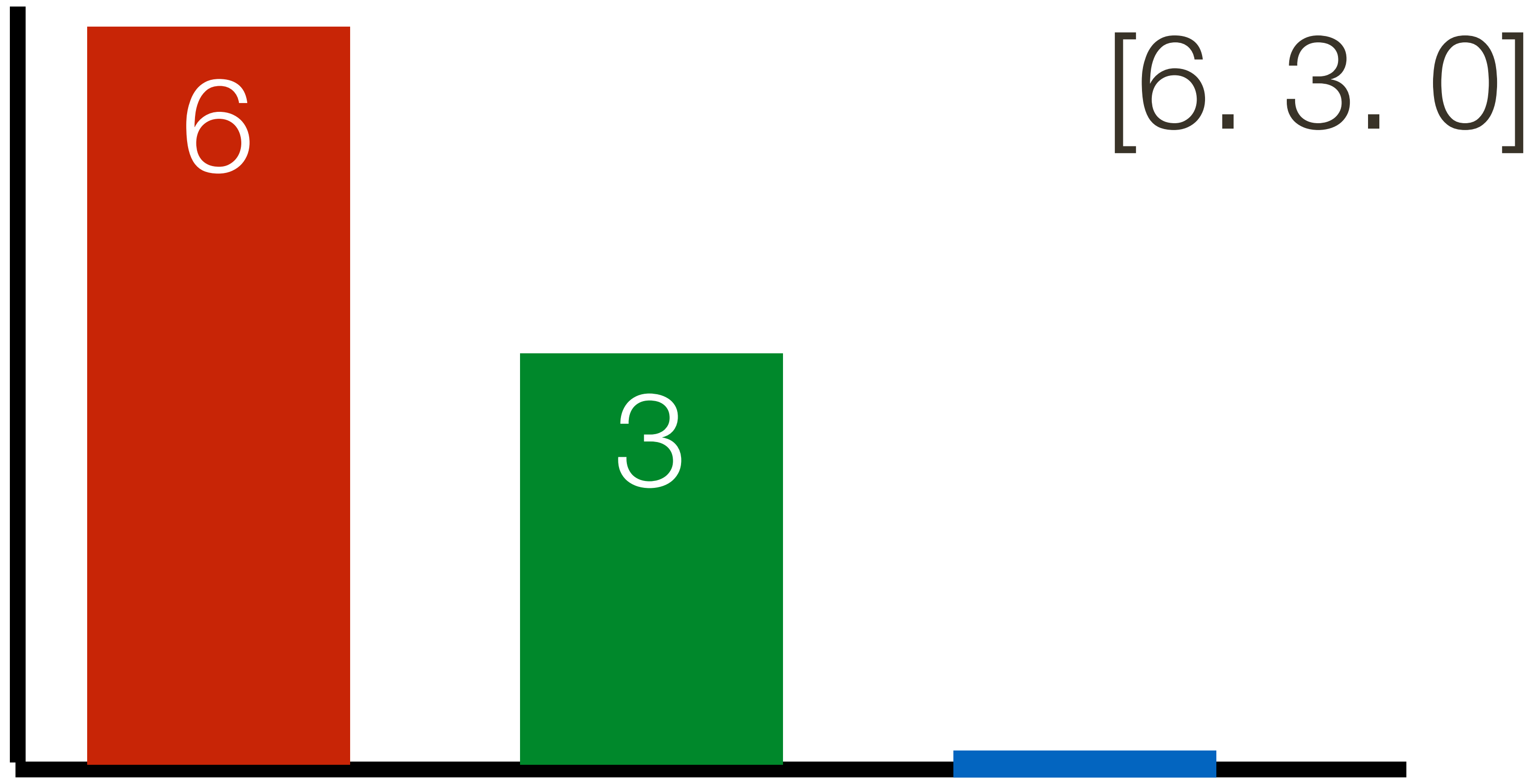
VLAD

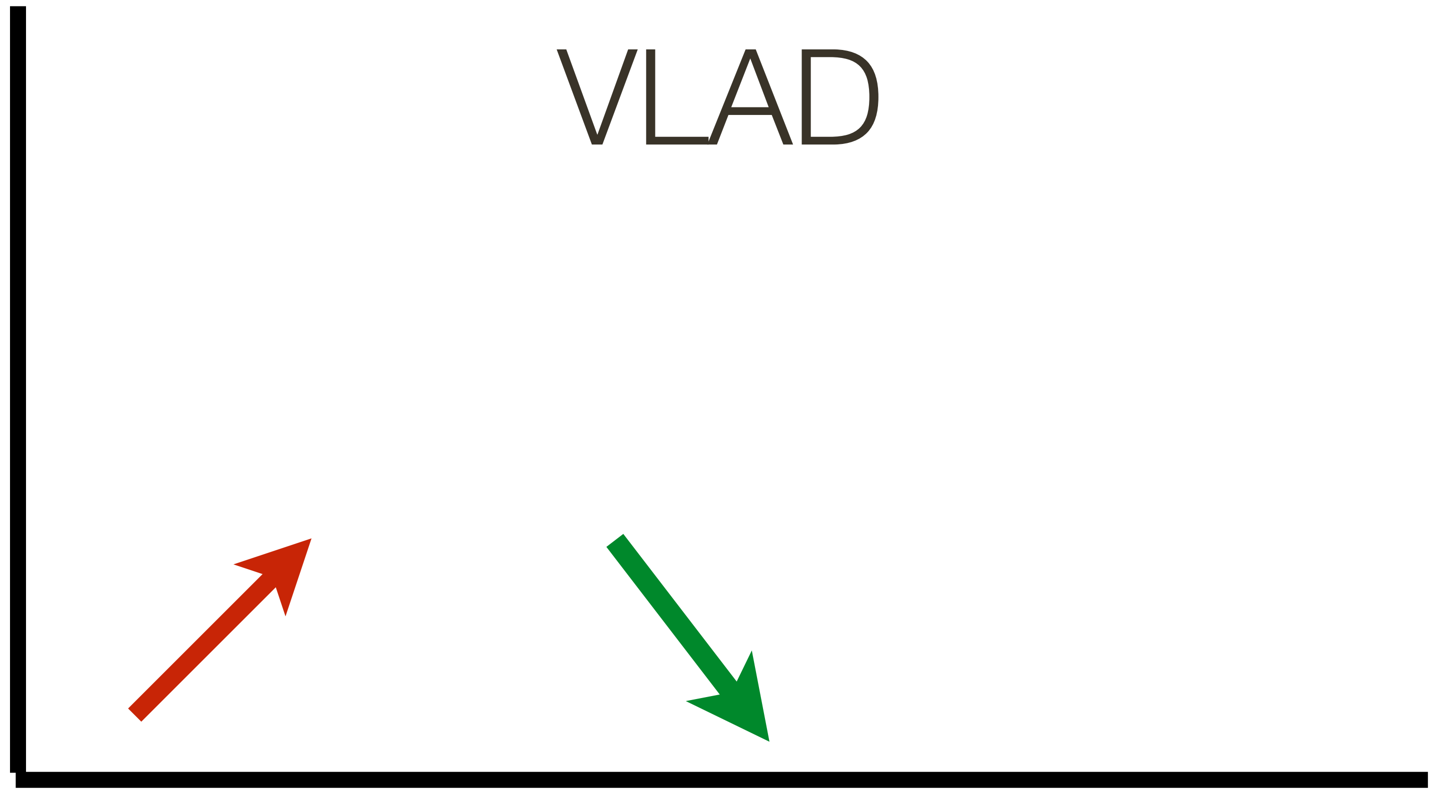


Example: VLAD



Bag of Word





VLAD (Vector of Locally Aggregated Descriptors)

The dimensionality of a **VLAD** descriptor is Kd

— K : number of codewords

— d : dimensionality of the local descriptor

VLAD characterizes the distribution of local descriptors with respect to the codewords

Recognition Overview: Early —> 2024

Recognition Overview: Early

→

Rule Based **Classifier**:
Distance + Threshold

There is nothing really to "**learn**" (no need for training data), just measure similarity using favorite distance and choose threshold based on validation set

Recognition Overview: Early

Rule Based **Classifier**:
Distance + Threshold

There is nothing really to "**learn**" (no need for training data), just measure similarity using favorite distance and choose threshold based on validation set

Local Features:
Edges

Rule Based **Classifier**:
Distance + Threshold

More robust, to lighting, but basically same

Recognition Overview: Early

Rule Based **Classifier**:
Distance + Threshold

There is nothing really to "**learn**" (no need for training data), just measure similarity using favorite distance and choose threshold based on validation set

Local Features:
Edges

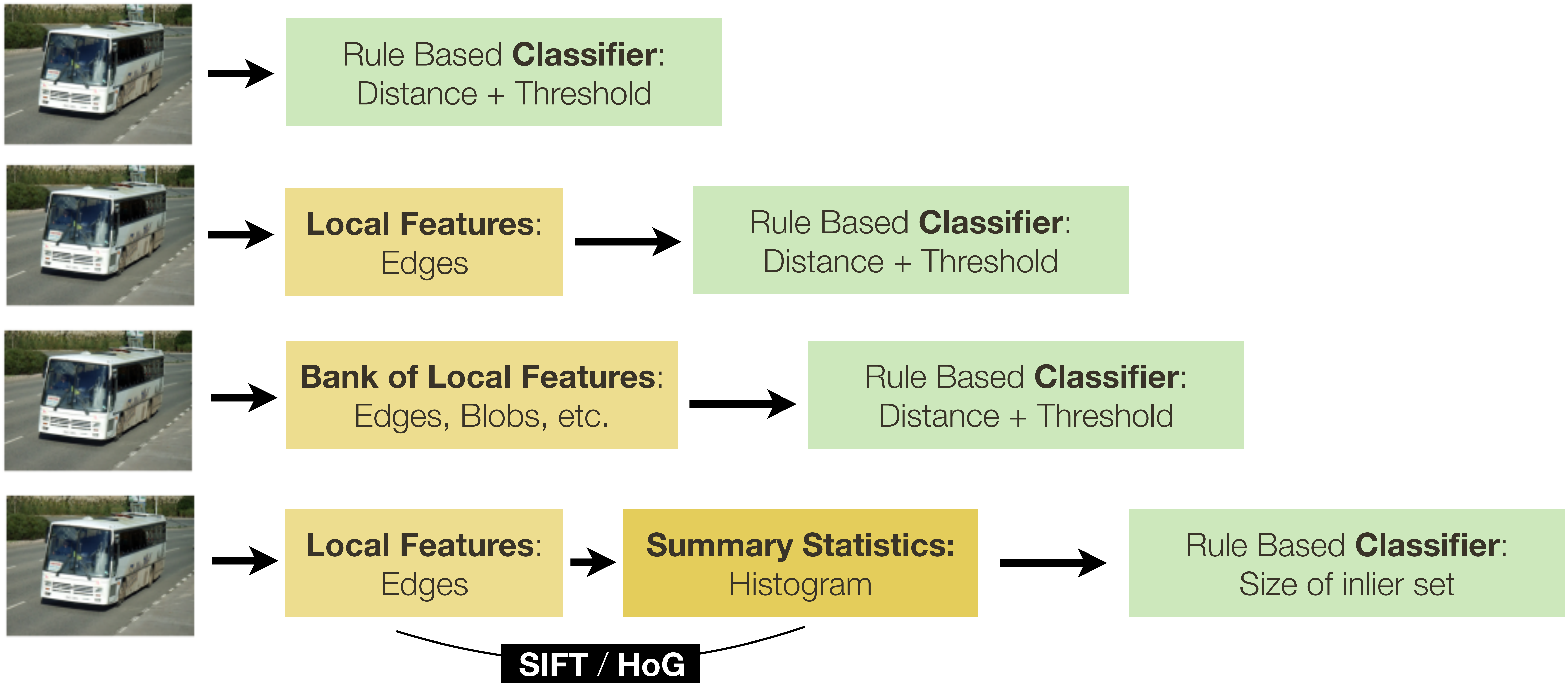
Rule Based **Classifier**:
Distance + Threshold

Bank of Local Features:
Edges, Blobs, etc.

Rule Based **Classifier**:
Distance + Threshold

More expressive, but basically same

Recognition Overview: Early

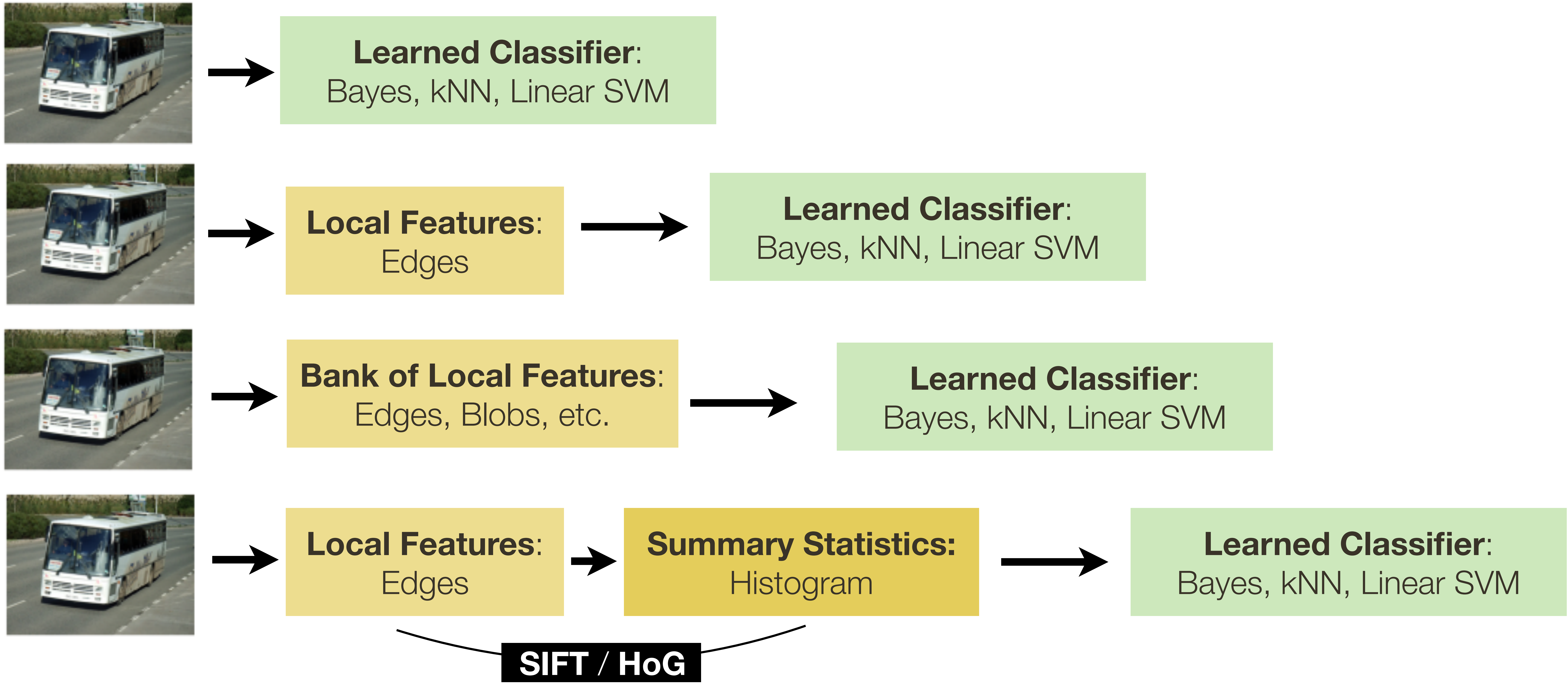


— No real learning, mostly parameter/design tuning using validation set

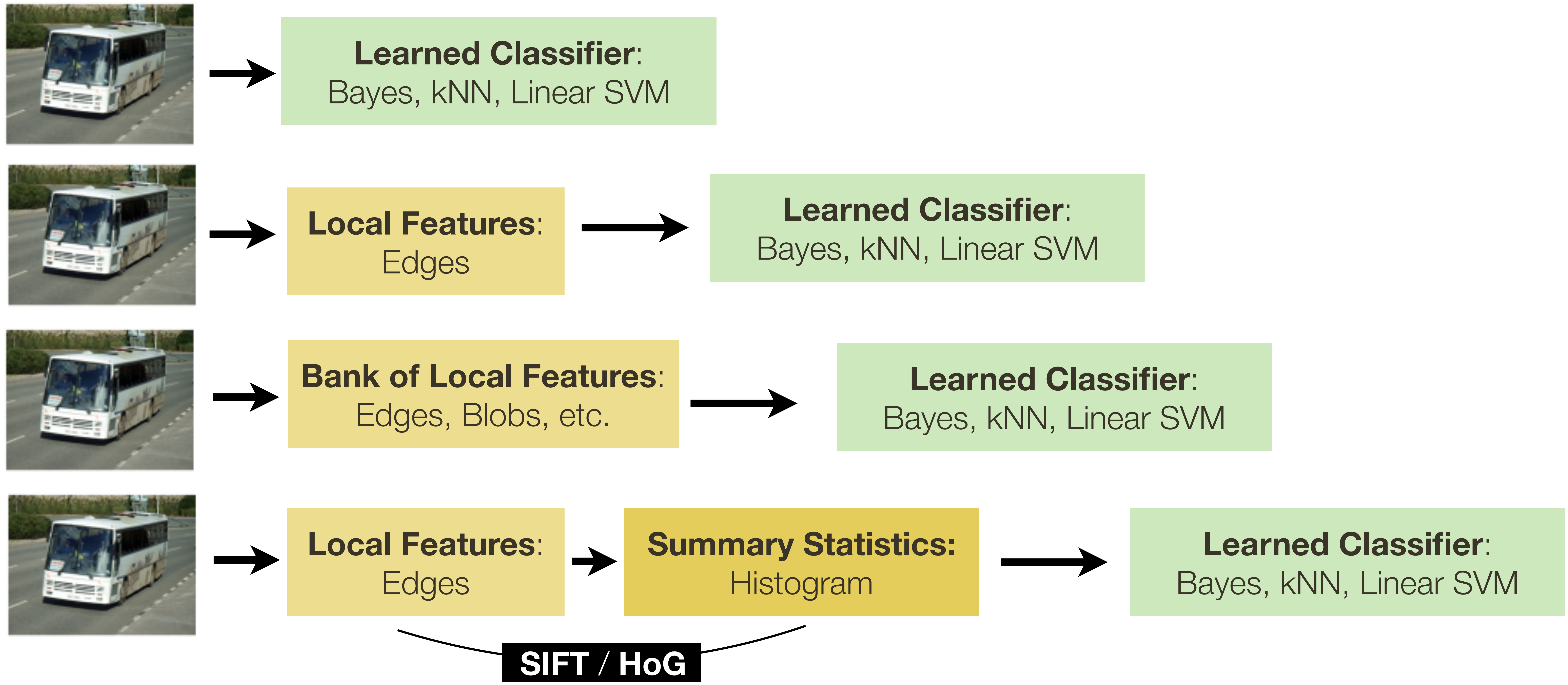
— Empirically engineered features with desired properties

— Pragmatically defined models (classifiers) that either defined by hand or require test time optimization

Recognition Overview: Learning



Recognition Overview: Learning



Bayes — estimate *parametric* form of distribution (requires training data) for each class

Recognition Overview: Learning

Learned Classifier: Bayes, kNN, Linear SVM

Local Features: Edges → **Learned Classifier**: Bayes, kNN, Linear SVM

Bank of Local Features: Edges, Blobs, etc. → **Learned Classifier**: Bayes, kNN, Linear SVM

Local Features: Edges → **Summary Statistics:** Histogram → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

Bayes — estimate *parametric* form of distribution (requires training data) for each class

kNN — *non-parametric* form of distribution (requires training data) for each class

More expressive

Recognition Overview: Learning

 → **Learned Classifier**: Bayes, kNN, Linear SVM

 → **Local Features**: Edges → **Learned Classifier**: Bayes, kNN, Linear SVM

 → **Bank of Local Features**: Edges, Blobs, etc. → **Learned Classifier**: Bayes, kNN, Linear SVM

 → **Local Features**: Edges → **Summary Statistics:** Histogram → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

Bayes — estimate *parametric* form of distribution (requires training data) for each class

kNN — *non-parametric* form of distribution (requires training data) for each class

More expressive

Linear SVM — *parametric* form of classifier (requires training data) with implicit feature selection / weighting

Recognition Overview

→ **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Local Features**: Edges → **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Bank of Local Features**: Edges, Blobs, etc. → **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Local Features**: Edges → **Summary Statistics:** Histogram → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

→ **Local Features**: Edges → **Summary Statistics:** Histogram → **K-means coding:** Bag of Words, VLAD → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

Recognition Overview

→ **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Local Features**: Edges → **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Bank of Local Features**: Edges, Blobs, etc. → **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Local Features**: Edges → **Summary Statistics:** Histogram → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

→ **Local Features**: Edges → **Summary Statistics:** Histogram → **K-means coding:** Bag of Words, VLAD → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

1. Now there is some **unsupervised** “learning” in feature extraction

Recognition Overview

→ **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Local Features**: Edges → **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Bank of Local Features**: Edges, Blobs, etc. → **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Local Features**: Edges → **Summary Statistics:** Histogram → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

→ **Local Features**: Edges → **Summary Statistics:** Histogram → **K-means coding:** Bag of Words, VLAD → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

1. Now there is some **unsupervised** “learning” in feature extraction
2. Histogram of histograms of gradients (i.e., simple **hierarchical aggregation**)

Recognition Overview

→ **Learned Classifier**: Bayes, kNN, Linear SVM

3. Features are still **not tuned for any specific task** (features for object vs. scene classification are exactly same) only classifier can be tuned

→ **Local Features**: Edges → **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Bank of Local Features**: Edges, Blobs, etc. → **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Local Features**: Edges → **Summary Statistics:** Histogram → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

→ **Local Features**: Edges → **Summary Statistics:** Histogram → **K-means coding:** Bag of Words, VLAD → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

1. Now there is some **unsupervised** "learning" in feature extraction
2. Histogram of histograms of gradients (i.e., simple **hierarchical aggregation**)

Recognition Overview: Convolutional Neural Nets (next week)

Deeper hierarchies of features (obtained by learned filters) **learned together with the classifier** for a specific task (classification, detection, segmentation)

→ **Learned Features**: Filters → **Learned Features**: Filters → **Learned Features**: Filters → **Learned Features**: Filters → **Learned Features**: Filters → **Learned Classifier**: Bayes, kNN, Linear SVM

→ **Local Features**: Edges → **Summary Statistics:** Histogram → **K-means coding:** Bag of Words, VLAD → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

1. Now there is some **unsupervised** "learning" in feature extraction
2. Histogram of histograms of gradients (i.e., simple **hierarchical aggregation**)

Recognition Overview: Foundational Models

1. "Pre-training" (optimizing) in an unsupervised / self-supervised manner (to get good feature extractors)

→ **Learned Features:** Filters → **Learned Features:** Filters → **Learned Features:** Filters → **Learned Features:** Filters → **Learned Features:** Filters → **Pre-text Tasks we don't really care about**

2. "Fine-tuning" (optimizing again from a warm start) to get good performance on the task

→ **Learned Features:** Filters → **Learned Features:** Filters → **Learned Features:** Filters → **Learned Features:** Filters → **Learned Features:** Filters → **Learned Classifier:** Bayes, kNN, Linear SVM

→ **Local Features:** Edges → **Summary Statistics:** Histogram → **K-means coding:** Bag of Words, VLAD → **Learned Classifier:** Bayes, kNN, Linear SVM

SIFT / HoG

1. Now there is some **unsupervised** "learning" in feature extraction
2. Histogram of histograms of gradients (i.e., simple **hierarchical aggregation**)

Let's do a bit of a case study …

 → **Learned Classifier**: Bayes, kNN, Linear SVM

 → **Local Features**: Edges → **Learned Classifier**: Bayes, kNN, Linear SVM

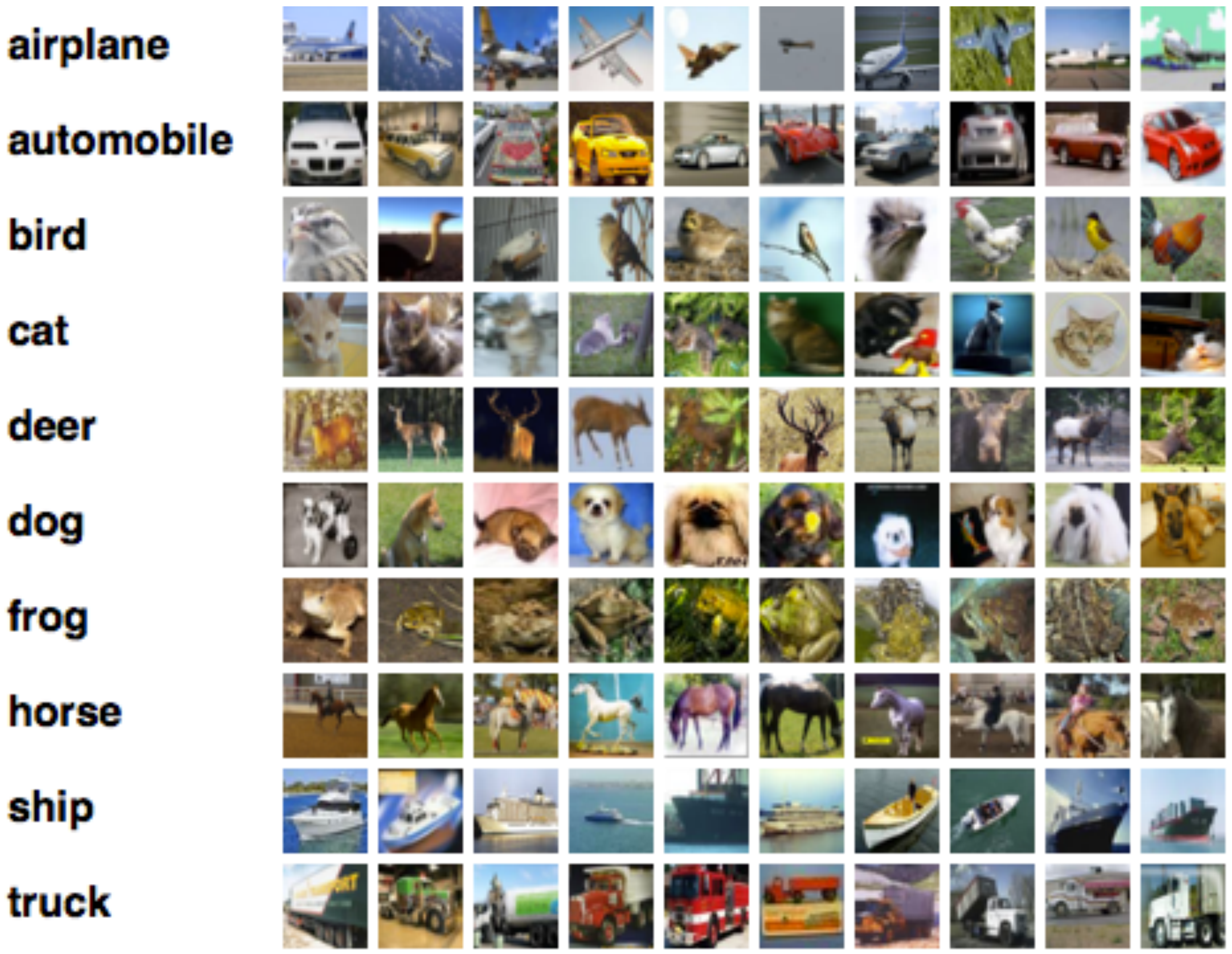
 → **Bank of Local Features**: Edges, Blobs, etc. → **Learned Classifier**: Bayes, kNN, Linear SVM

 → $\underbrace{\textbf{Local Features: } \text{Edges} \rightarrow \textbf{Summary Statistics: } \text{Histogram}}_{\textbf{SIFT / HoG}}$ → **Learned Classifier**: Bayes, kNN, Linear SVM

 → $\underbrace{\textbf{Local Features: } \text{Edges} \rightarrow \textbf{Summary Statistics: } \text{Histogram}}_{\textbf{SIFT / HoG}}$ → **K-means coding:** Bag of Words, VLAD → **Learned Classifier**: Bayes, kNN, Linear SVM

CIFAR10 Dataset

- Hand labelled set of 10 categories from Tiny Images dataset
- 60,000 32x32 images in 10 classes (50k train, 10k test)

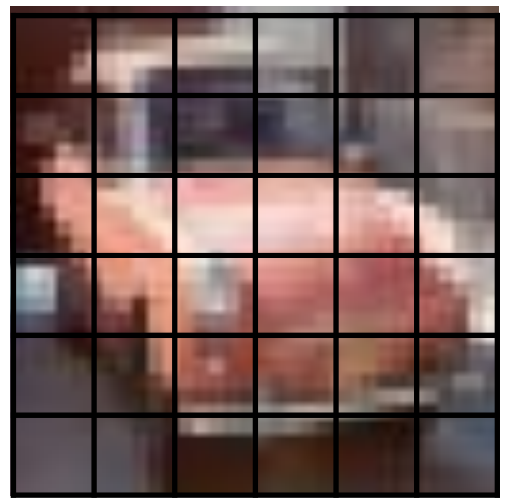


Good test set for visual recognition problems

CIFAR10 Classification

Let's build an image classifier

Start by vectorizing the data x = 3072 element vector of 0-255



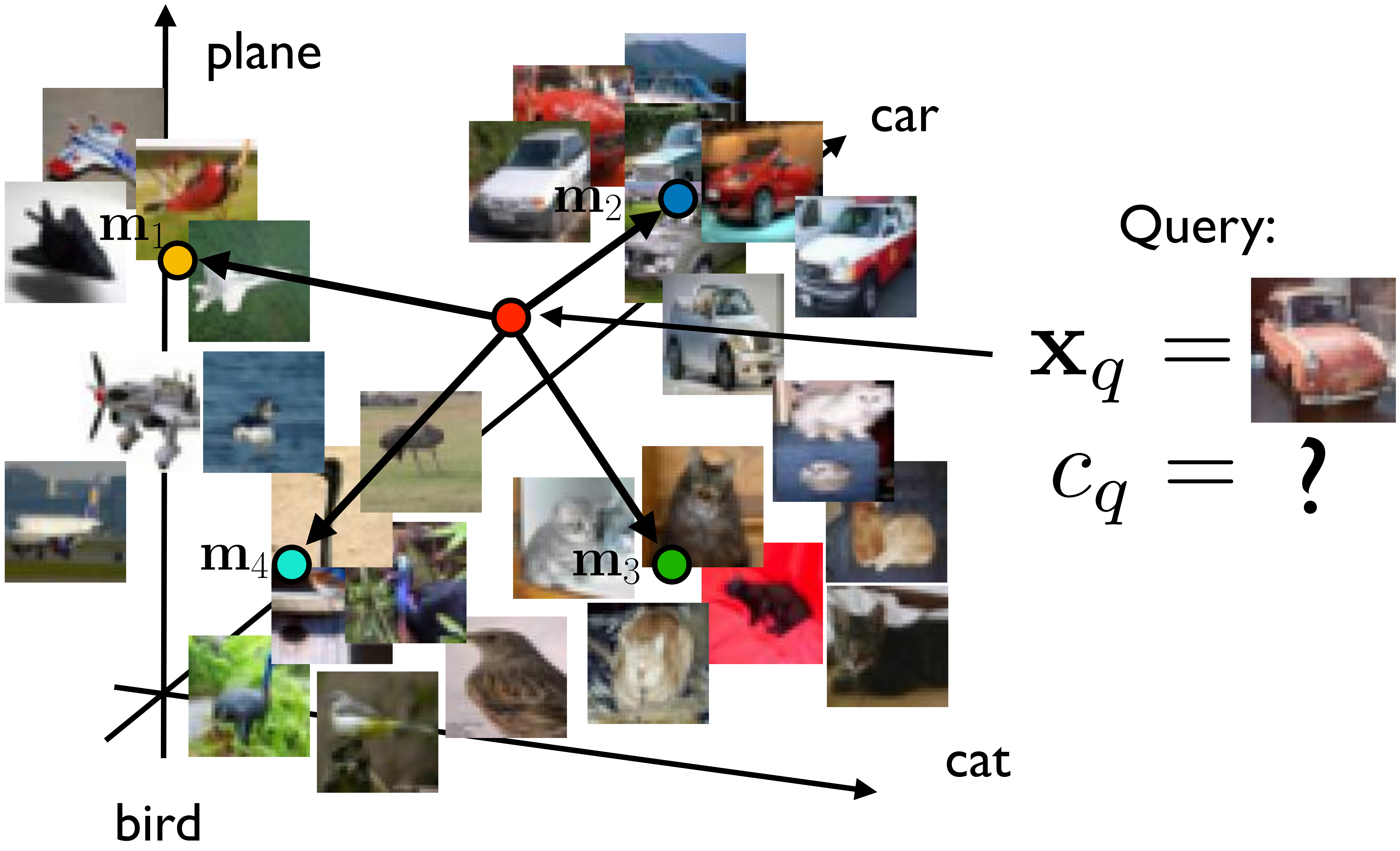
32 x 32 x RGB (8 bit) image →

x = [65 102 33 57 54 …]

x = 3072 element vector of 0-255

Nearest Mean Classifier

Compute a single "average" template per class

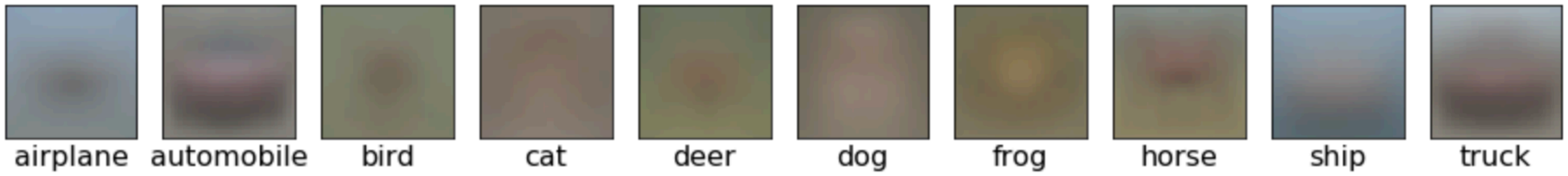


Nearest Mean Classifier

Find the nearest mean and assign class:

$$c_q = \arg\min_i |\mathbf{x}_q - \mathbf{m}_i|^2$$

CIFAR10 class means:

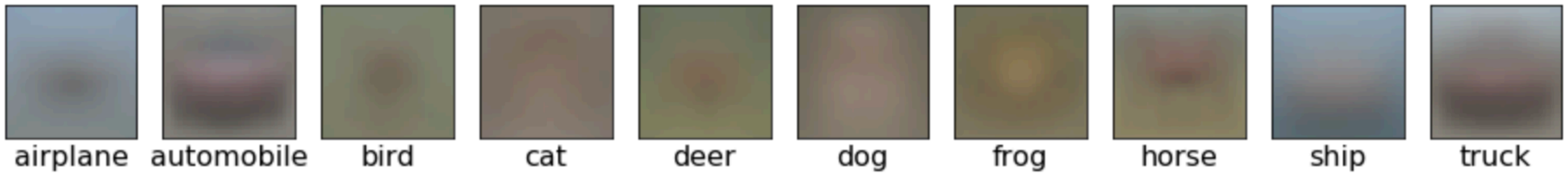


Nearest Mean Classifier

Find the nearest mean and assign class:

$$c_q = \arg\min_i |\mathbf{x}_q - \mathbf{m}_i|^2$$

CIFAR10 class means:

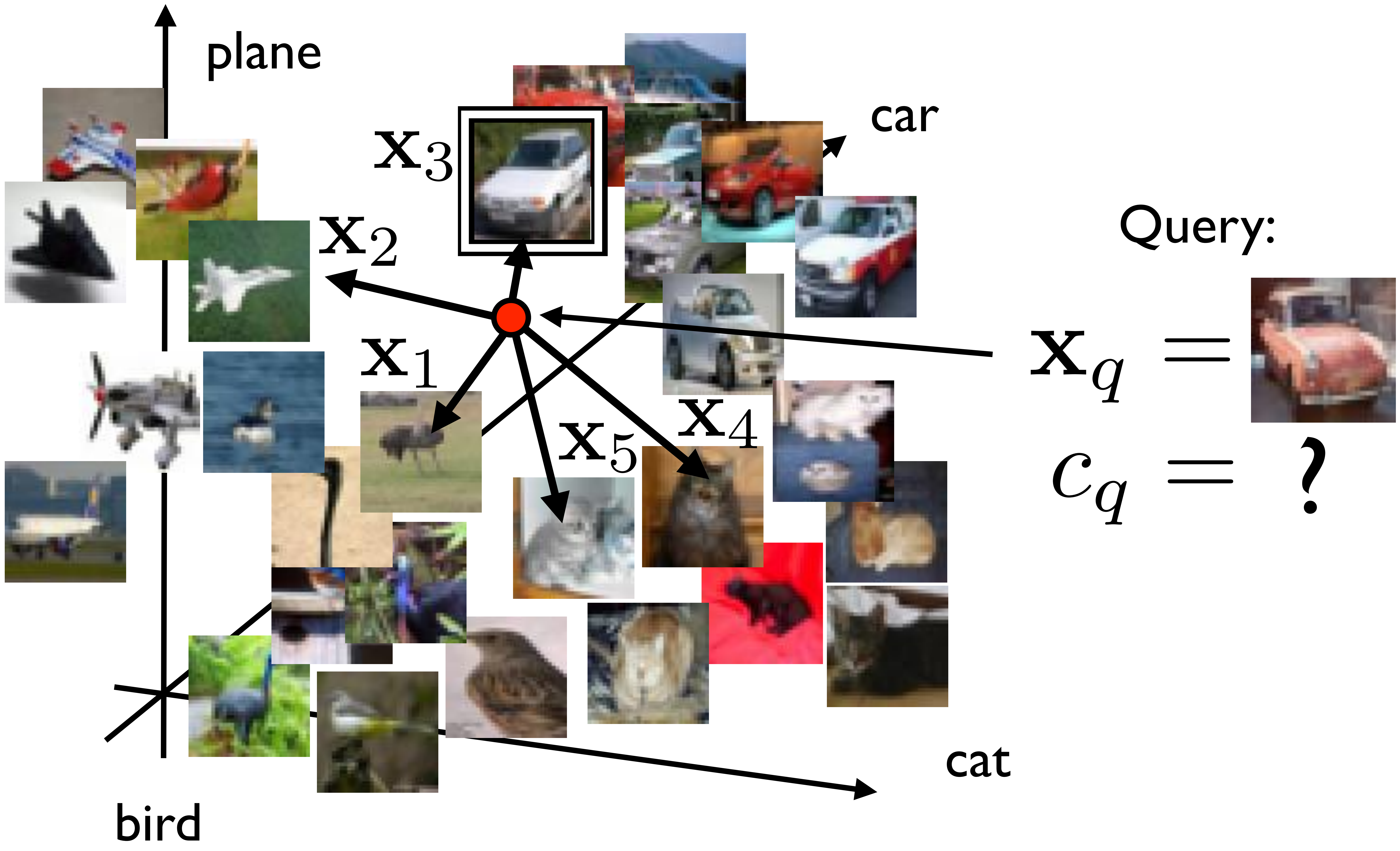


Performance:

Chance performance:	10%
Human performance:	~94%
Nearest Mean Classifier (pixels):	**37%**

Nearest Neighbor Classifier

We can view each image as a point in a high dimensional space



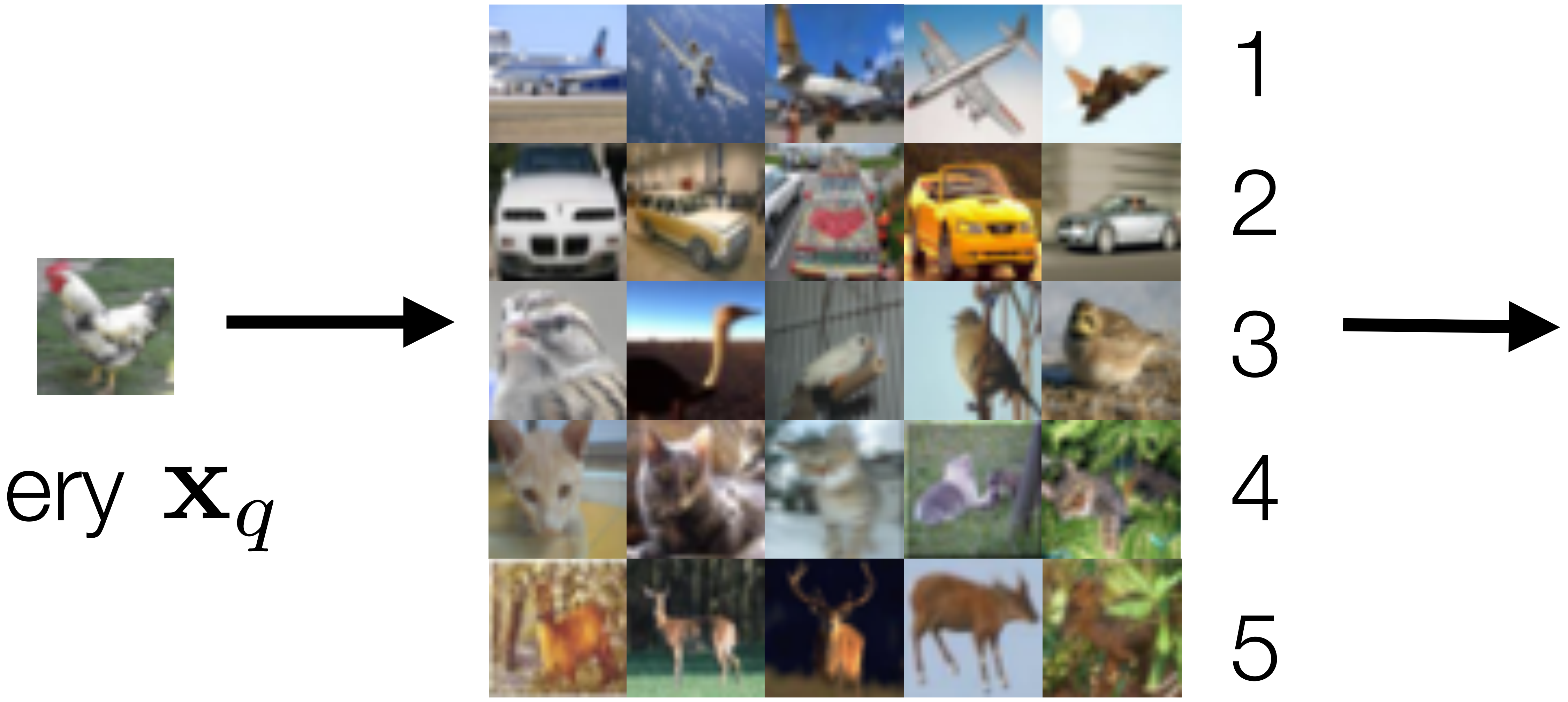
Nearest Neighbor Classifier

Find nearest neighbour in training set:

$$i_{NN} = \arg\min_i |\mathbf{x}_q - \mathbf{x}_i|$$

Assign class to class of the nearest neighbour:

$$\hat{y}(\mathbf{x}_q) = y(\mathbf{x}_{i_{NN}})$$



Result = 3

Query $\mathbf{x}_q$

Calculate $|\mathbf{x}_q - \mathbf{x}_i|$
for all training data

Nearest Neighbor Classifier

Find nearest neighbour in training set:

$$i_{NN} = \arg\min_i |\mathbf{x}_q - \mathbf{x}_i|$$

Assign class to class of the nearest neighbour:

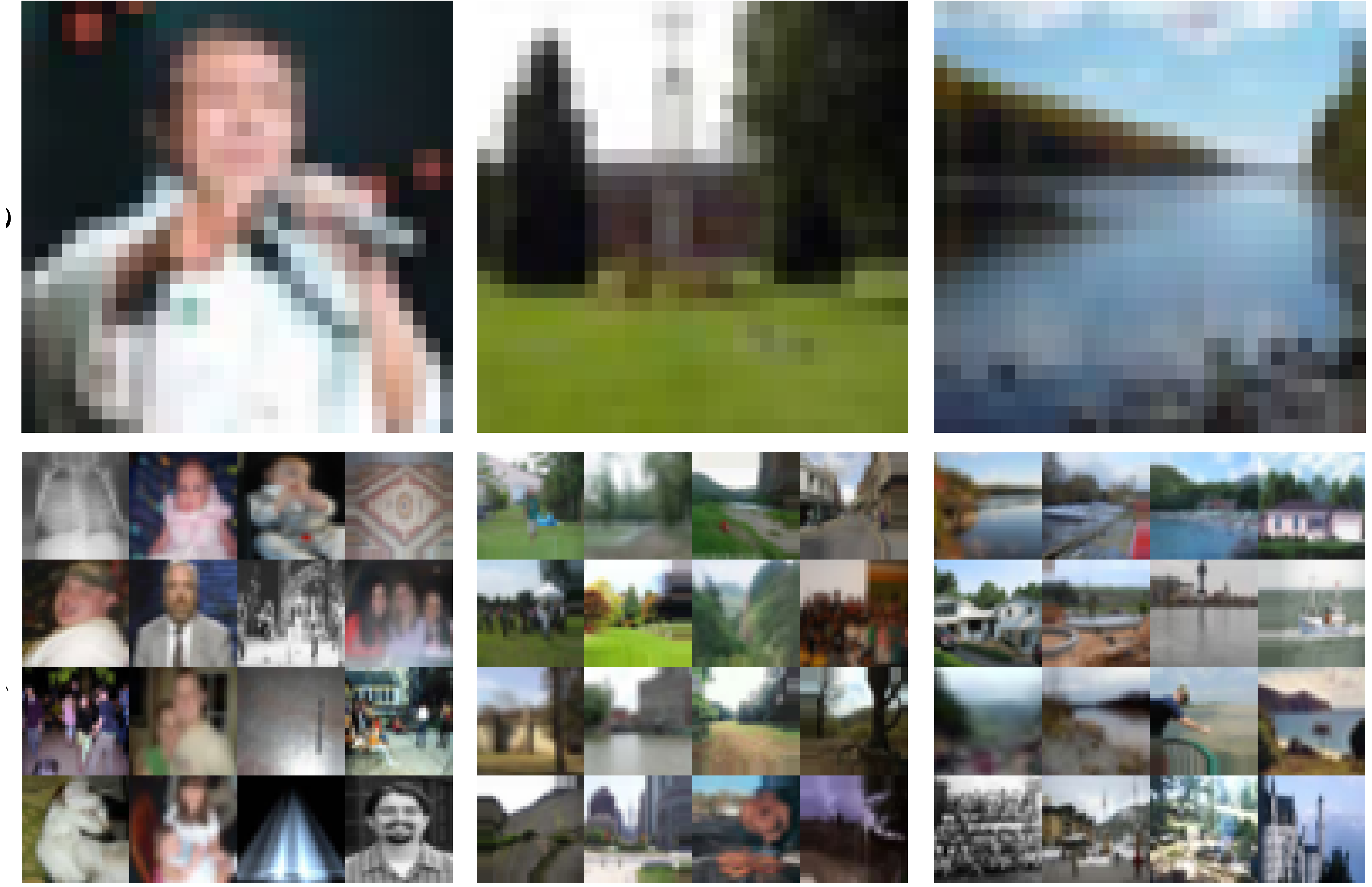
$$\hat{y}(\mathbf{x}_q) = y(\mathbf{x}_{i_{NN}})$$

Performance:

Chance performance:	10%
Human performance:	~94%
Nearest Neighbor (pixels):	**40.8%**
Nearest Neighbor (HoG):	**58.3%**

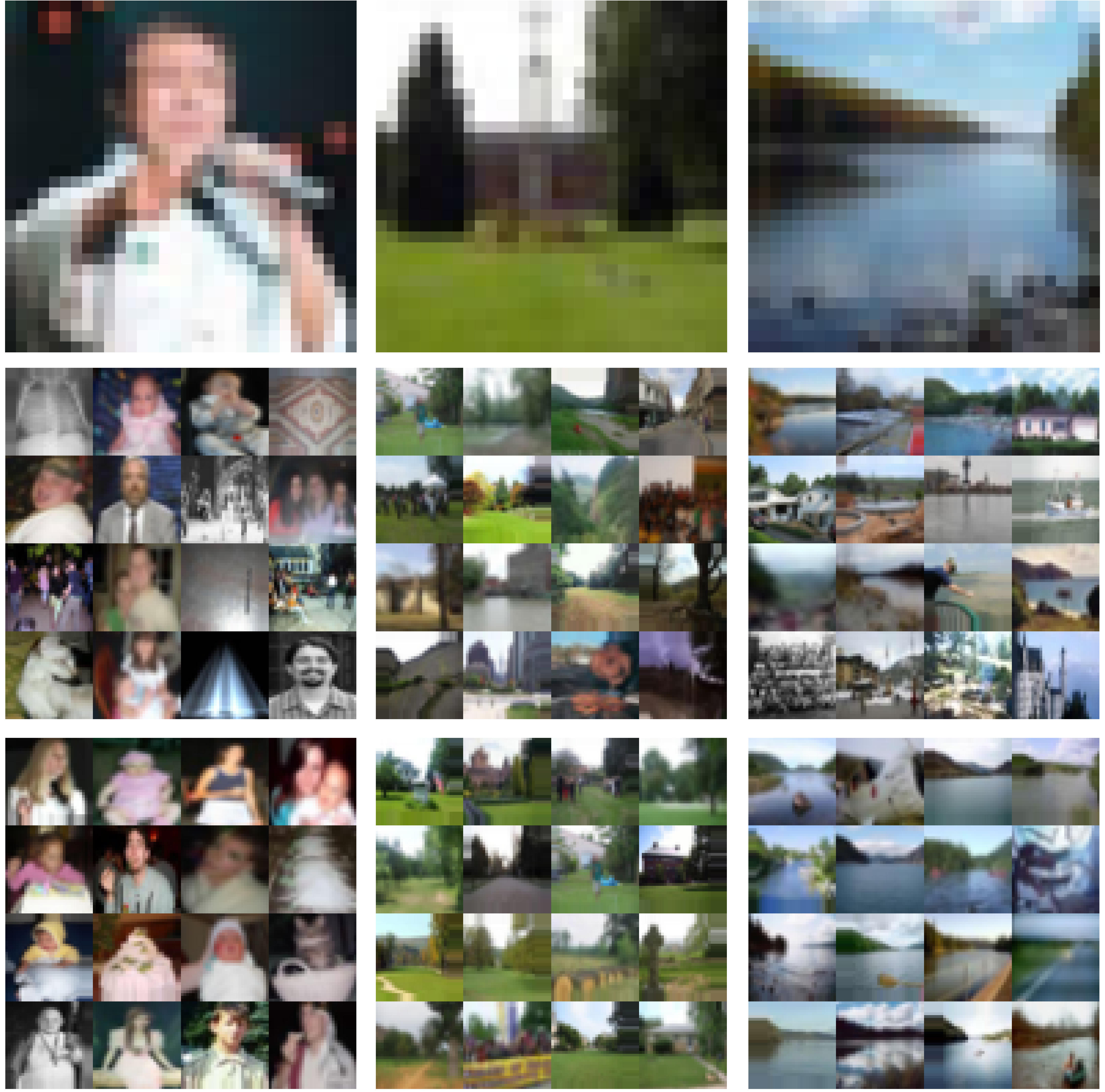
Source: https://cran.r-project.org/web/packages/KernelKnn/vignettes/image_classification_using_MNIST_CIFAR_data.html

Query



Query

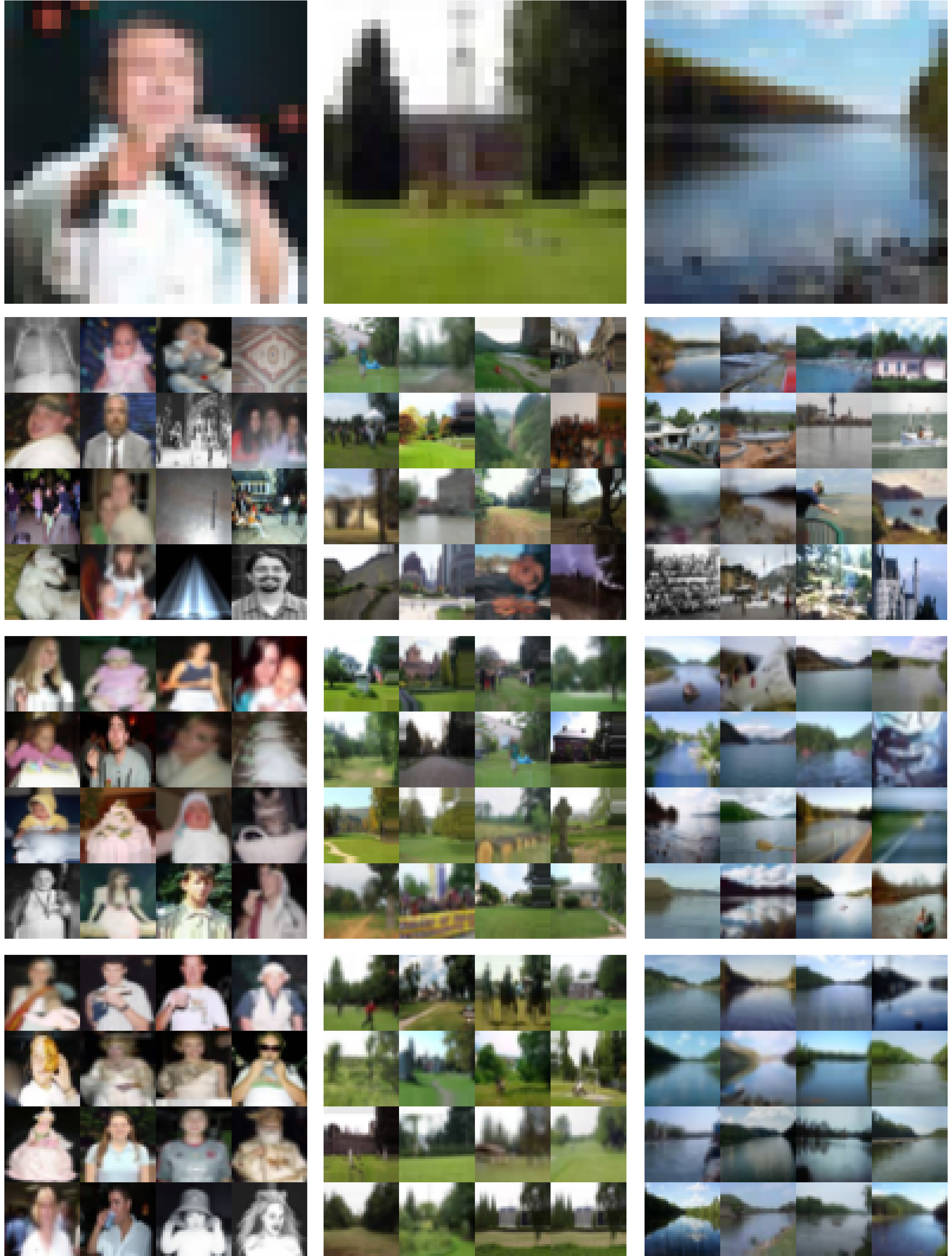
7900



Query

7900

790,000



Query

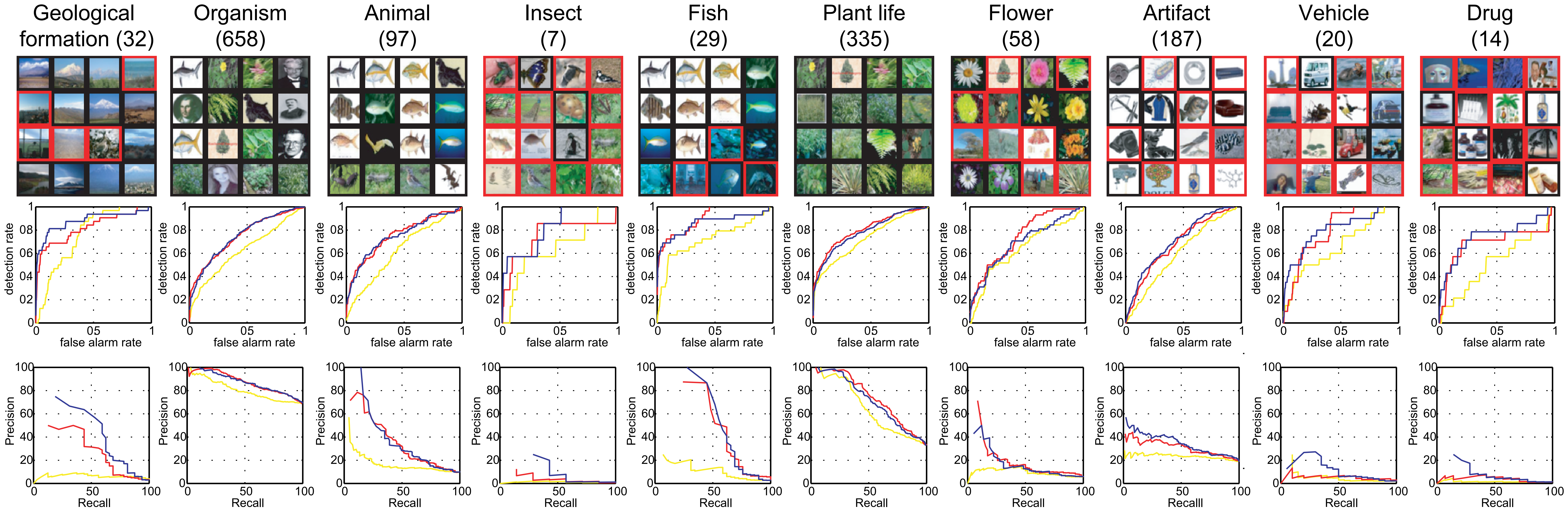
7900

790,000

79,000,000

Tiny Image Recognition

[Torralba, Fergus, Freeman '08]

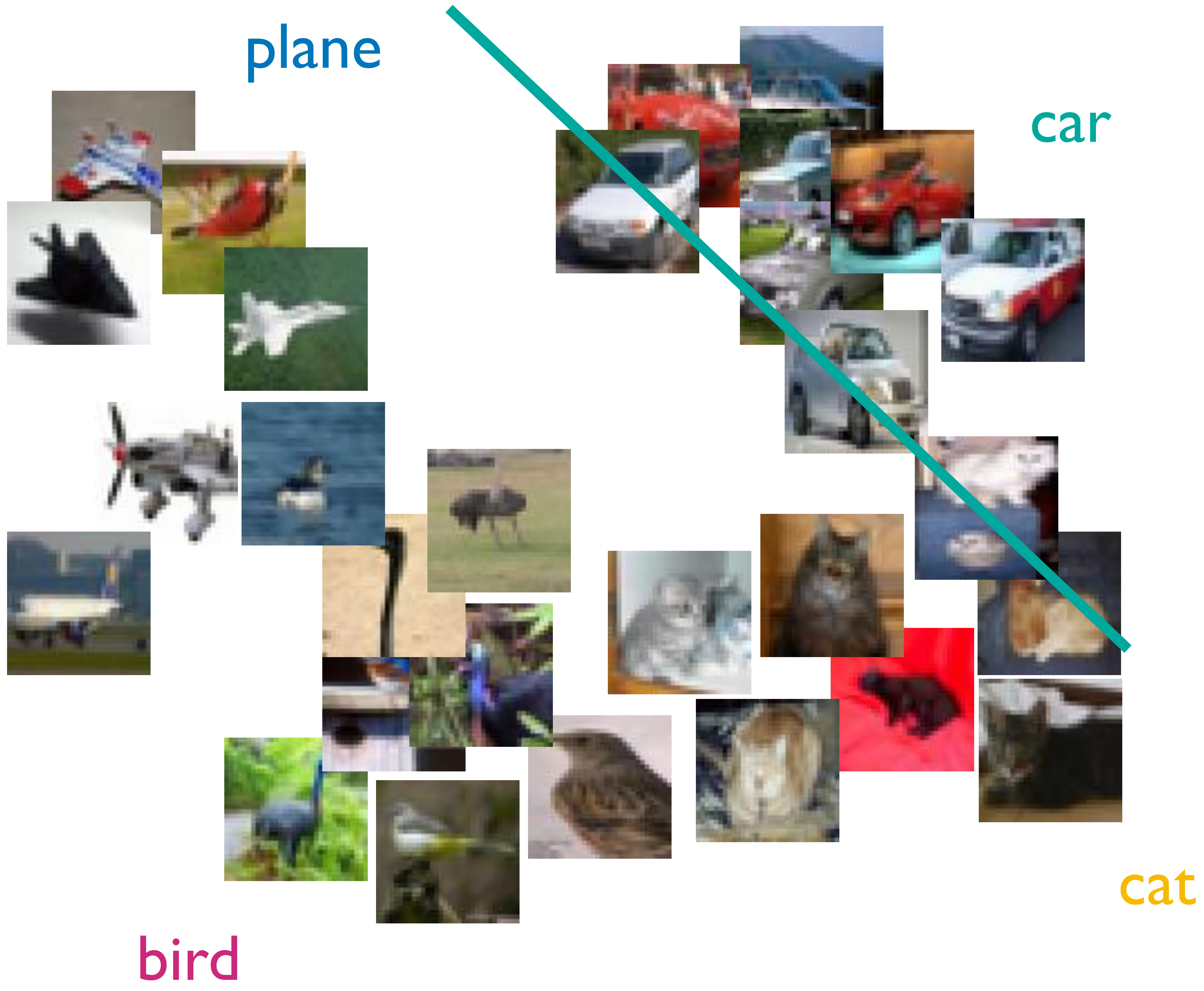


yellow = 7900, red = 790,000, blue = 79,000,000

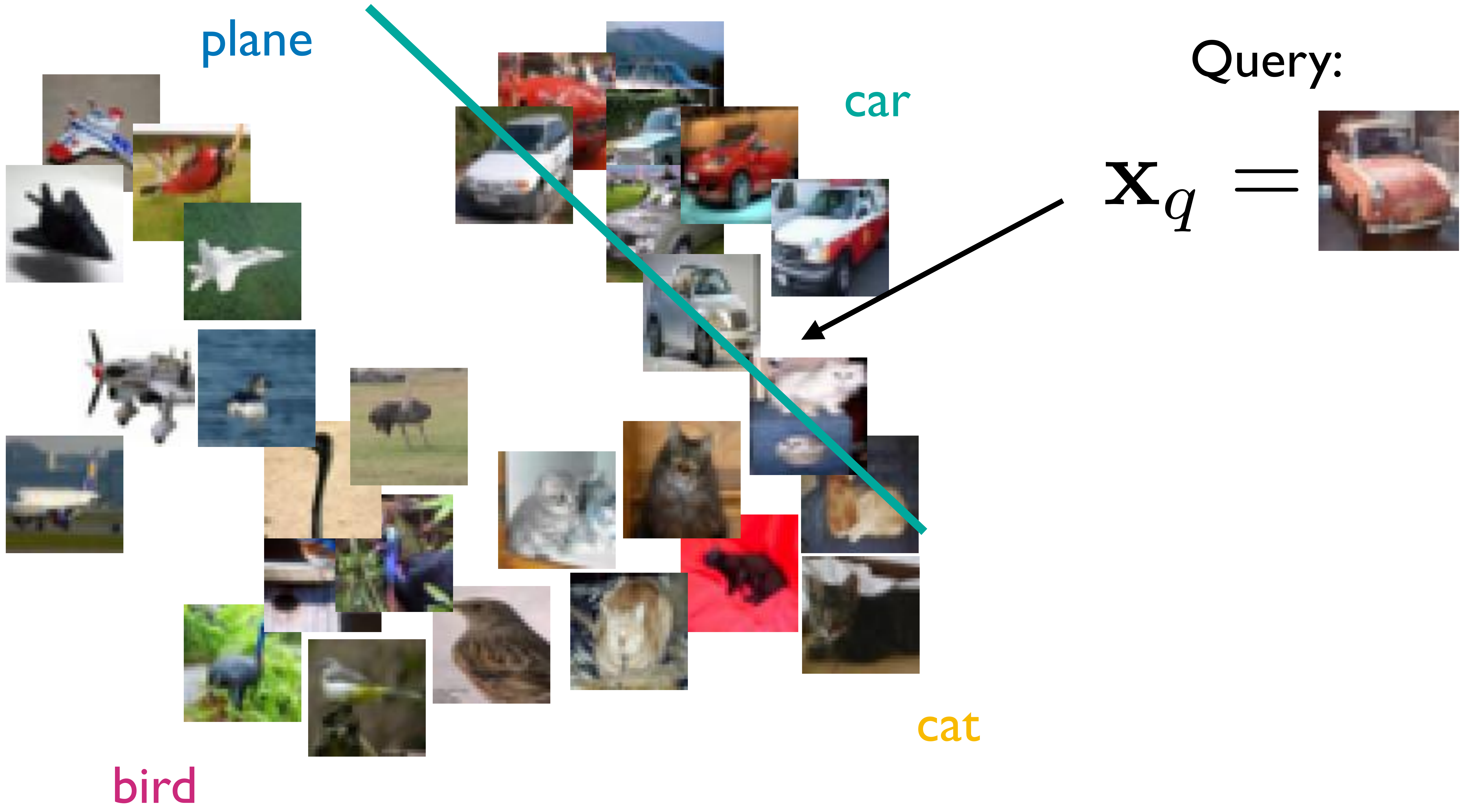
Nearest neighbour becomes increasingly accurate as N increases, but do we need to store a dataset of 80 million images?

1-vs-All **Linear SVM**

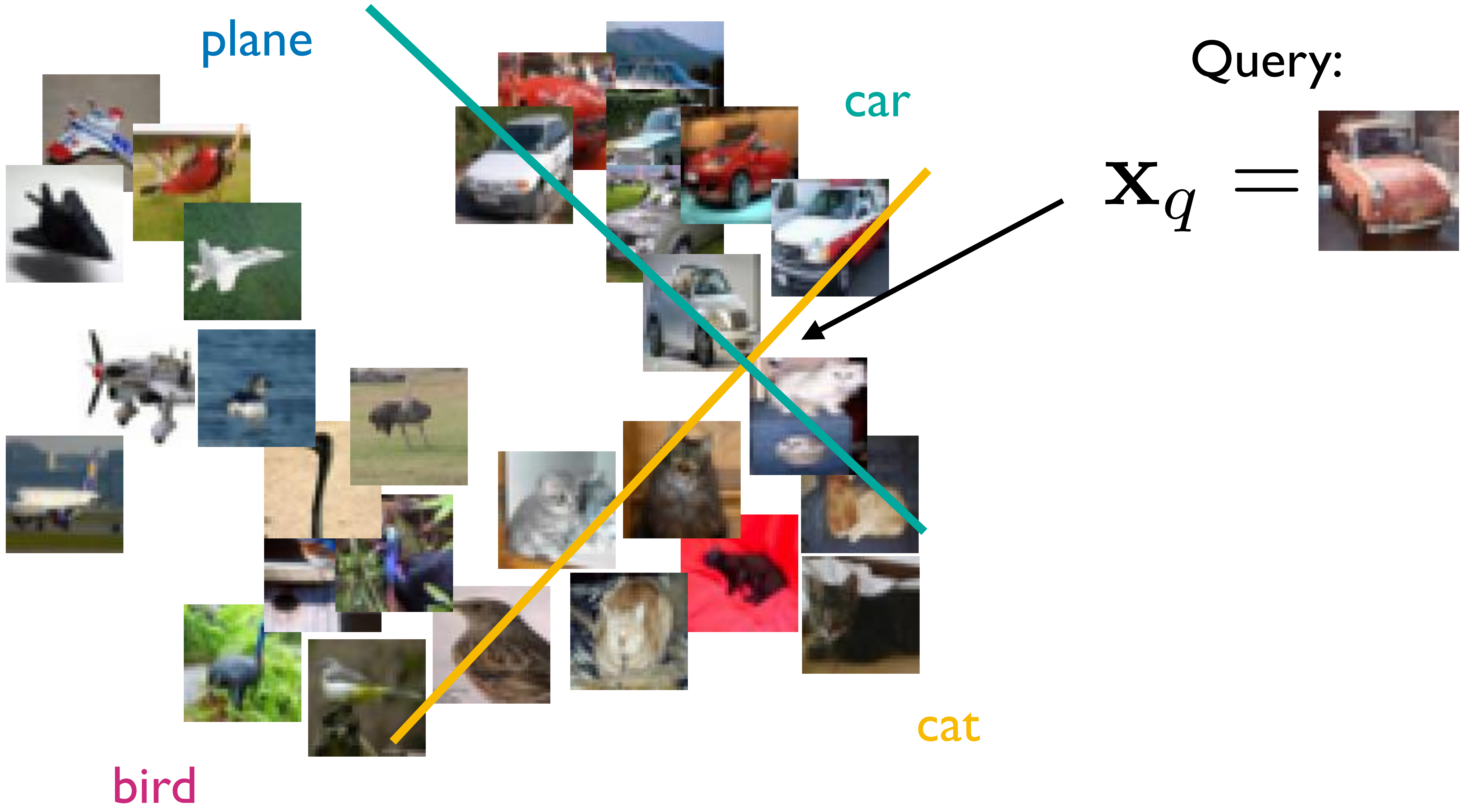
1-vs-All **Linear SVM**



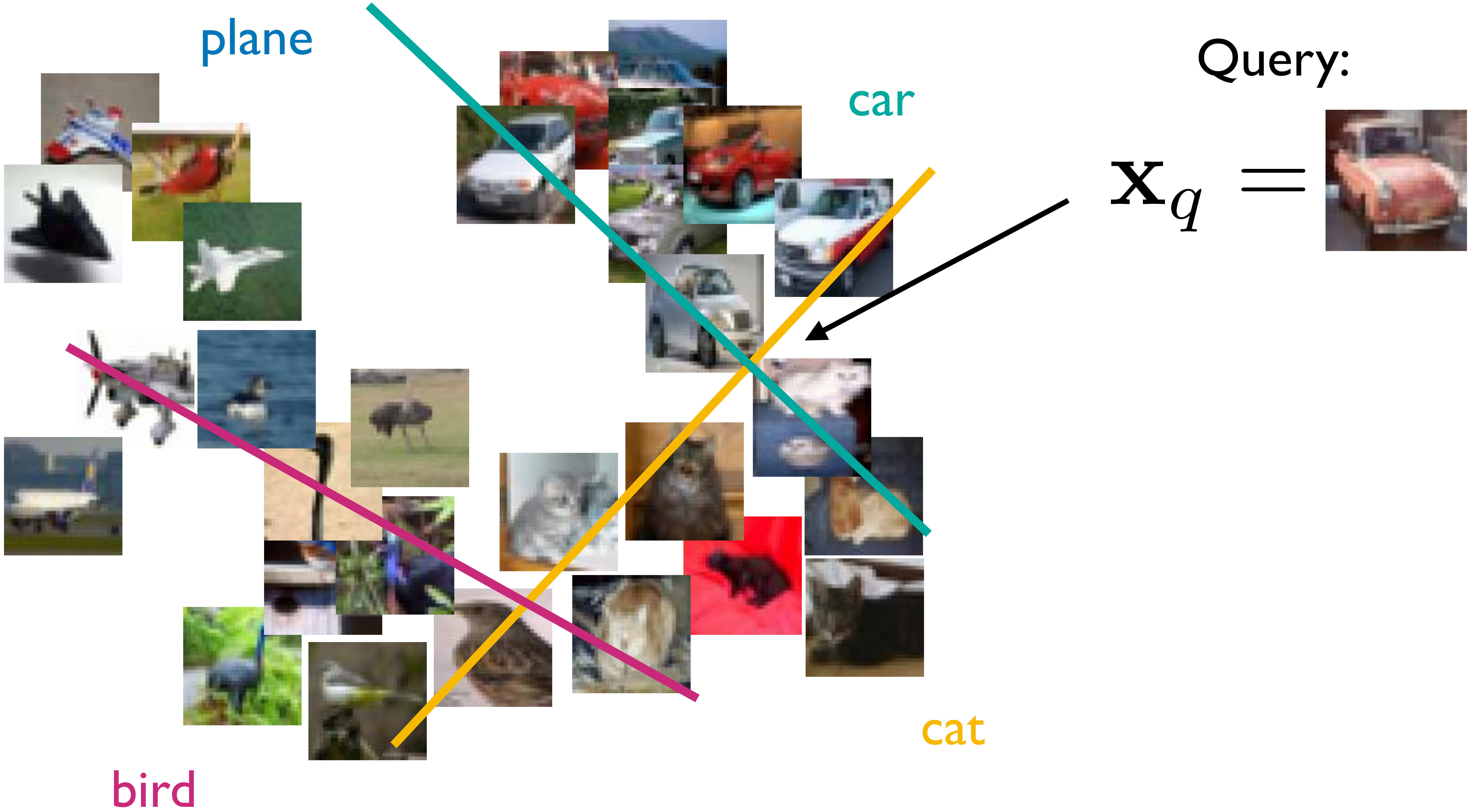
1-vs-All **Linear SVM**



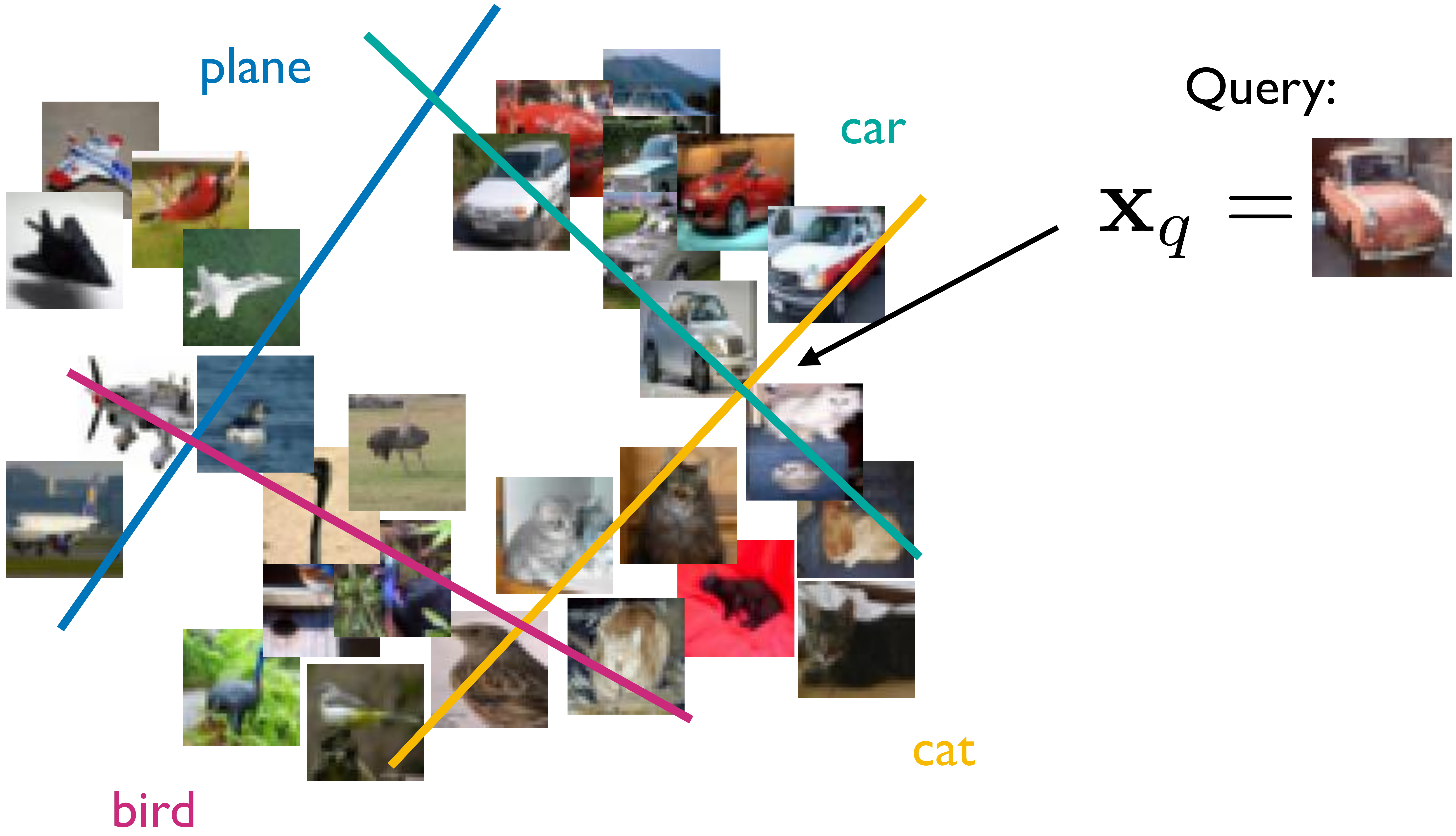
1-vs-All **Linear SVM**



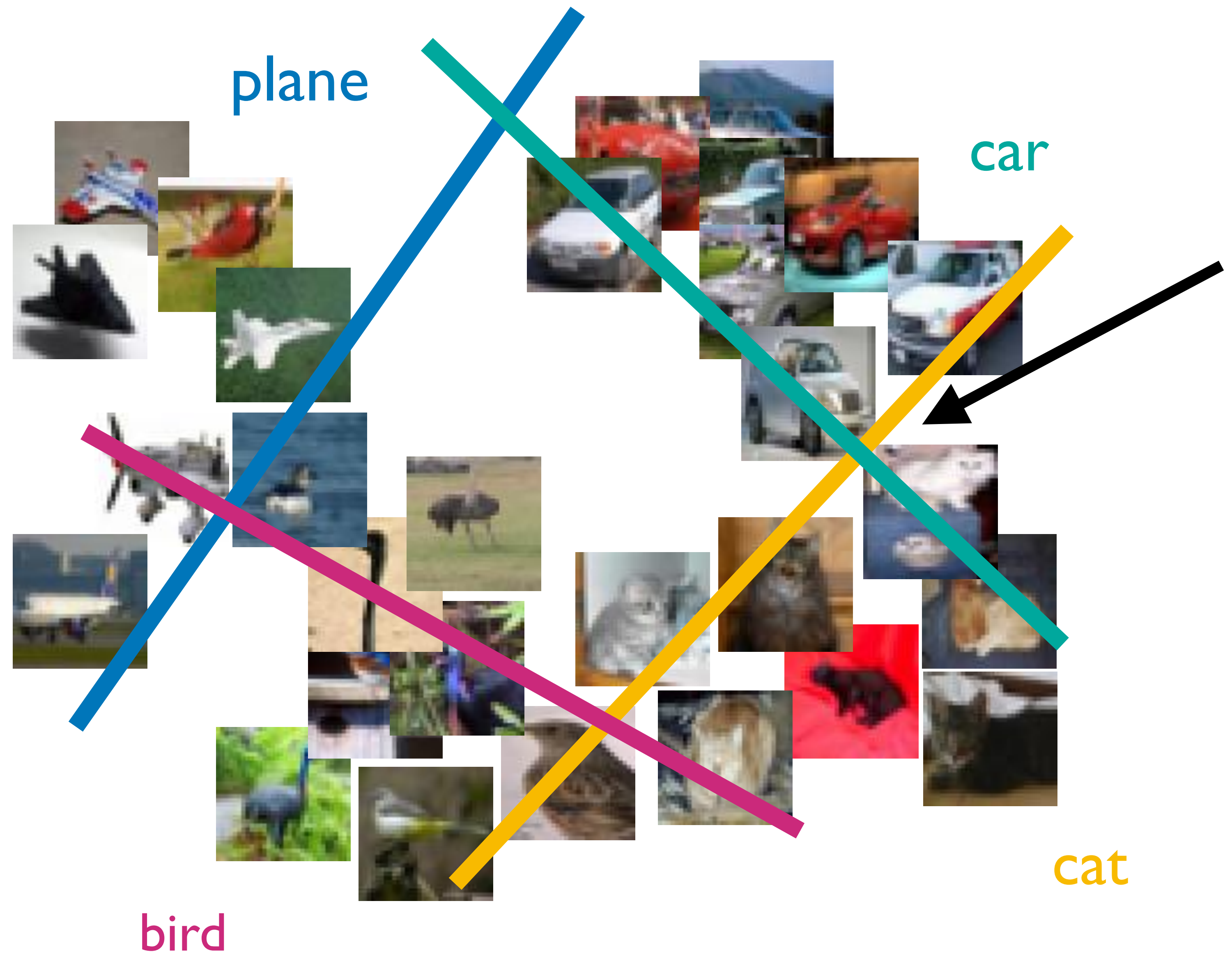
1-vs-All **Linear SVM**



1-vs-All **Linear SVM**



1-vs-All **Linear SVM**



$$\mathbf{x}_q =$$

Hard voting: $f_k(x) = \begin{cases} 1 & \text{if } k = \arg\min_j \|c^{(j)} - x\|_2^2 \\ 0 & \text{otherwise.} \end{cases}$

Soft voting: $f_k(x) = \max\{0, \mu(z) - z_k\}$

L2 distance to centroid k

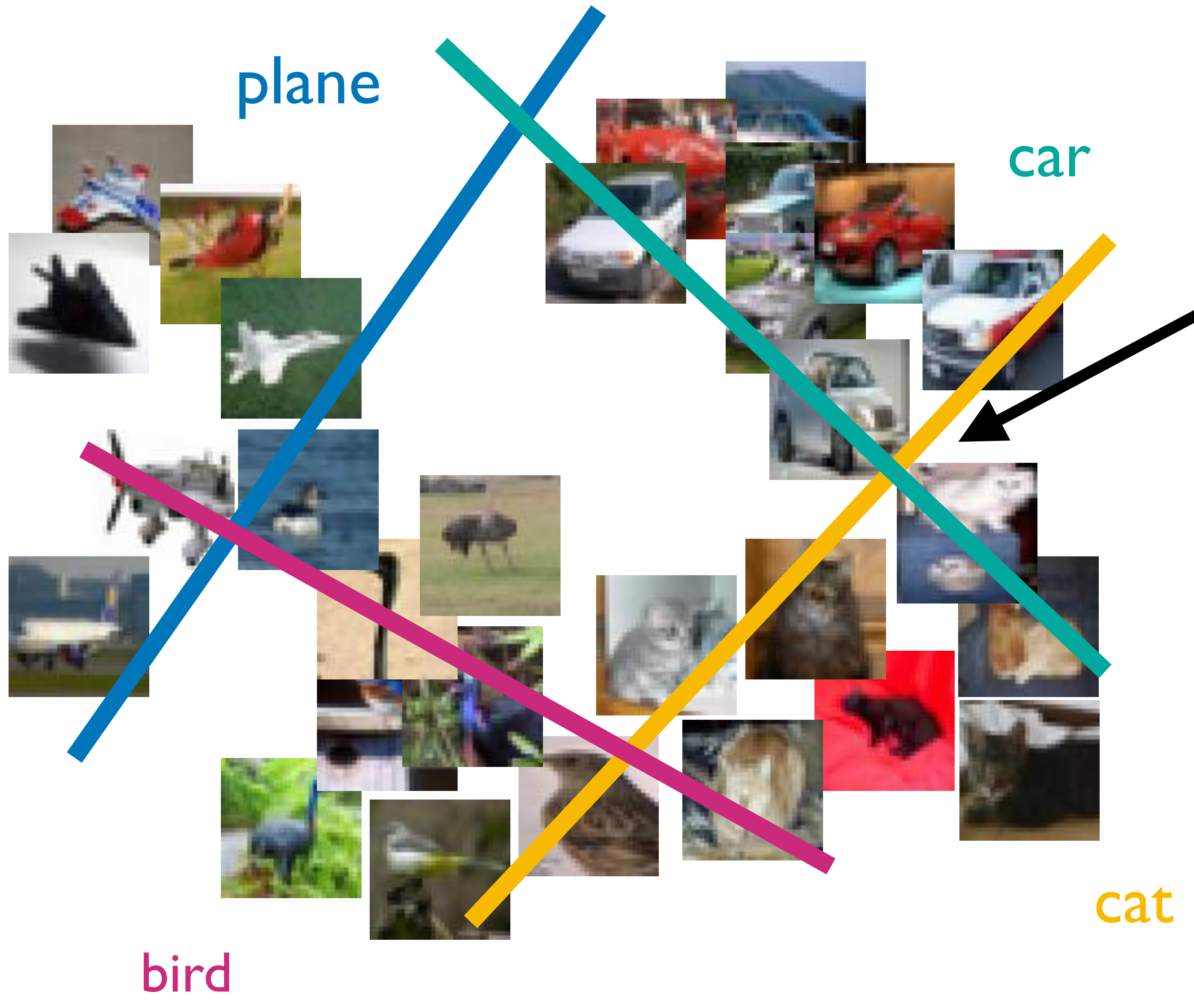
Performance:

Chance performance:	10%
Human performance:	~94%
Linear SVM (pixels):	**37.3**% [2] / **39.5**%* [1]
Linear SVM (SIFT):	**65.6**%* [1]
Linear SVM (BoW /w SIFT, 1600 words, hard voting):	**68.6**% [2]
Linear SVM (BoW /w SIFT, 1600 words, soft voting):	**77.9**% [2]
Linear SVM (BoW /w SIFT, 4000 words, soft voting):	**79.6**% [2]

[1] https://proceedings.neurips.cc/paper_files/paper/2010/file/4558dbb6f6f8bb2e16d03b85bde76e2c-Paper.pdf

[2] https://cs.stanford.edu/~acoates/papers/coatesleeng_aistats_2011.pdf

Deep Learning



Query:

$\mathbf{x}_q =$ 

Performance:

Chance performance:	10%
Human performance:	~94%
Linear SVM (pixels):	**37.3%** [2] / **39.5%***[1]
Linear SVM (SIFT):	**65.6%***[1]
Linear SVM (BoW /w SIFT, 1600 words, hard voting):	**68.6%** [2]
Linear SVM (BoW /w SIFT, 1600 words, soft voting):	**77.9%** [2]
Linear SVM (BoW /w SIFT, 4000 words, soft voting):	**79.6%** [2]
*Convolutional Neural Net (CNN):	**91.3%** [3]
*DINO [Caron et al., 2021]:	**94.4%** [3]
*RandSAC [Hua et al., 2023]:	**96.9%** [3]

[3] https://arxiv.org/pdf/2203.12054.pdf

Take home **messages** …

— Both **classification** and **feature representation** play significant role

— Classifiers need to be **expressive** to do well, but so do the **features**

— **Parametric classifiers** are much easier to work with (they are faster)

— Which is more significant, in part, depends on the amount of available data

More complex classifiers …

Lets look at more expressive classifiers that, for example, explicitly do feature selection

→ **Local Features**: Edges → **Summary Statistics:** Histogram → **K-means coding:** Bag of Words, VLAD → **Learned Classifier**: Bayes, kNN, Linear SVM

SIFT / HoG

Back to **Classification**

Decision Tree

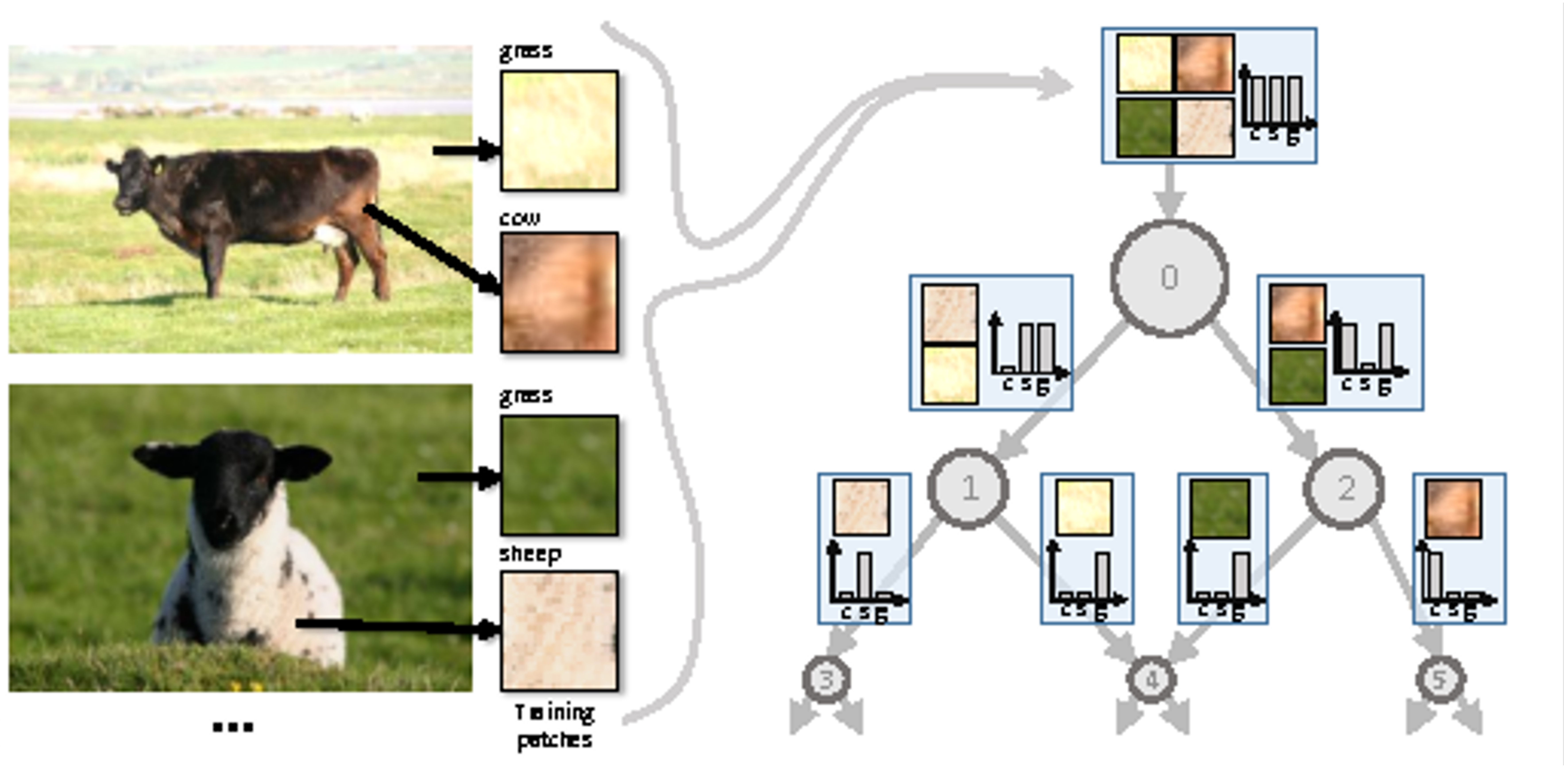
A **decision tree** is a simple non-linear parametric classifier

Consists of a tree in which each internal node is associated with a feature test

A data point starts at the root and recursively proceeds to the child node determined by the feature test, until it reaches a leaf node

The leaf node stores a class label or a probability distribution over class labels

Decision Tree



Decision Tree

Learning a decision tree from a training set involves selecting an efficient sequence of feature tests

Example: Waiting for a restaurant table

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

Decision Tree

Learning a decision tree from a training set involves selecting an efficient sequence of feature tests

Example: Waiting for a restaurant table

Is there an alternative restaurant near by?

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*\$\$\$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*\$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*\$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*\$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*\$\$\$*	*F*	*T*	*French*	>60	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*\$\$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*\$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*\$\$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*\$*	*T*	*F*	*Burger*	>60	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*\$\$\$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*\$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*\$*	*F*	*F*	*Burger*	*30–60*	*T*

Decision Tree

Learning a decision tree from a training set involves selecting an efficient sequence of feature tests

Example: Waiting for a restaurant table

Is there a bar at the restaurant?

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

Decision Tree

Learning a decision tree from a training set involves selecting an efficient sequence of feature tests

Example: Waiting for a restaurant table

Is it Friday night?

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

Decision Tree

Learning a decision tree from a training set involves selecting an efficient sequence of feature tests

Example: Waiting for a restaurant table

How many people in the restaurant?

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

Decision Tree

Which test is more helpful?

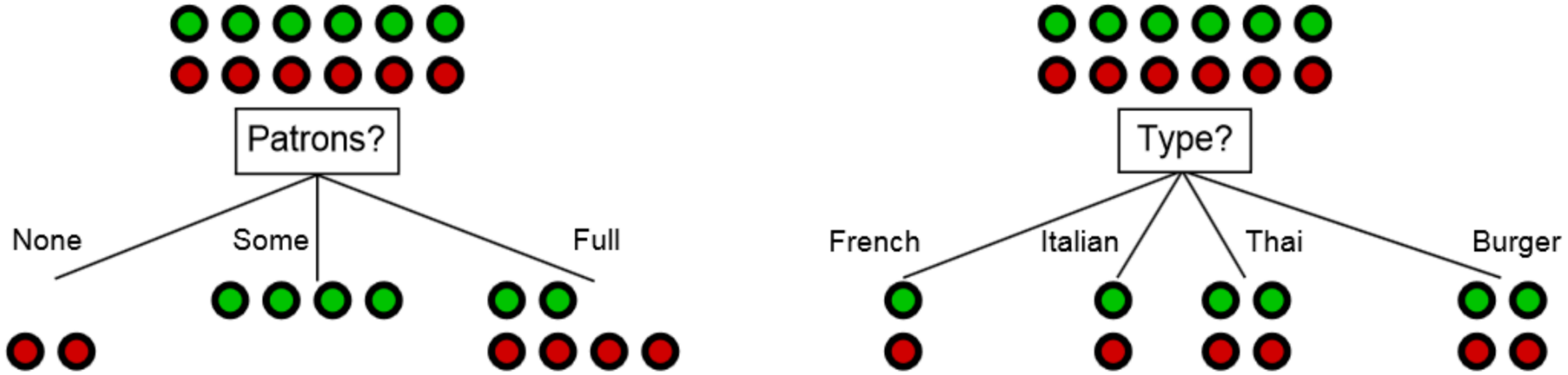


Figure credit: Russell and Norvig (3rd ed.)

Decision Tree

The **entropy** of a set S of data samples is defined as

$$H(S) = -\sum_{c \in C} p(c) \log(p(c))$$

where C is the set of classes represented in S, and $p(c)$ is the empirical distribution of class c in S

Entropy is highest when data samples are spread equally across all classes, and zero when all data samples are from the same class.

Entropy at each node …

Which test is more helpful?

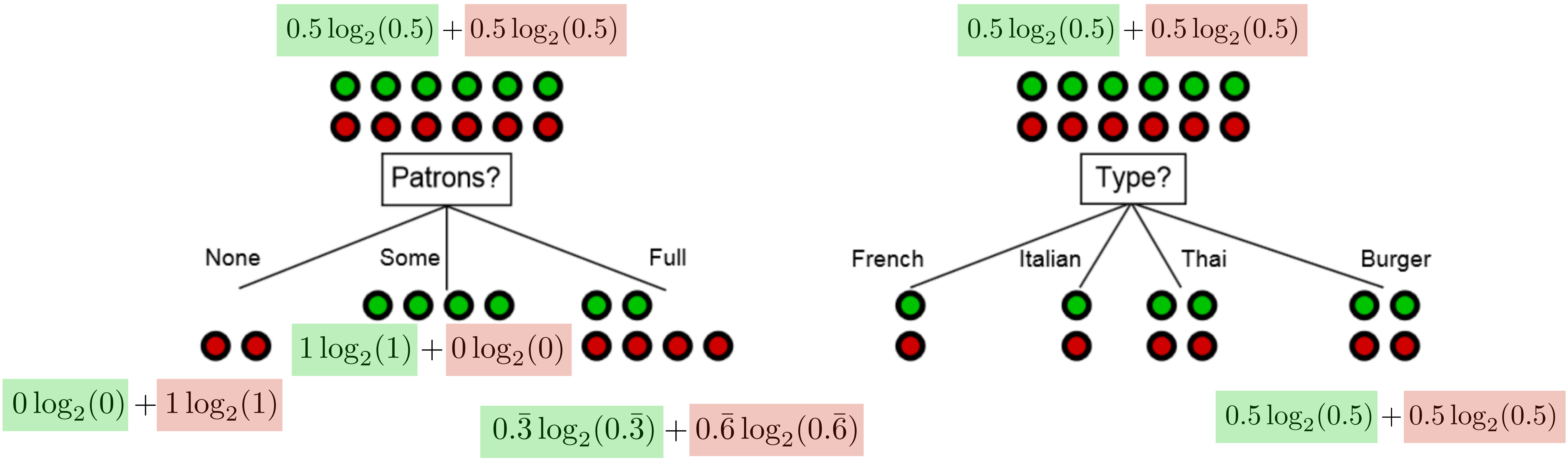


Figure credit: Russell and Norvig (3rd ed.)

Decision Tree

In general we try to select the feature test that maximizes the **information gain**:

$$I = H(S) - \sum_{i \in \{children\}} \frac{|S^i|}{|S|} H(S^i)$$

In the previous example, the information gains of the two candidate tests are:

$$I_{Patrons} = 0.541 \qquad\qquad I_{Type} = 0$$

So we choose the ‘Patrons’ test.

Decision Tree

In general we try to select the feature test that maximizes the **information gain**:

$$I = H(S) - \sum_{i \in \{children\}} \frac{|S^i|}{|S|} H(S^i)$$

In the previous example, the information gains of the two candidate tests are:

$$I_{Patrons} = 0.541 \qquad I_{Type} = 0$$

So we choose the ‘Patrons’ test.

Build a tree in a **greedy recursive** manner by maximizing information gain at each node

Decision Tree

Following this construction procedure we obtain the final decision tree:

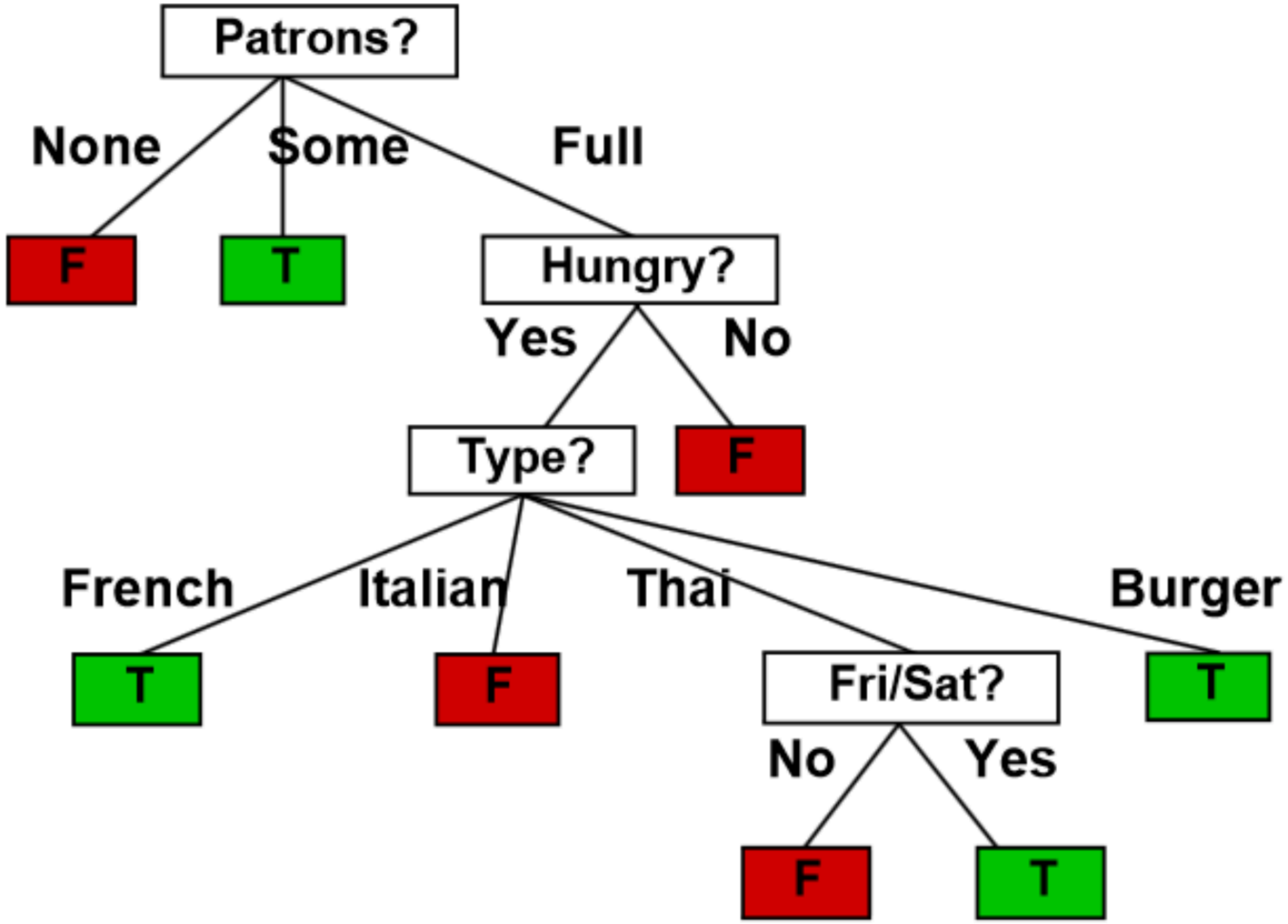


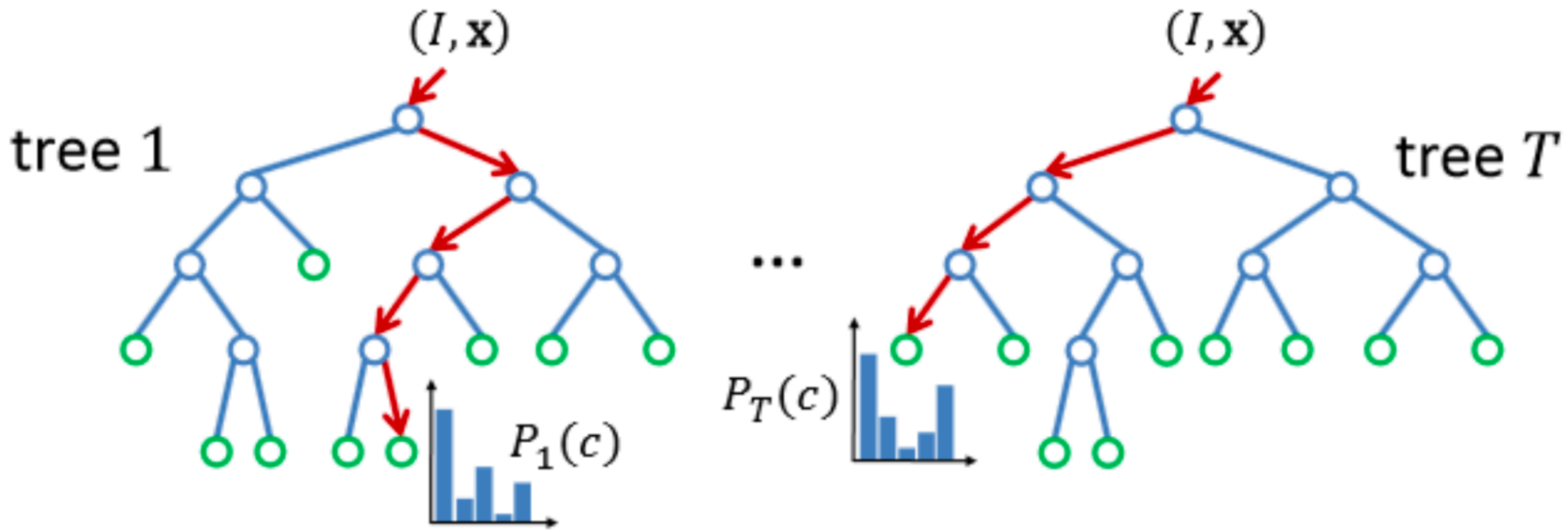
Figure credit: Russell and Norvig (3rd ed.)

Decision Tree

A **random forest** is an ensemble of decision trees.

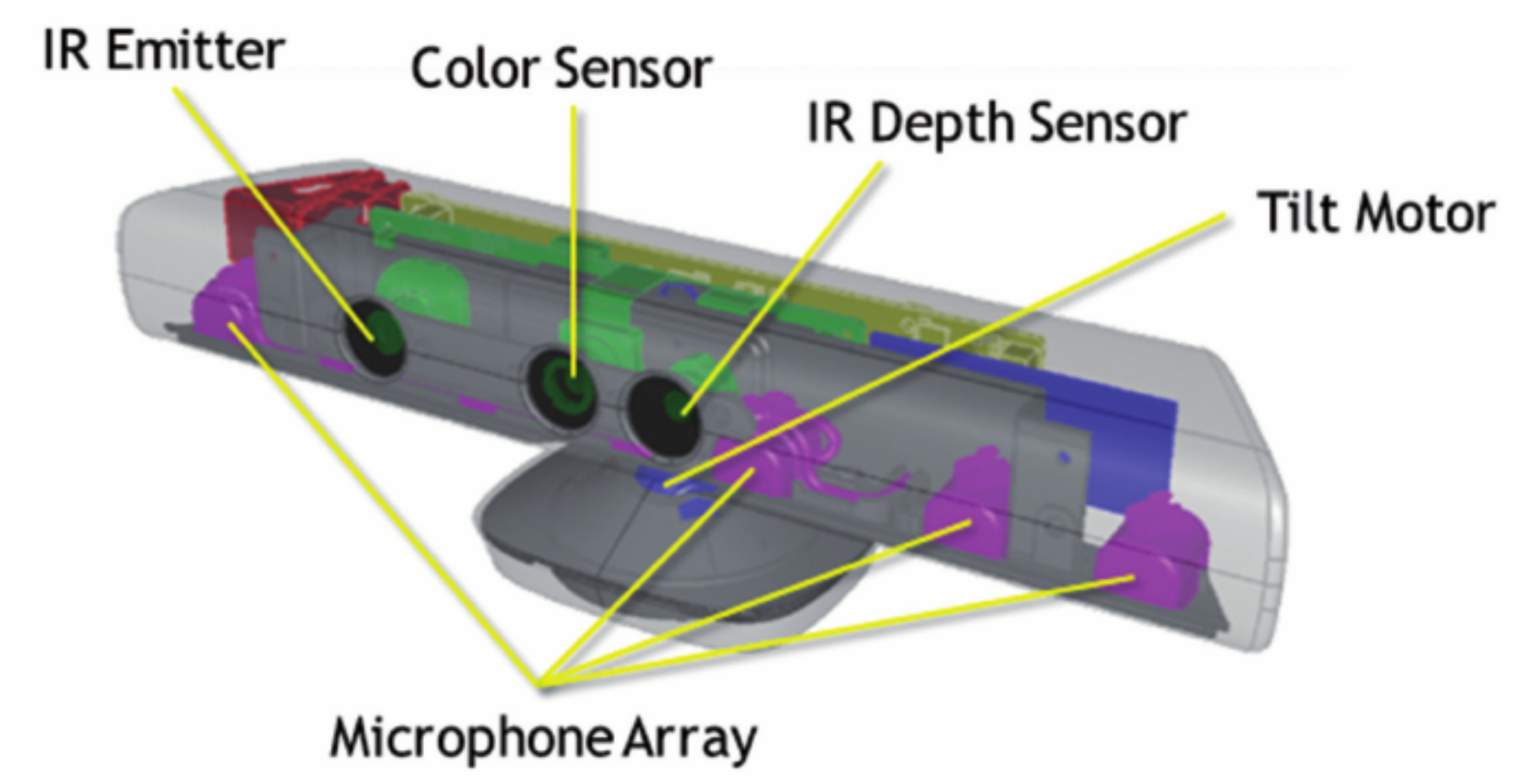
Randomness is incorporated via training set sampling and/or generation of the candidate binary tests

The prediction of the random forest is obtained by averaging over all decision trees.



Forsyth & Ponce (2nd ed.) Figure 14.19. Original credit: J. Shotton et al., 2011

Microsoft **Kinect**



Example 1: Kinect

Kinect allows users of Microsoft's Xbox 360 console to interact with games using natural body motions instead of a traditional handheld controller. The pose (joint positions) of the user is predicted using a random forest trained on depth features.

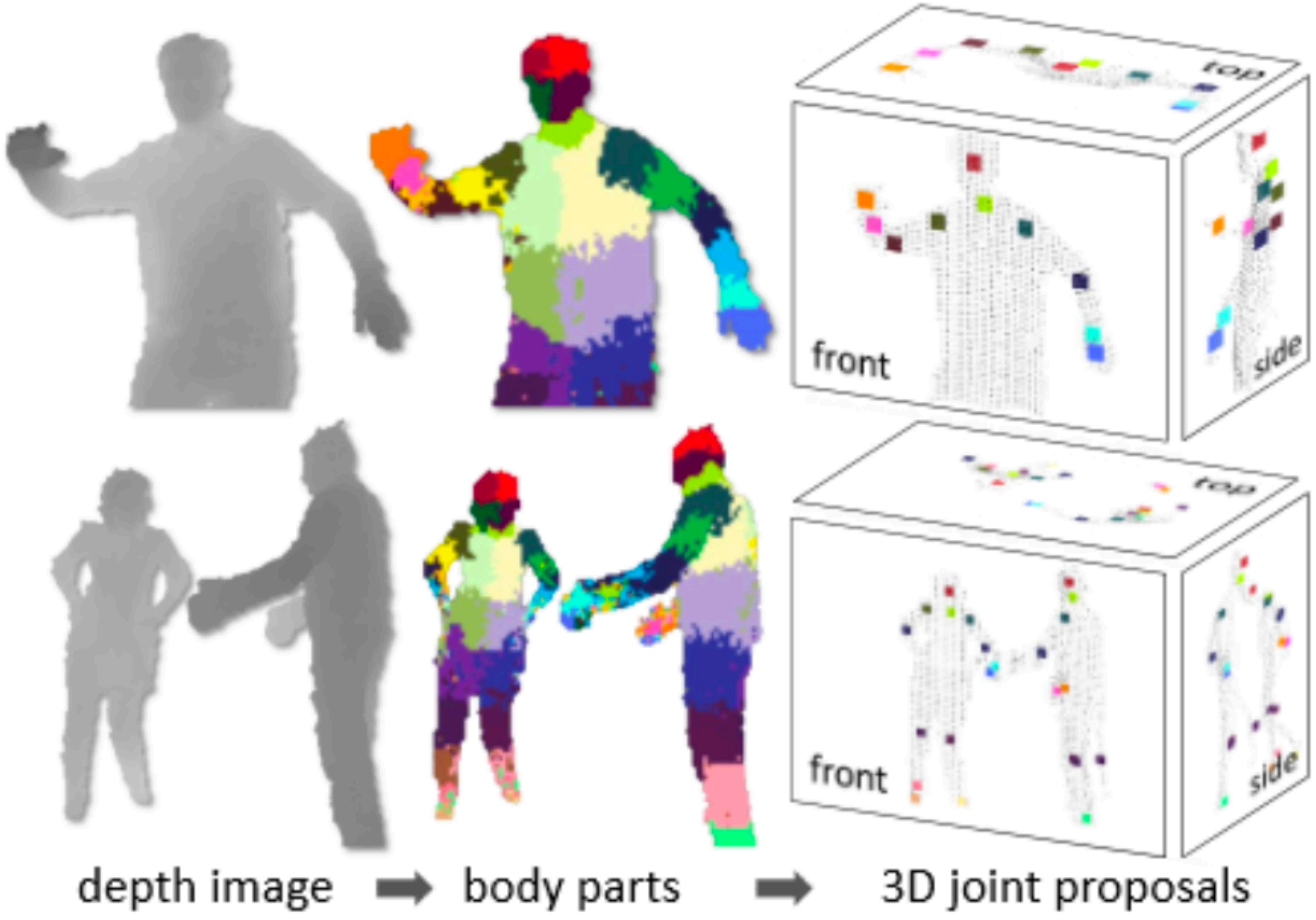
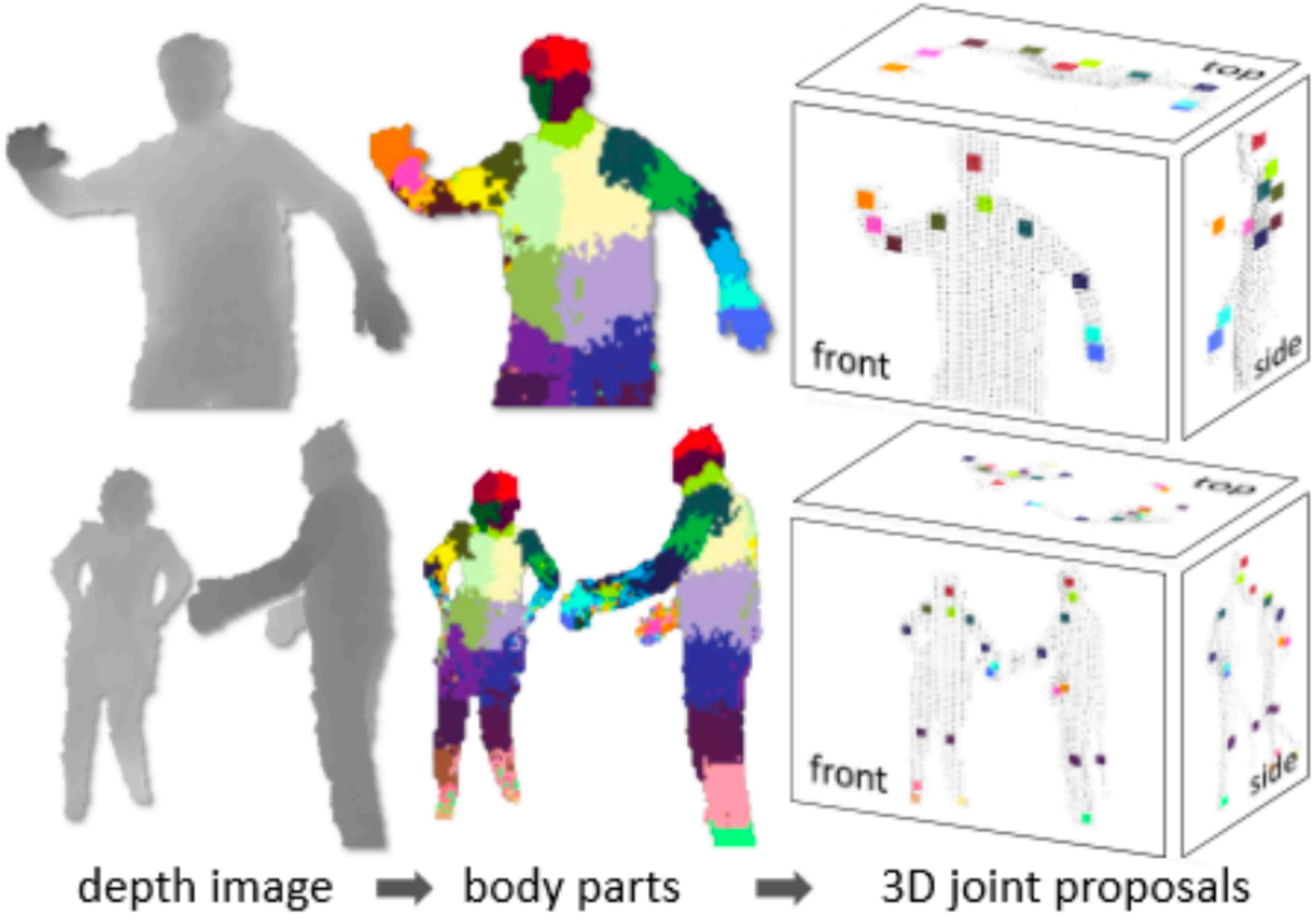


Figure credit: J. Shotton et al., 2011

Example 1: Kinect

Kinect allows users of Microsoft's Xbox 360 console to interact with games using natural body motions instead of a traditional handheld controller. The pose (joint positions) of the user is predicted using a random forest trained on depth features.

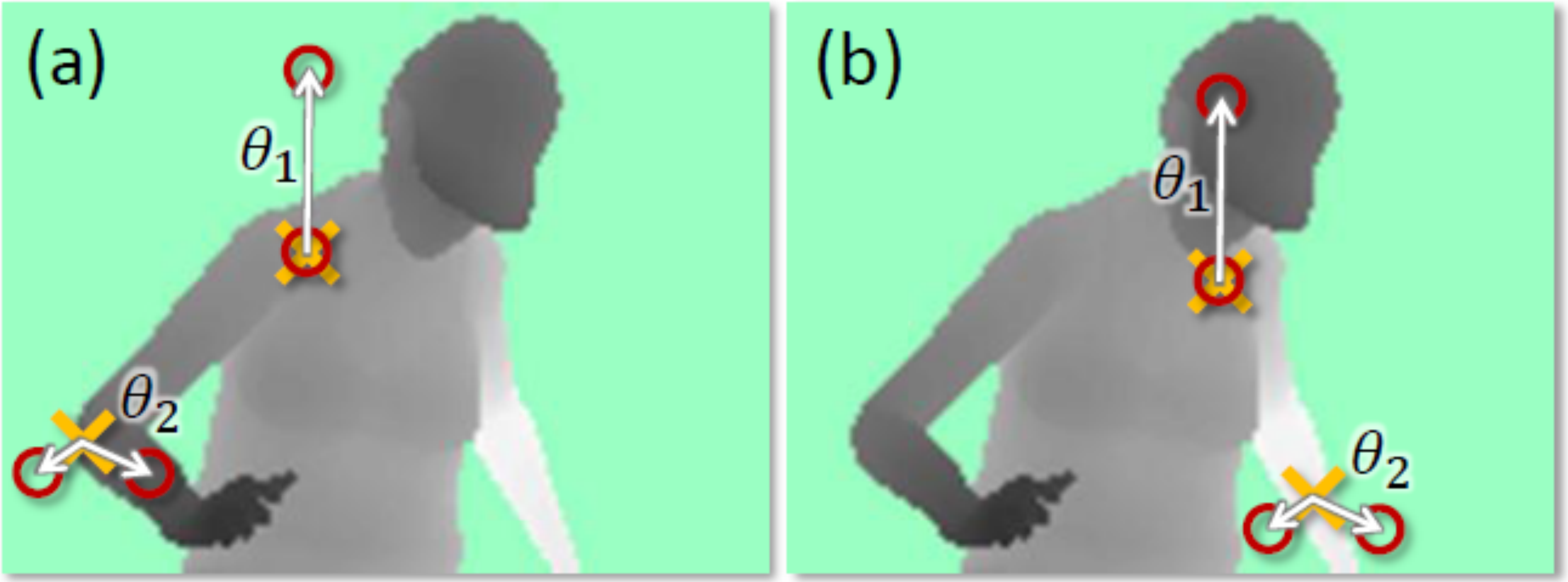


Jamie Shotton

Figure credit: J. Shotton et al., 2011

Example 1: Kinect

Simple test: threshold on the difference of two depth values at an offset from a target pixel …



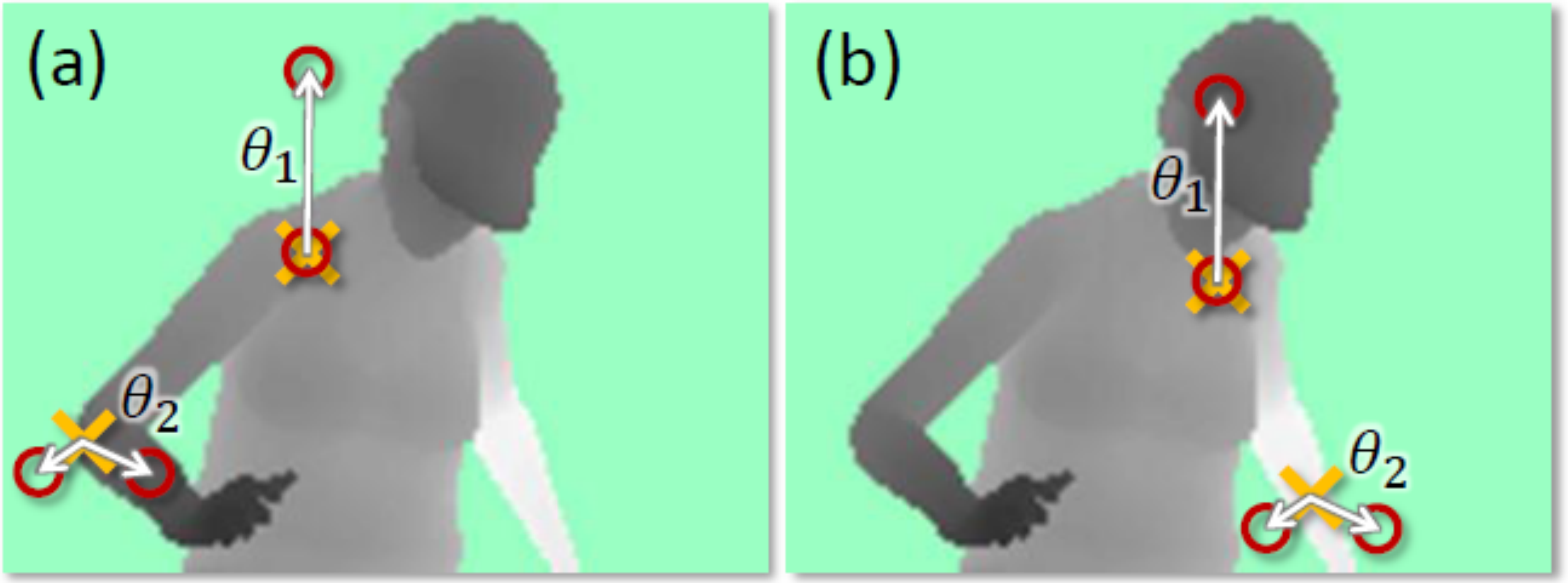
$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Figure credit: J. Shotton et al., 2011

Example 1: Kinect

$$f_\theta(I, \mathbf{x}) > \Theta_j$$

What are the parameters of this test?

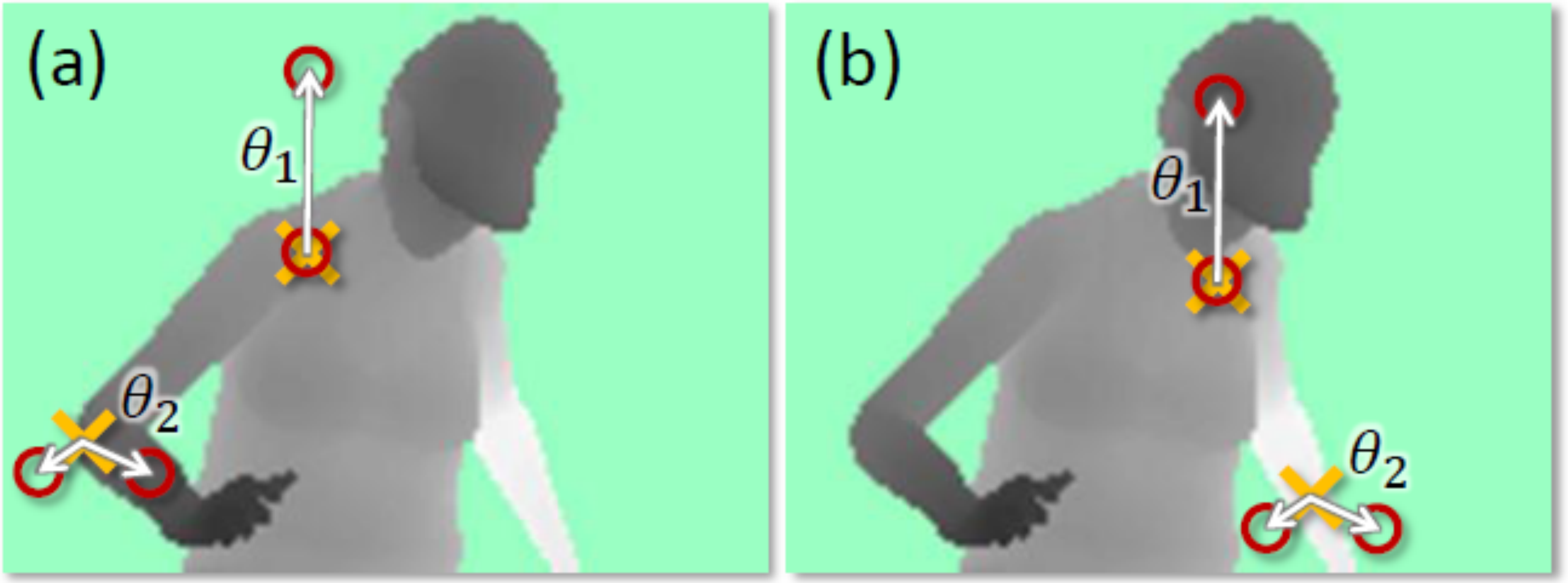


$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Example 1: Kinect

$$f_\theta(I, \mathbf{x}) > \Theta_j$$

What are the parameters of this test?



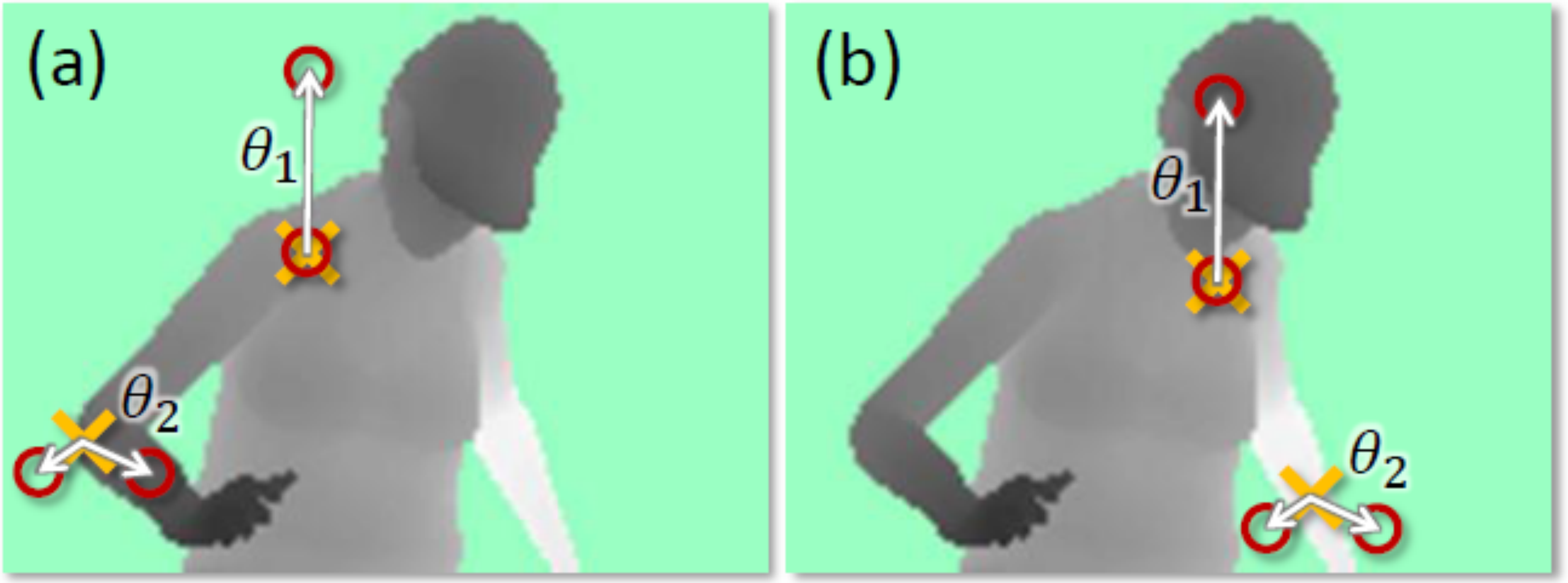
$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Example 1: Kinect

$$f_\theta(I, \mathbf{x}) > \Theta_j$$

What are the parameters of this test?

How many such tests can we have?



$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

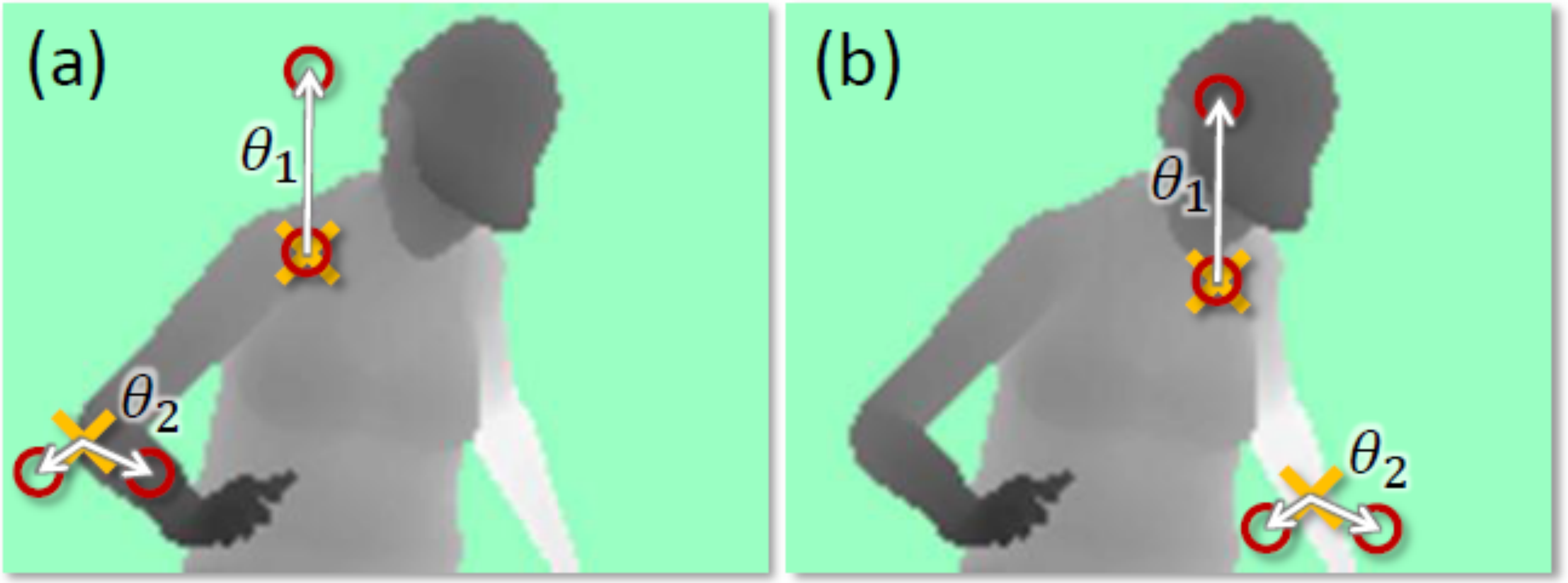
Example 1: Kinect

$$f_\theta(I, \mathbf{x}) > \Theta_j$$

What are the parameters of this test?

How many such tests can we have?

(# pix) x (# pix) x (# threshold)



$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

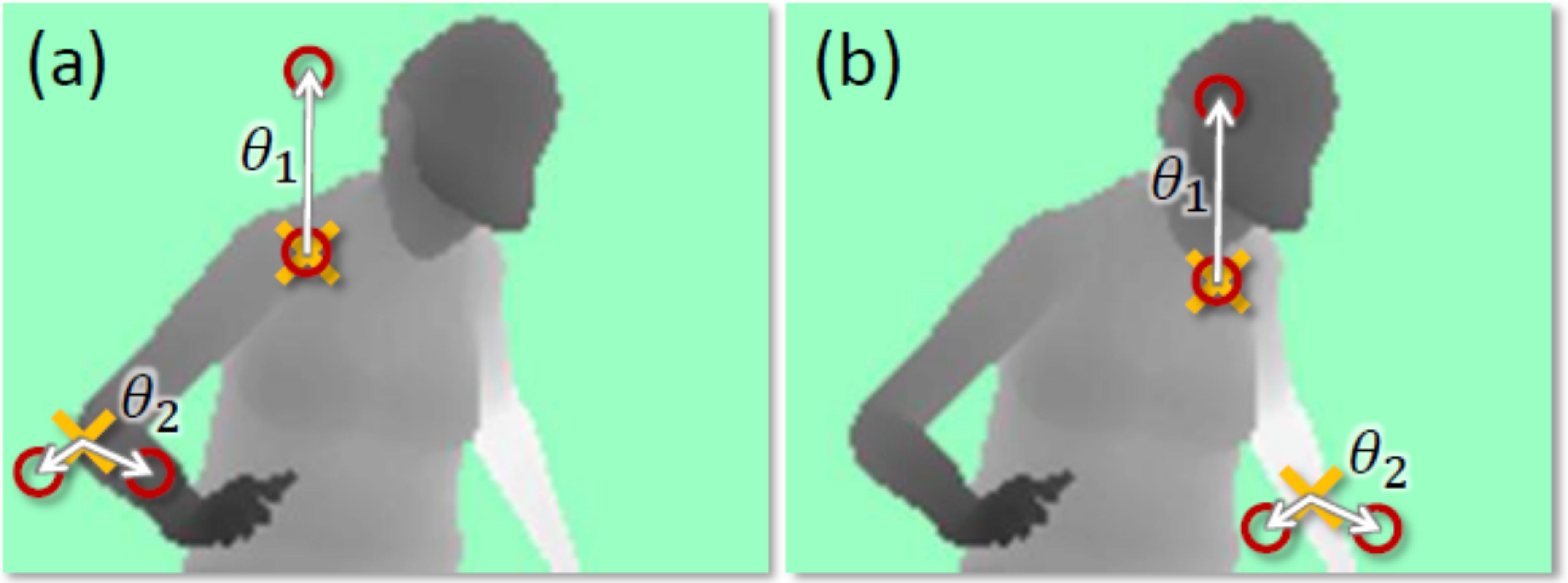
Example 1: Kinect

$$f_\theta(I, \mathbf{x}) > \Theta_j$$

What are the parameters of this test?

How many such tests can we have?

(# pix) x (# pix) x (# threshold)



Learning is slow (weeks)!

$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

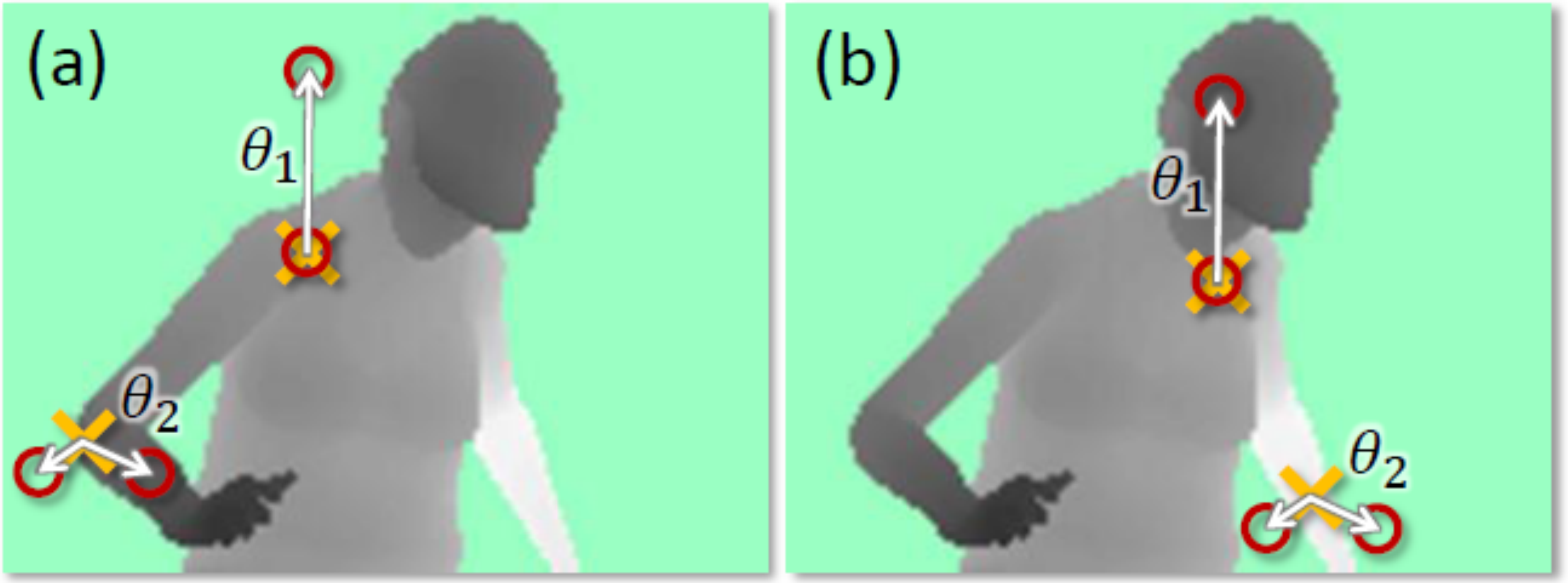
Example 1: Kinect

$$f_\theta(I, \mathbf{x}) > \Theta_j$$

What are the parameters of this test?

How many such tests can we have?

(# pix) x (# pix) x (# threshold)



$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

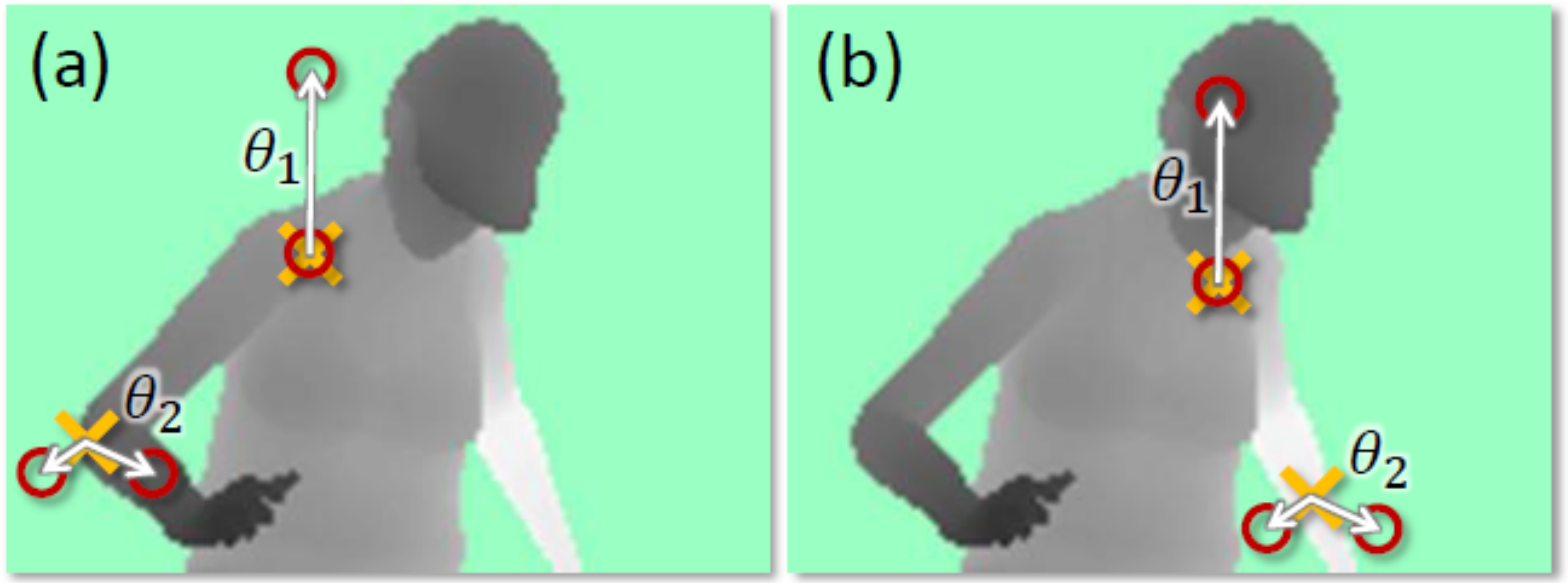
Learning is slow (weeks)!

Inference is fast (real-time)!

Example 1: Kinect

$\mathbf{u}$	$\mathbf{u}$	Θ_j
$\swarrow$	$\longrightarrow$	0.5
$\swarrow$	$\longrightarrow$	0.4
$\swarrow$	$\longrightarrow$	-0.2
…	…	…
$\swarrow$	$\uparrow$	0.7
$\leftarrow$	$\uparrow$	-0.7
$\longrightarrow$	$\nwarrow$	0.45
…	…	…

$$f_\theta(I, \mathbf{x}) > \Theta_j$$

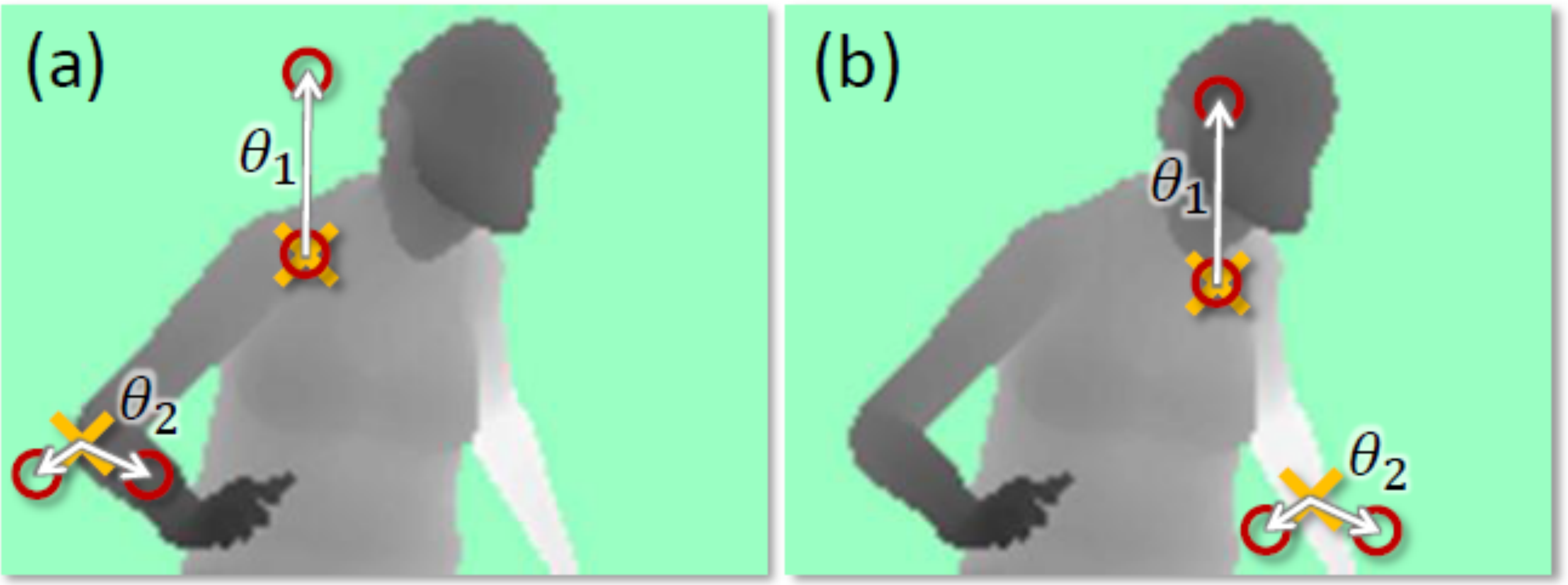


$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Example 1: Kinect

$\mathbf{u}$	$\mathbf{u}$	Θ_j	**information gain**
$\swarrow$	$\longrightarrow$	0.5	0.3
$\swarrow$	$\longrightarrow$	0.4	0.4
$\swarrow$	$\longrightarrow$	-0.2	0.7
…	…	…	…
$\swarrow$	$\uparrow$	0.7	0.2
$\leftarrow$	$\uparrow$	-0.7	0.8
$\longrightarrow$	$\nwarrow$	0.45	0.1
…	…	…	…

$$f_\theta(I, \mathbf{x}) > \Theta_j$$

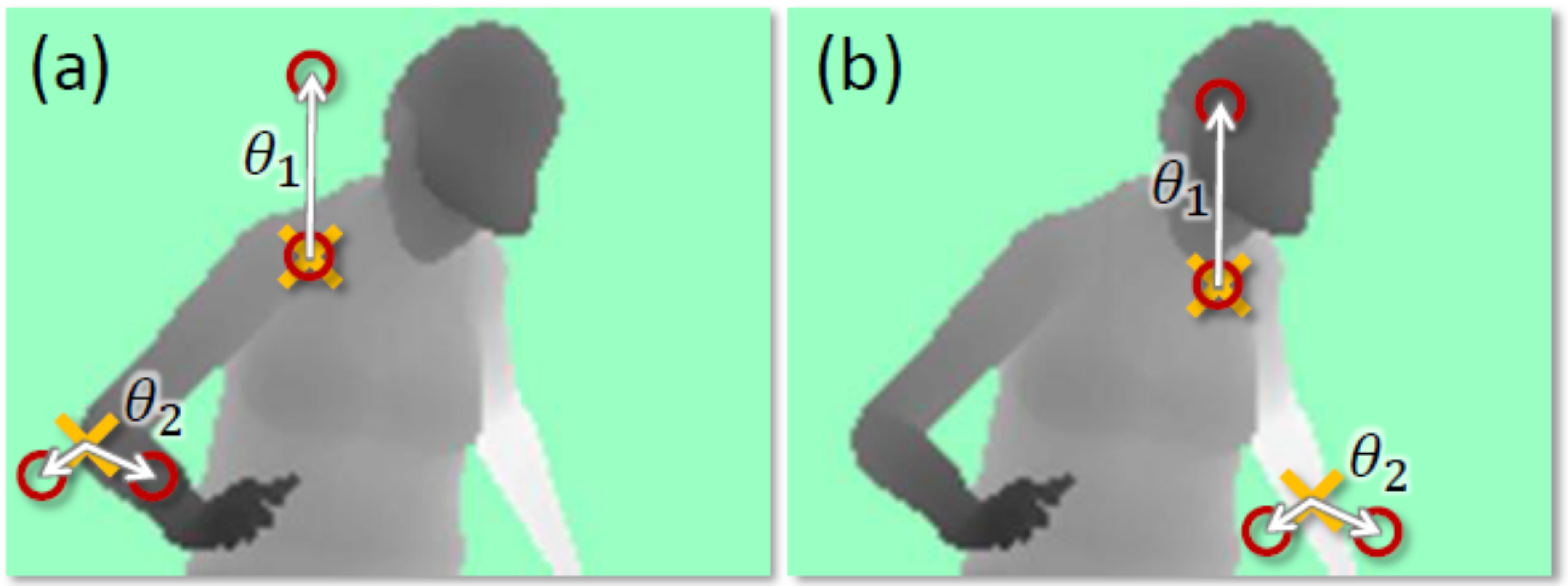


$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Example 1: Kinect

$\mathbf{u}$	$\mathbf{u}$	Θ_j	**information gain**
↙	→	0.5	0.3
↙	→	0.4	0.4
↙	→	-0.2	0.7
…	…	…	…
↙	↑	0.7	0.2
←	**↑**	**-0.7**	**0.8**
→	↖	0.45	0.1
…	…	…	…

$$f_\theta(I, \mathbf{x}) > \Theta_j$$



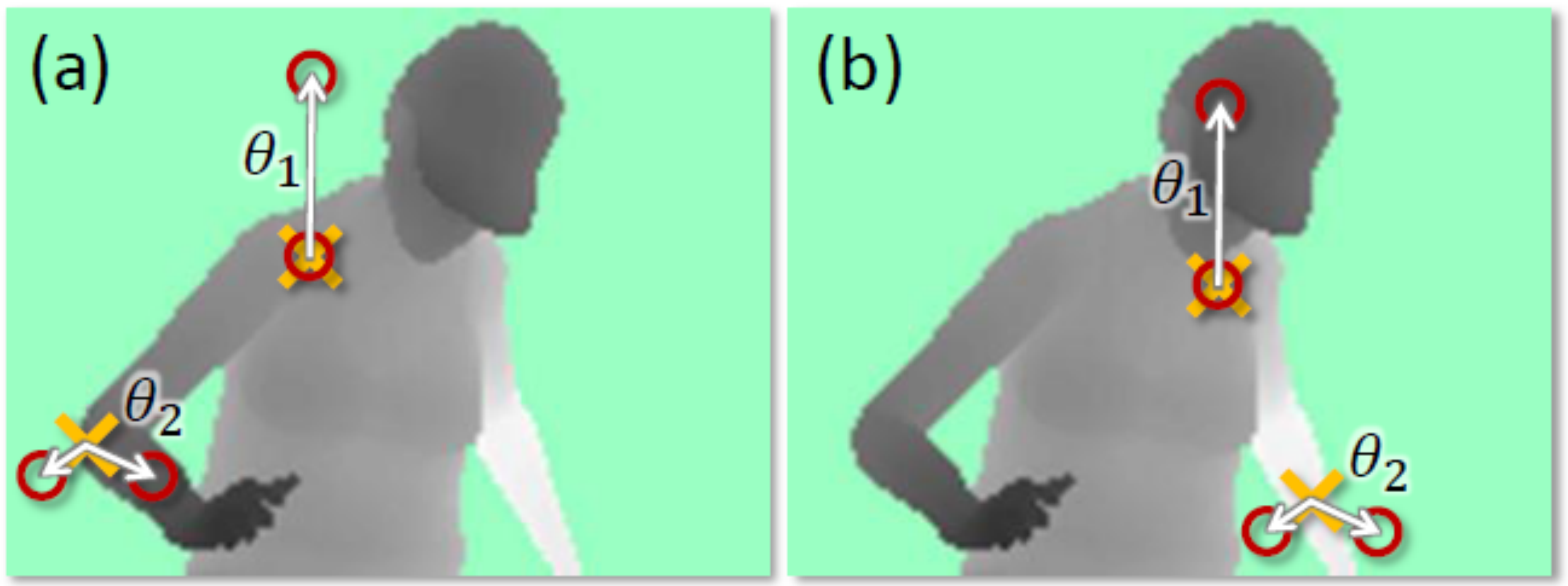
$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Example 1: Kinect

$\mathbf{u}$	$\mathbf{u}$	Θ_j	**information gain**
$\swarrow$	$\longrightarrow$	0.5	0.3
$\swarrow$	$\longrightarrow$	0.4	0.4
$\swarrow$	$\longrightarrow$	-0.2	0.7
...	...	...	...
$\swarrow$	$\uparrow$	0.7	0.2
$\leftarrow$	$\uparrow$	**-0.7**	**0.8**
$\longrightarrow$	$\nwarrow$	0.45	0.1
...	...	...	...

$d_I(\mathbf{x} + \longleftarrow) - d_I(\mathbf{x} + \uparrow\ ) > -0.7$

$d_I(\mathbf{x} + \longleftarrow) - d_I(\mathbf{x} + \uparrow\ ) < -0.7$



$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Example 1: Kinect

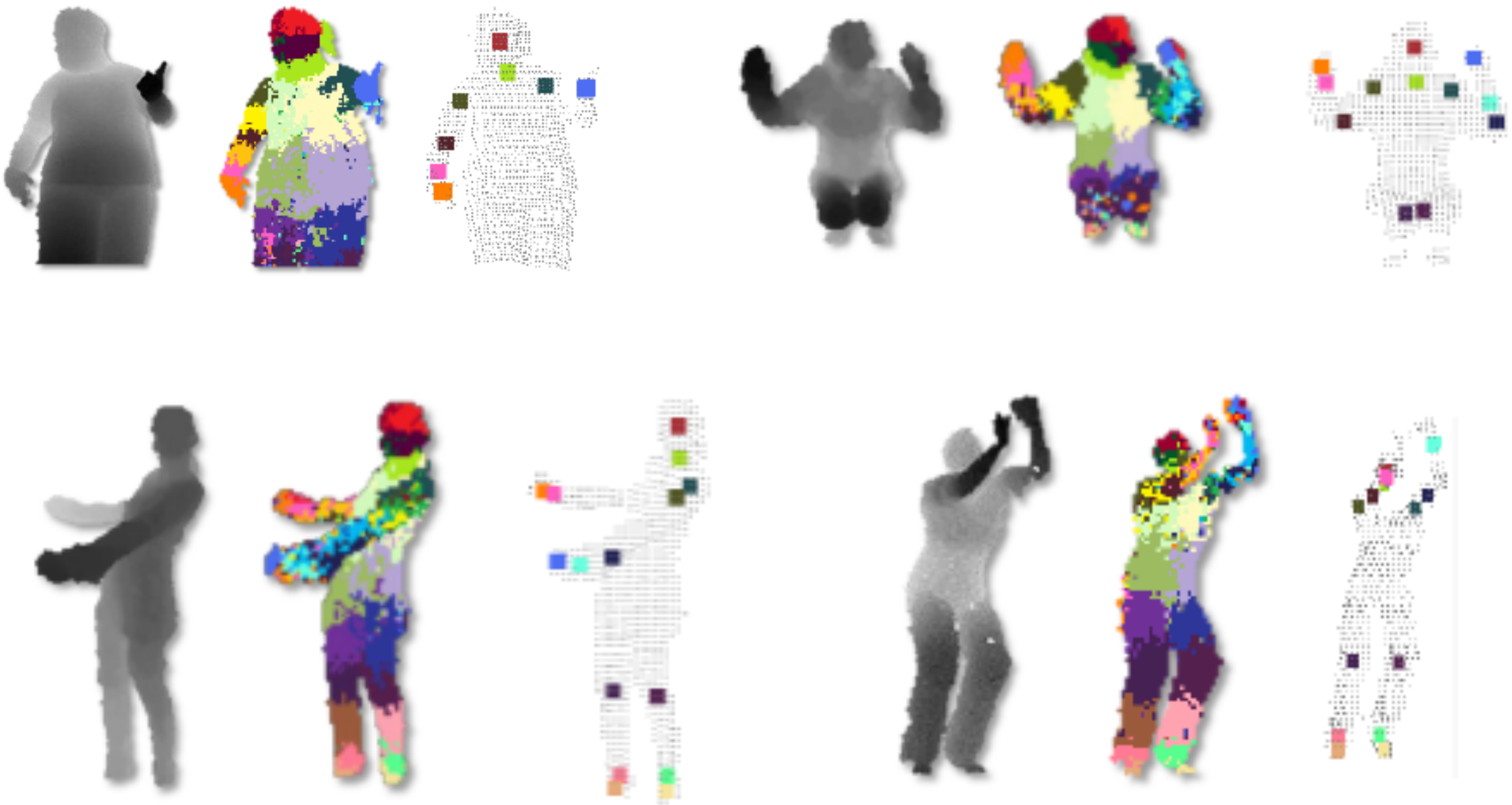


Figure credit: J. Shotton et al., 2011

Combining **Classifiers**

One common strategy to obtain a better classifier is to combine multiple classifiers.

A simple approach is to train an ensemble of independent classifiers, and average their predictions.

Boosting is another approach.

— Train an ensemble of classifiers sequentially.

— Bias subsequent classifiers to correctly predict training examples that previous classifiers got wrong.

— The final boosted classifier is a weighted combination of the individual classifiers.

Combining Classifiers: **Boosting**

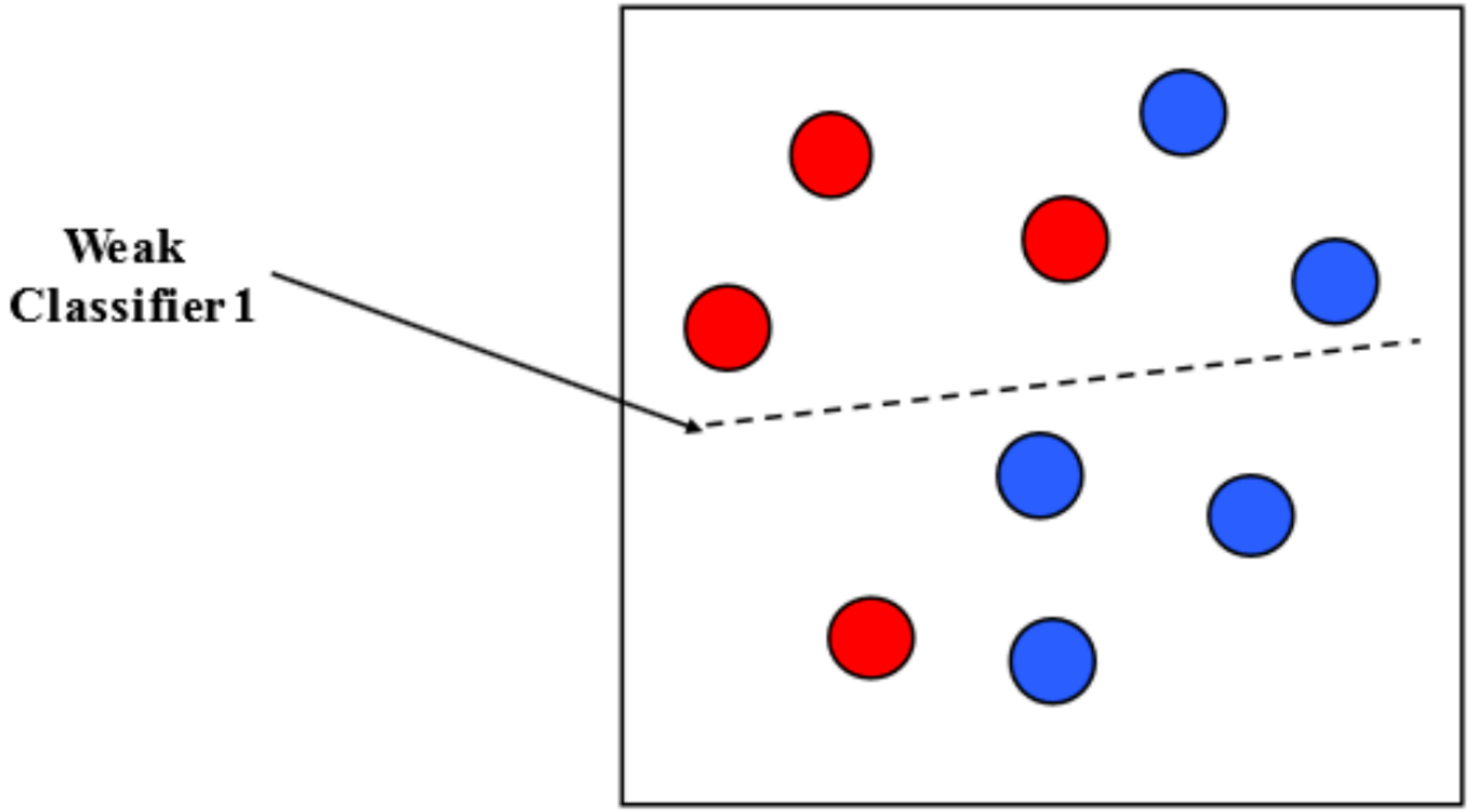


Figure credit: Paul Viola

Combining Classifiers: **Boosting**

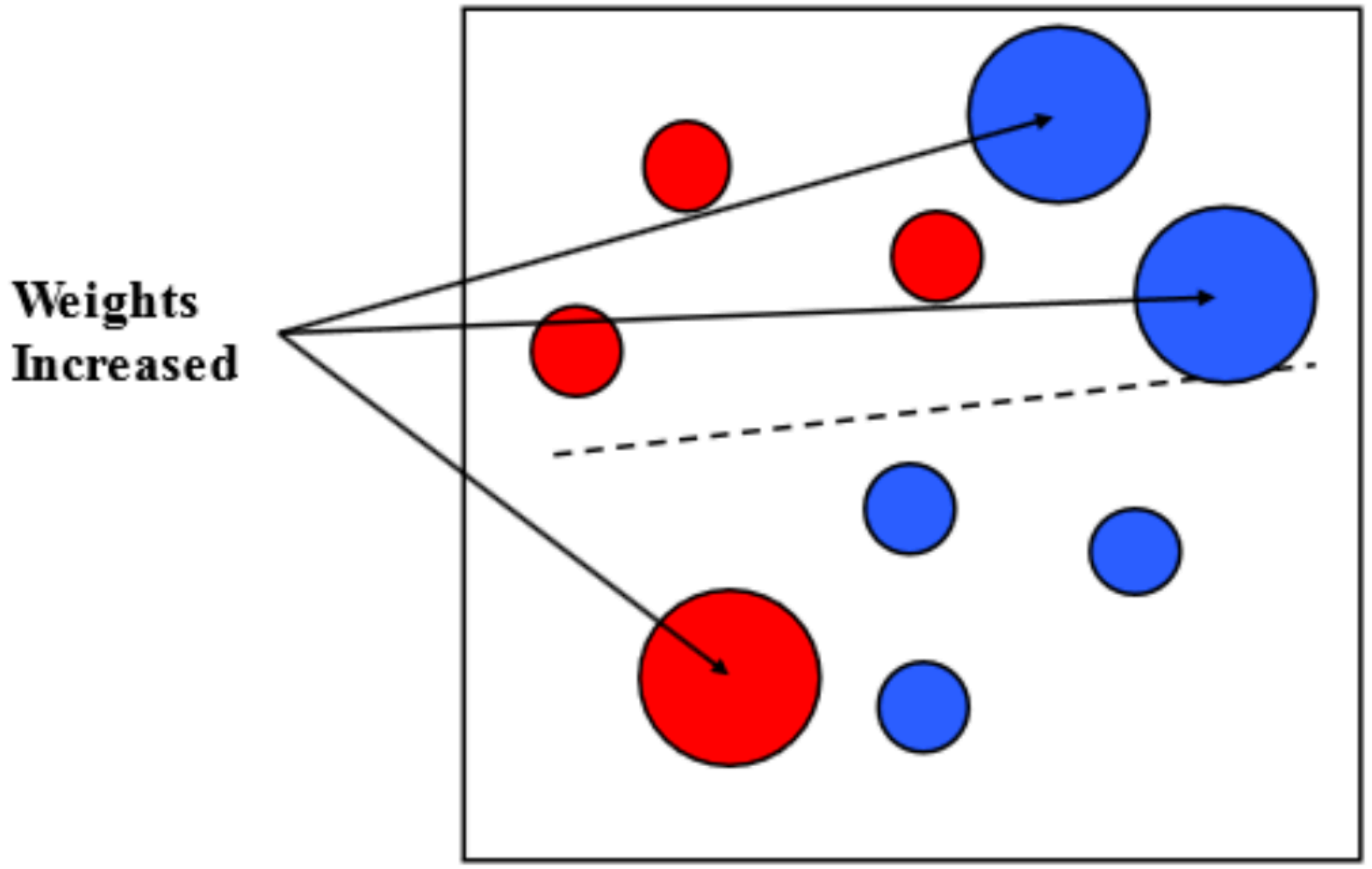


Figure credit: Paul Viola

Combining Classifiers: **Boosting**

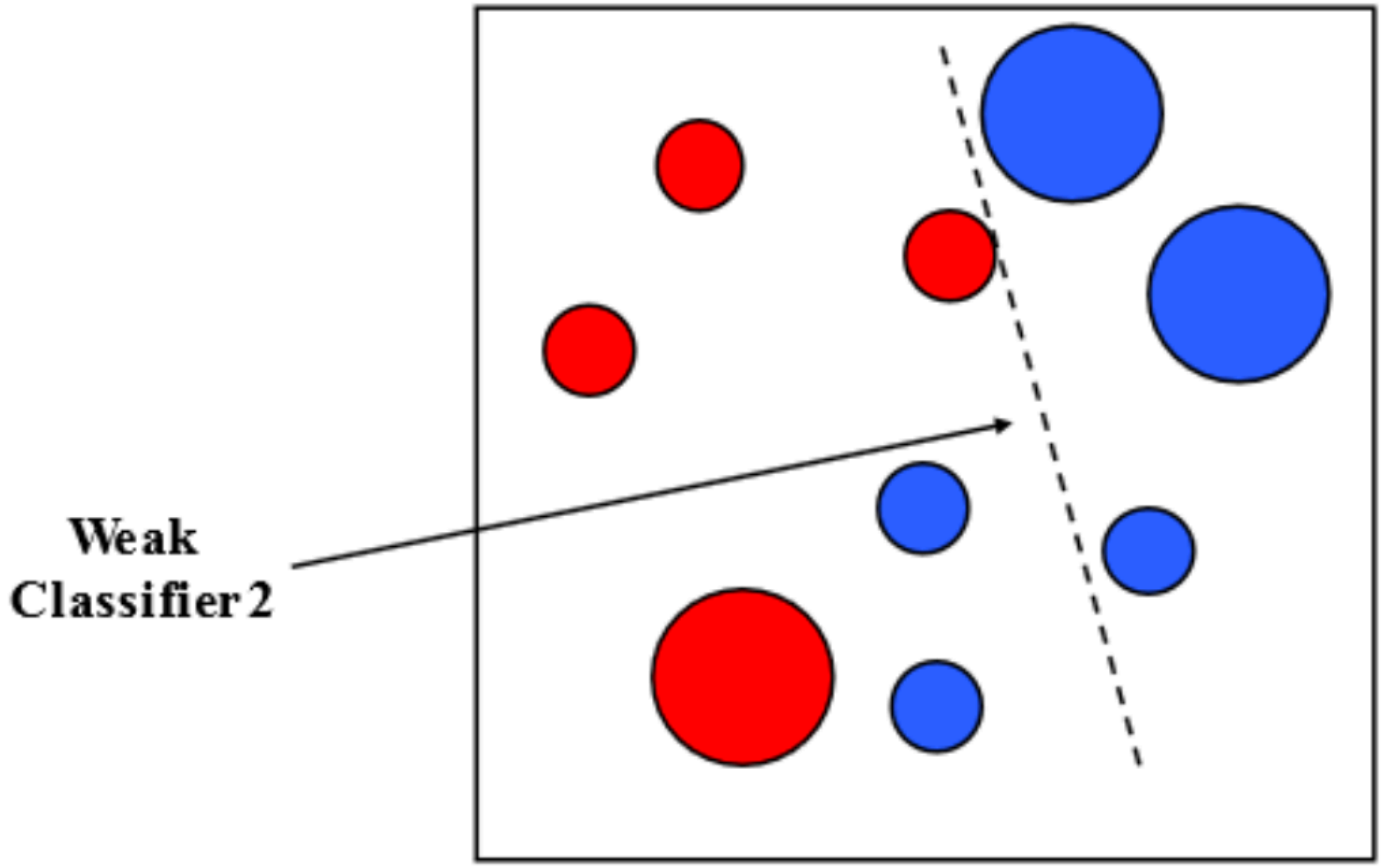


Figure credit: Paul Viola

Combining Classifiers: **Boosting**

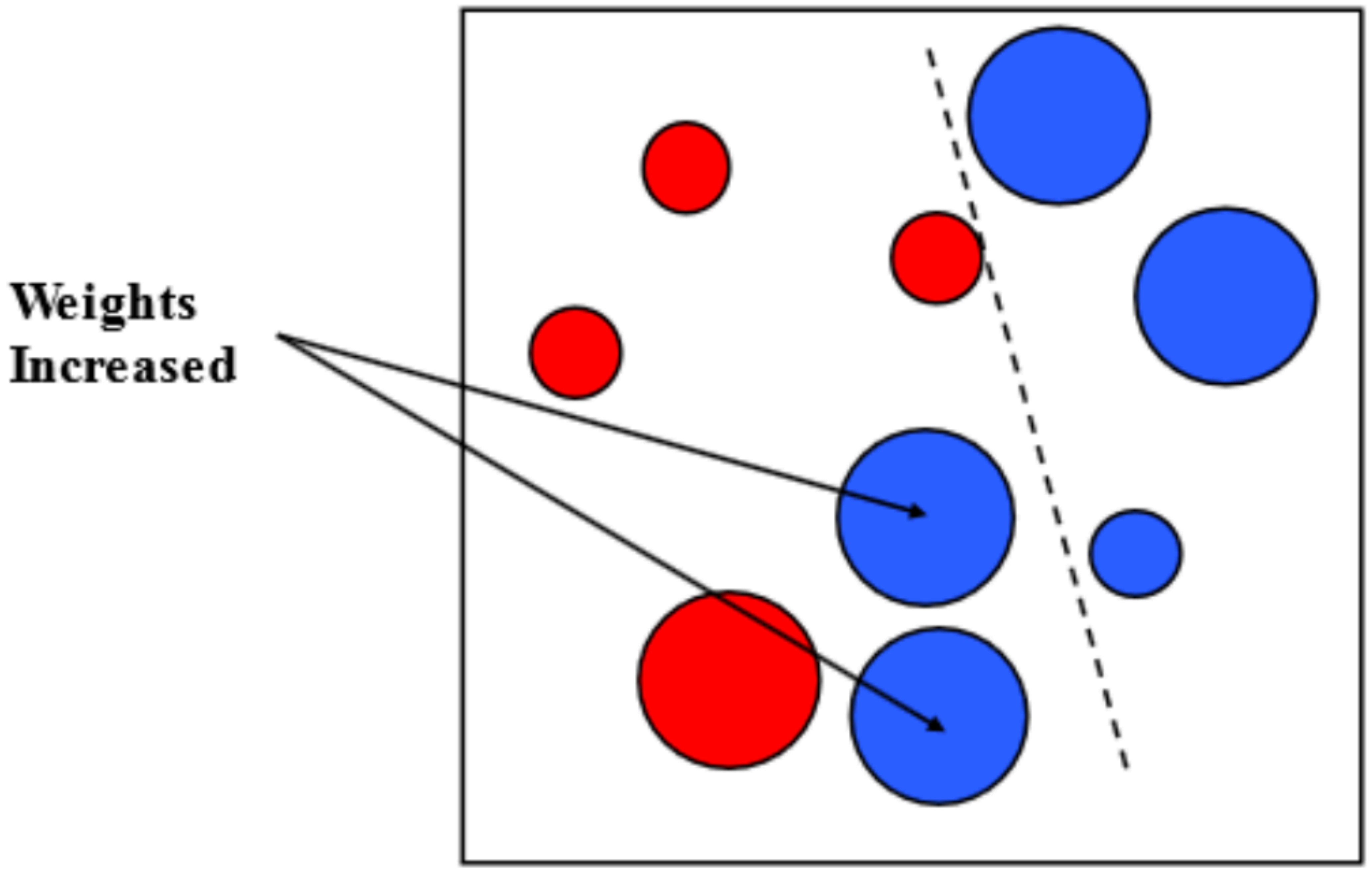


Figure credit: Paul Viola

Combining Classifiers: **Boosting**

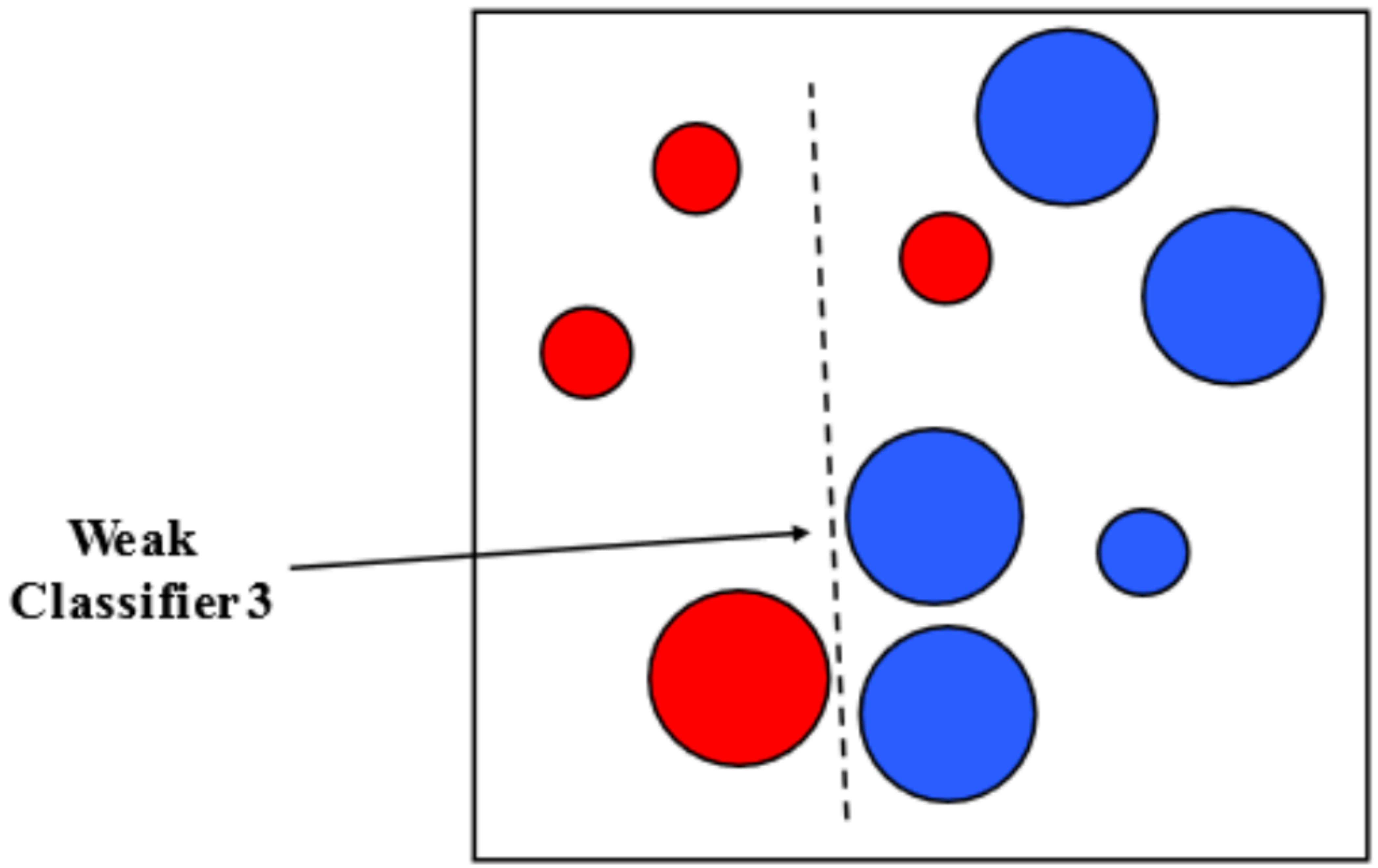


Figure credit: Paul Viola

Combining Classifiers: **Boosting**

Final classifier is a combination of weak classifiers

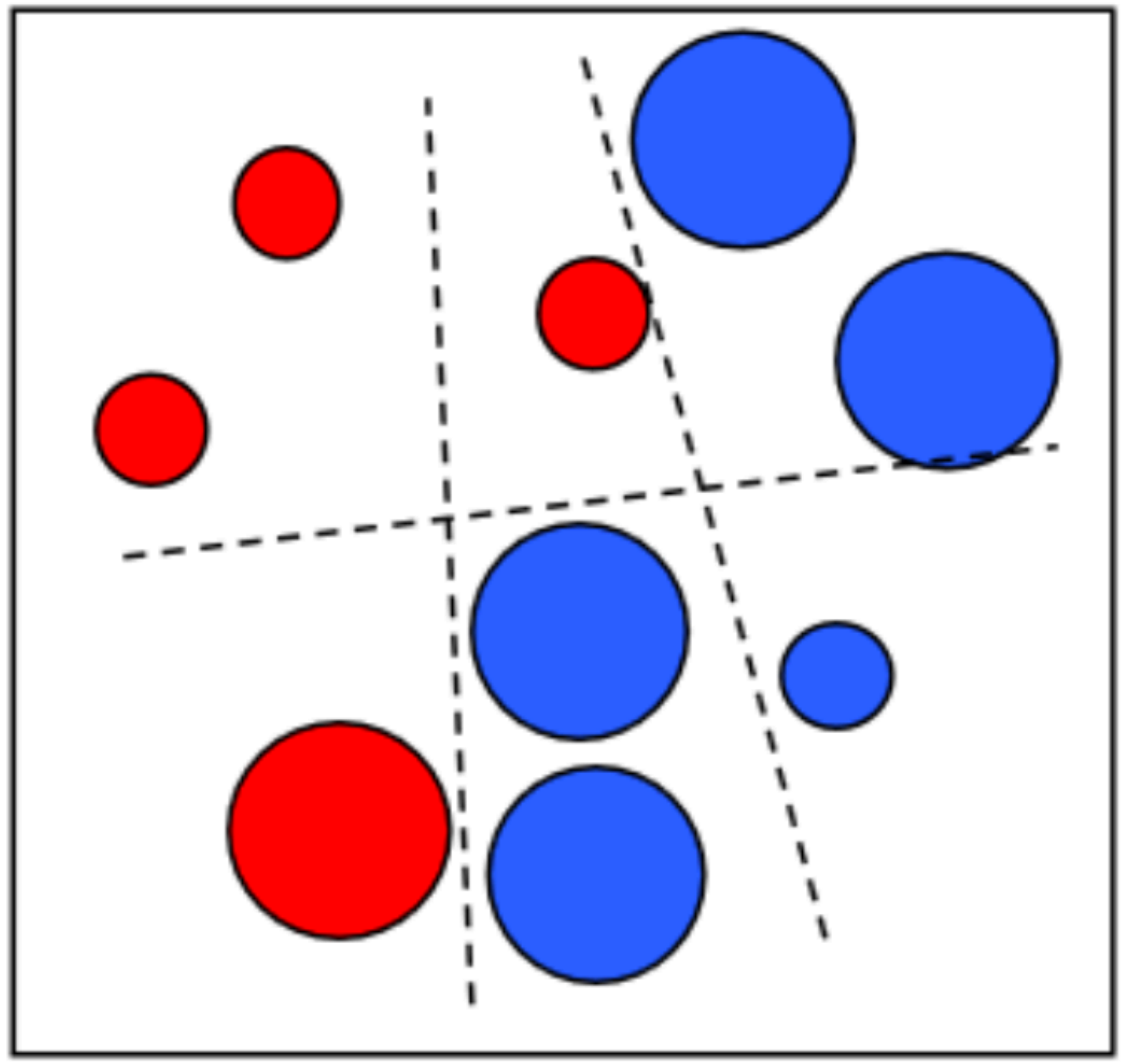


Figure credit: Paul Viola

Object Detection: Introduction

We have been discussing **image classification**, where we pass a whole image into a classifier and obtain a class label as output

We assumed the image contained a single, central object

The task of **object detection** is to detect and localize all instances of a target object class in an image
— Localization typically means putting a tight bounding box around the object

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

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.

Is there a car?

Image credit: KITTI Vision Benchmark

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.

Is there a car?

Image credit: KITTI Vision Benchmark

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.

Is there a car?

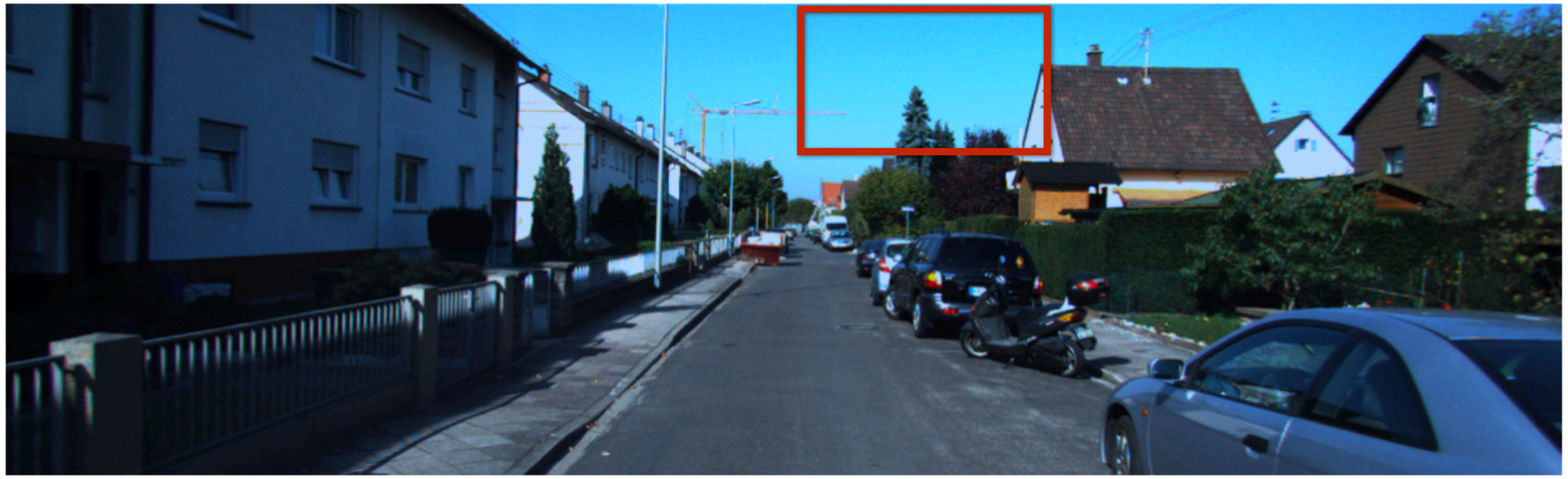


Image credit: KITTI Vision Benchmark

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.

Is there a car?

Image credit: KITTI Vision Benchmark

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

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

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.

Is there a car?

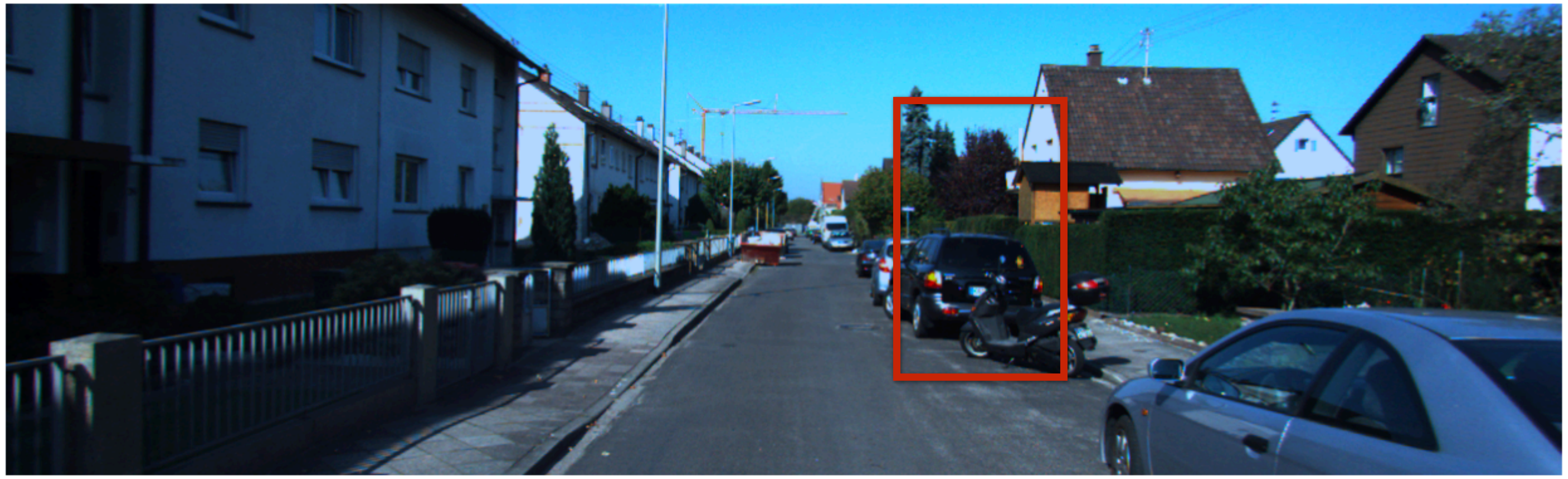


Image credit: KITTI Vision Benchmark

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.

Is there a car?

Image credit: KITTI Vision Benchmark

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

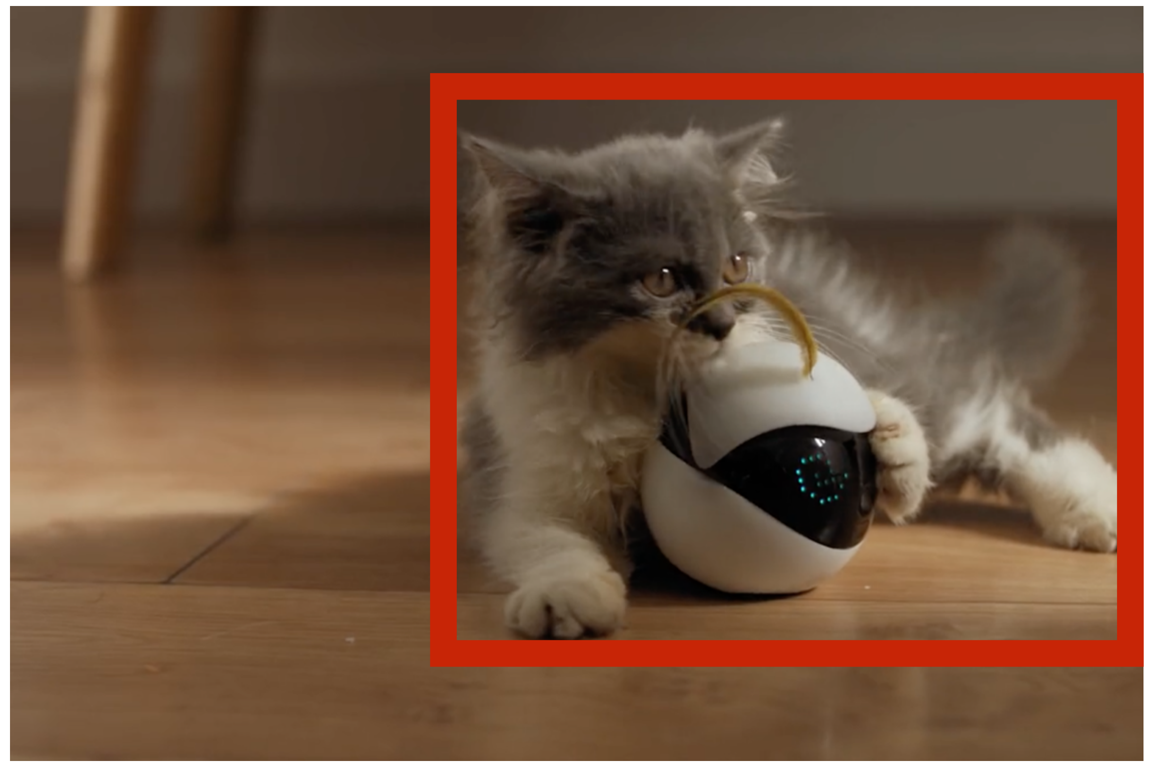
What data we **train** a classifier on?

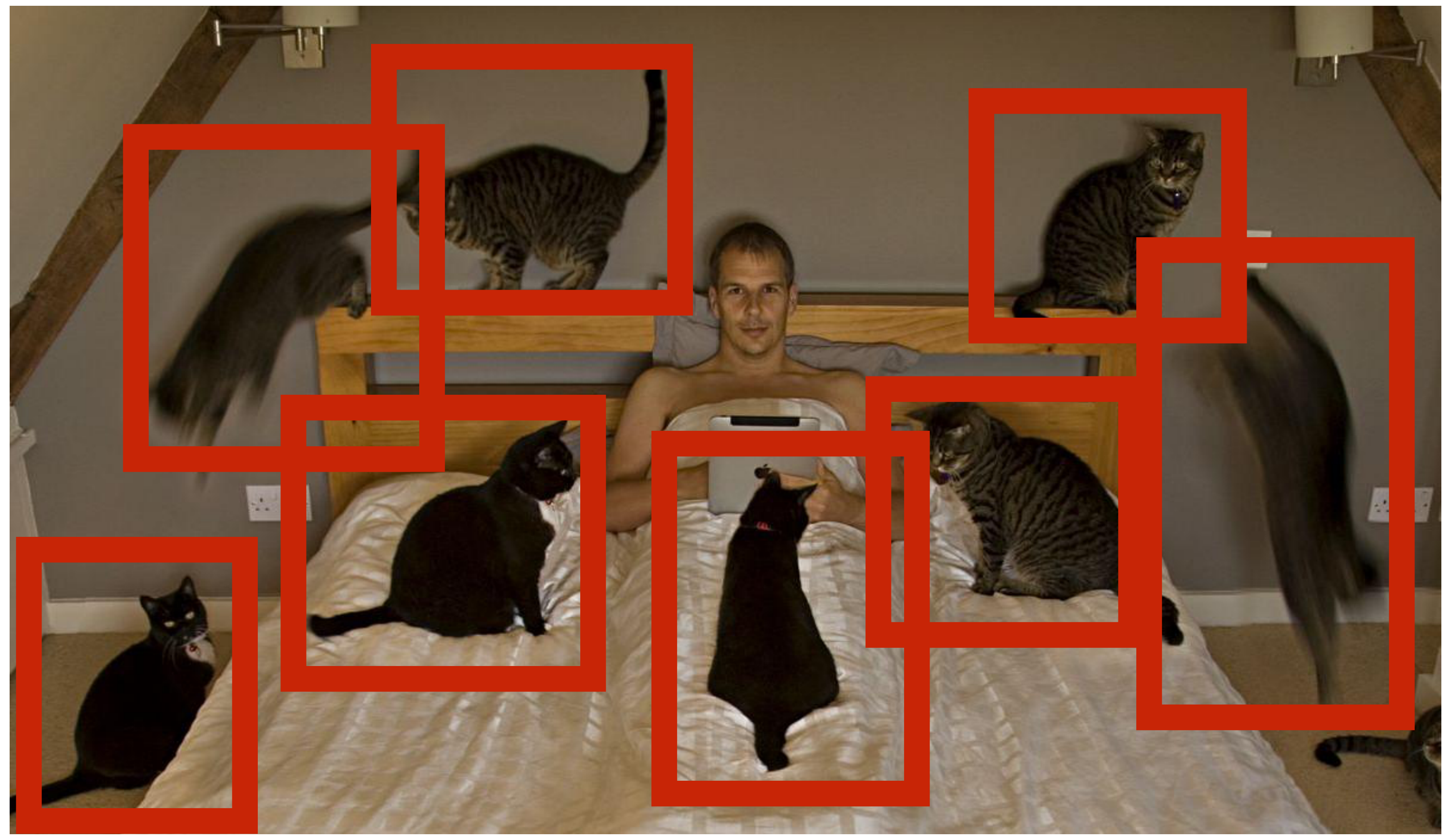
Image Classifiers

Image classifiers can be applied to regions/windows, but do not work so well in practice …

What data we **train** a classifier on?

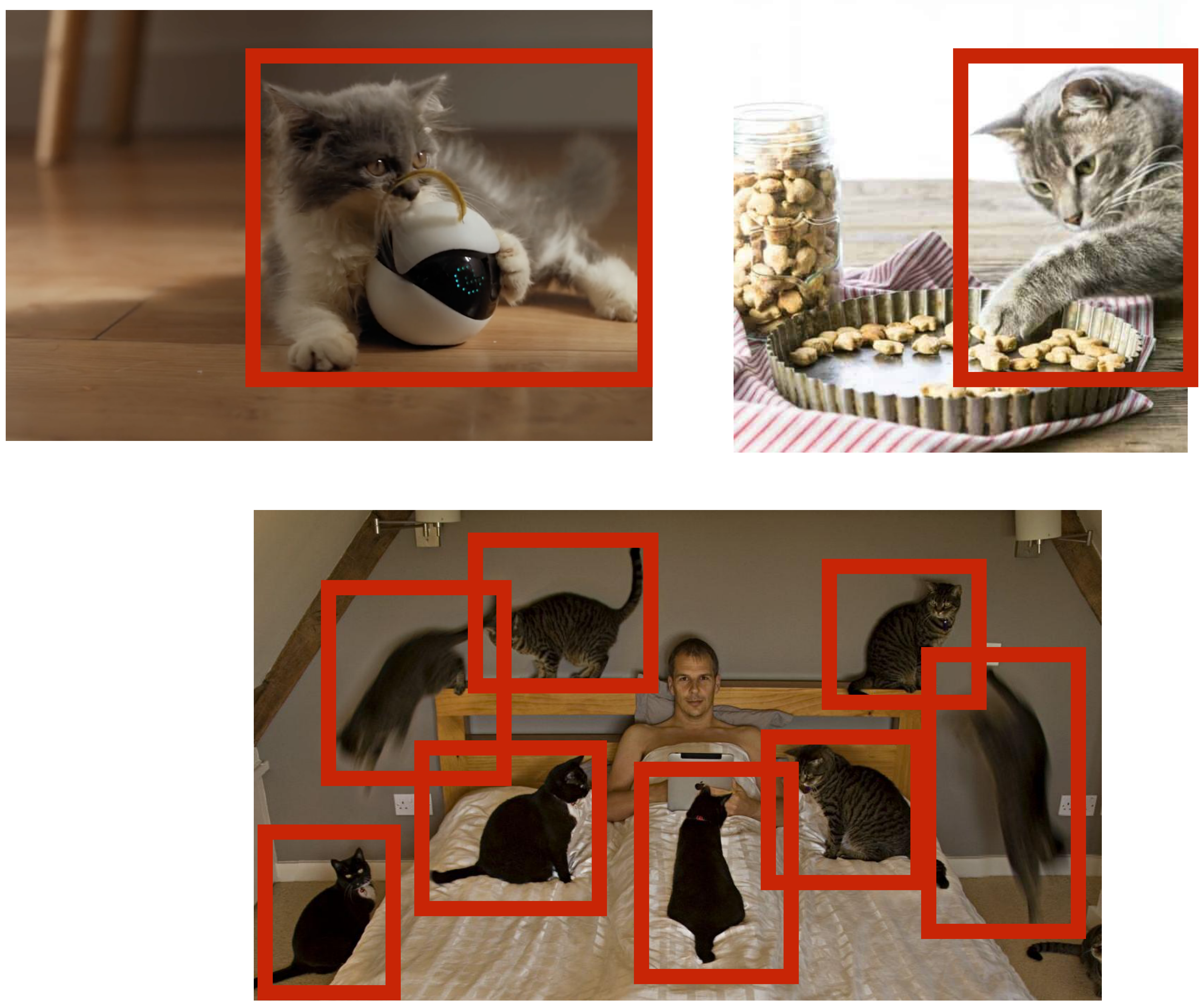
Image Classifiers





What data we **train** a classifier on?

Image Classifiers



Object Classifiers

Let’s assume we have **object** labeled data …

Object Classifiers

Object classifiers work a lot better … but require expensive bounding box annotations …

Let’s assume we have **object** labeled data …

Object Classifiers

(for convenience we will normalize patches to 64x64 … or 128x128)

Object classifiers work a lot better … but require expensive bounding box annotations …

Example: Face Detection

The **Viola-Jones** face detector is a classic sliding window detector that learns both efficient features and a classifier

A key strategy is to use features that are fast to evaluate to reject most windows early

The Viola-Jones detector computes ‘rectangular’ features within each window

Example: Face Detection Summary

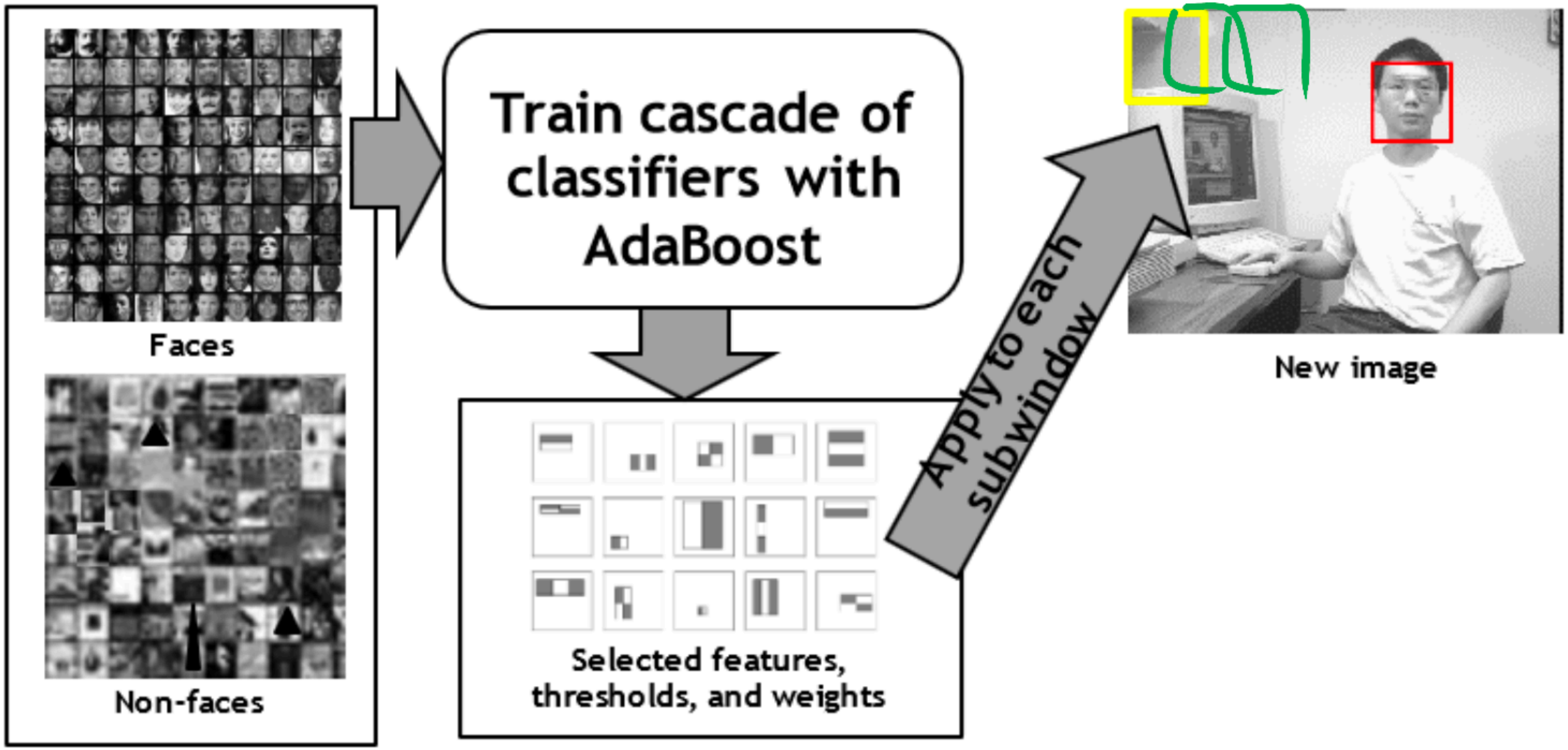
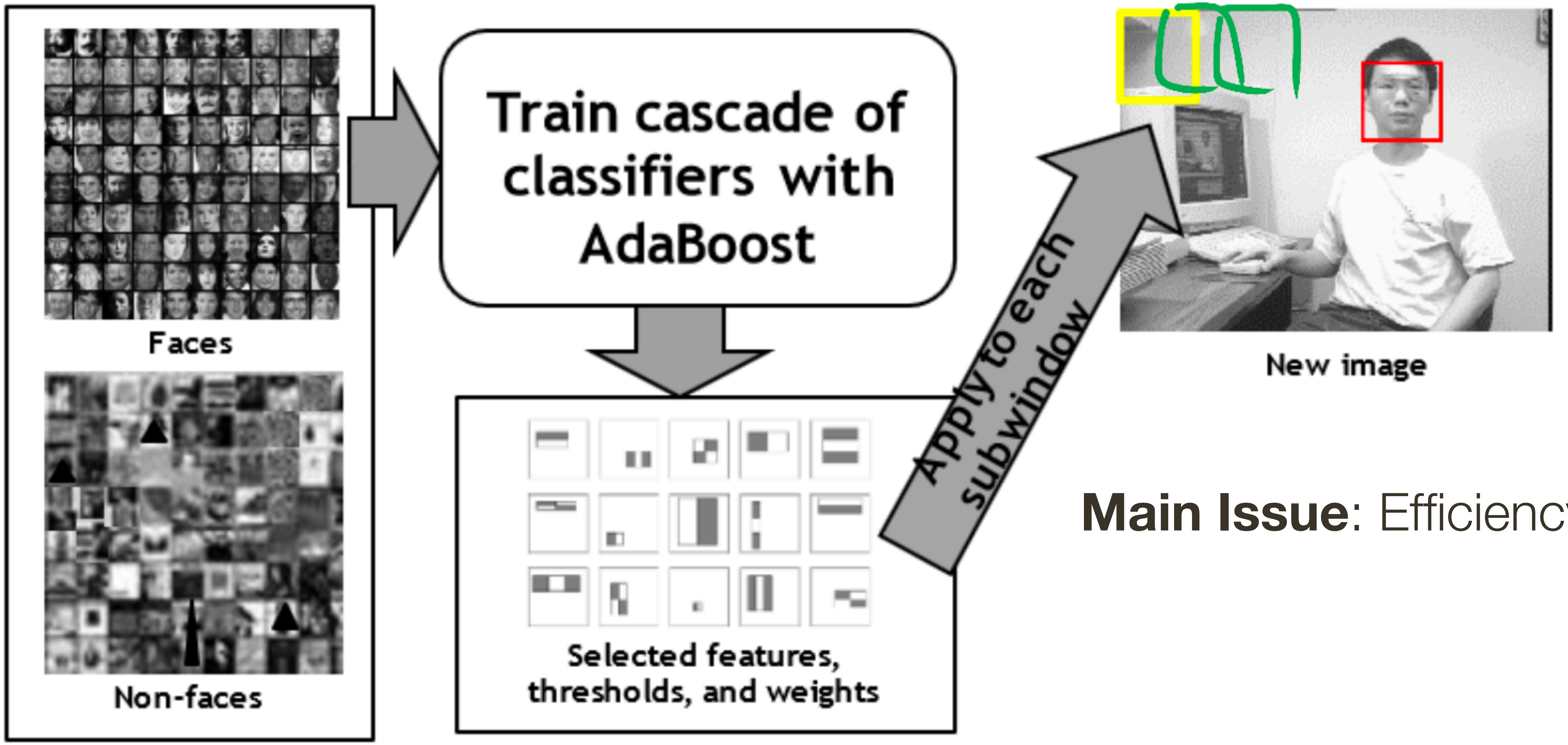


Figure credit: K. Grauman

Example: Face Detection Summary



Main Issue: Efficiency

Figure credit: K. Grauman

Example: Face Detection

Observations:

— On average only **0.01%** of all sub-windows are positive (faces)

— Equal computation time is spent on all sub-window

— Shouldn’t we spend most time only on **potentially positive** sub-windows?

Example: Face Detection

Observations:

— On average only **0.01%** of all sub-windows are positive (faces)

— Equal computation time is spent on all sub-window

— Shouldn't we spend most time only on **potentially positive** sub-windows?

A simple 2-feature classifier can achieve almost 100% detection rate (0% false negatives) with 50% false positive rate

Example: Face Detection

Observations:

— On average only **0.01%** of all sub-windows are positive (faces)

— Equal computation time is spent on all sub-window

— Shouldn't we spend most time only on **potentially positive** sub-windows?

A simple 2-feature classifier can achieve almost 100% detection rate (0% false negatives) with 50% false positive rate

Solution:

— A simple 2-feature classifier can act as a 1st layer of a series to filter out most negative (clearly non-face) windows

— 2nd layer with 10 features can tackle "harder" negative-windows which survived the 1st layer, and so on…

Cascading Classifiers

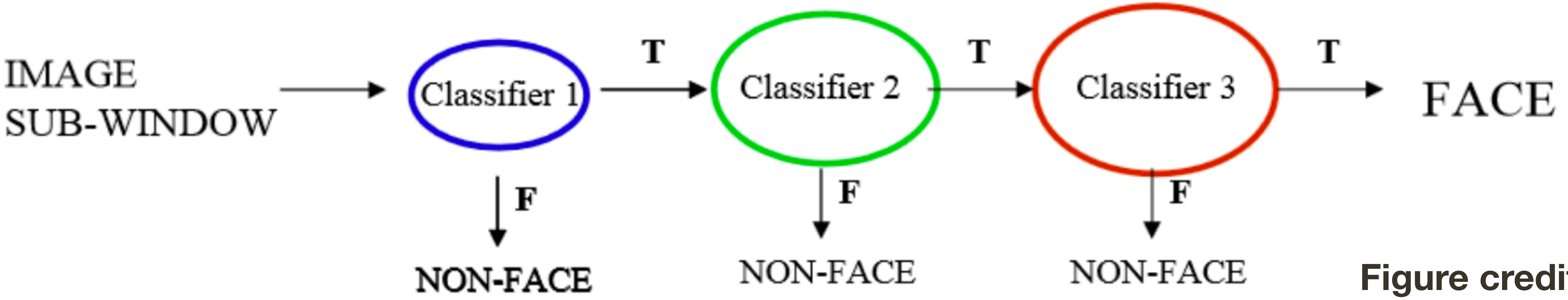


Figure credit: P. Viola

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

Cascading Classifiers

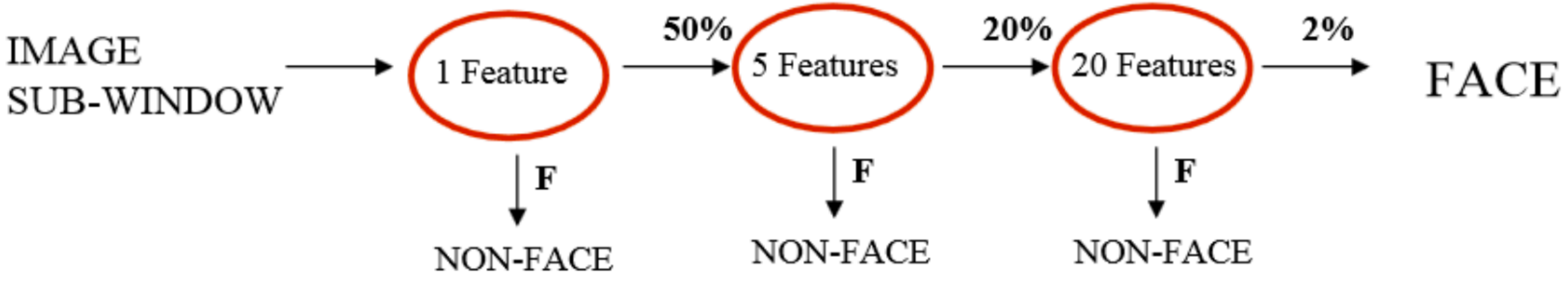


Figure credit: P. Viola

A **classifier** in the cascade is not necessarily restricted to a single feature

Example: Face Detection Summary

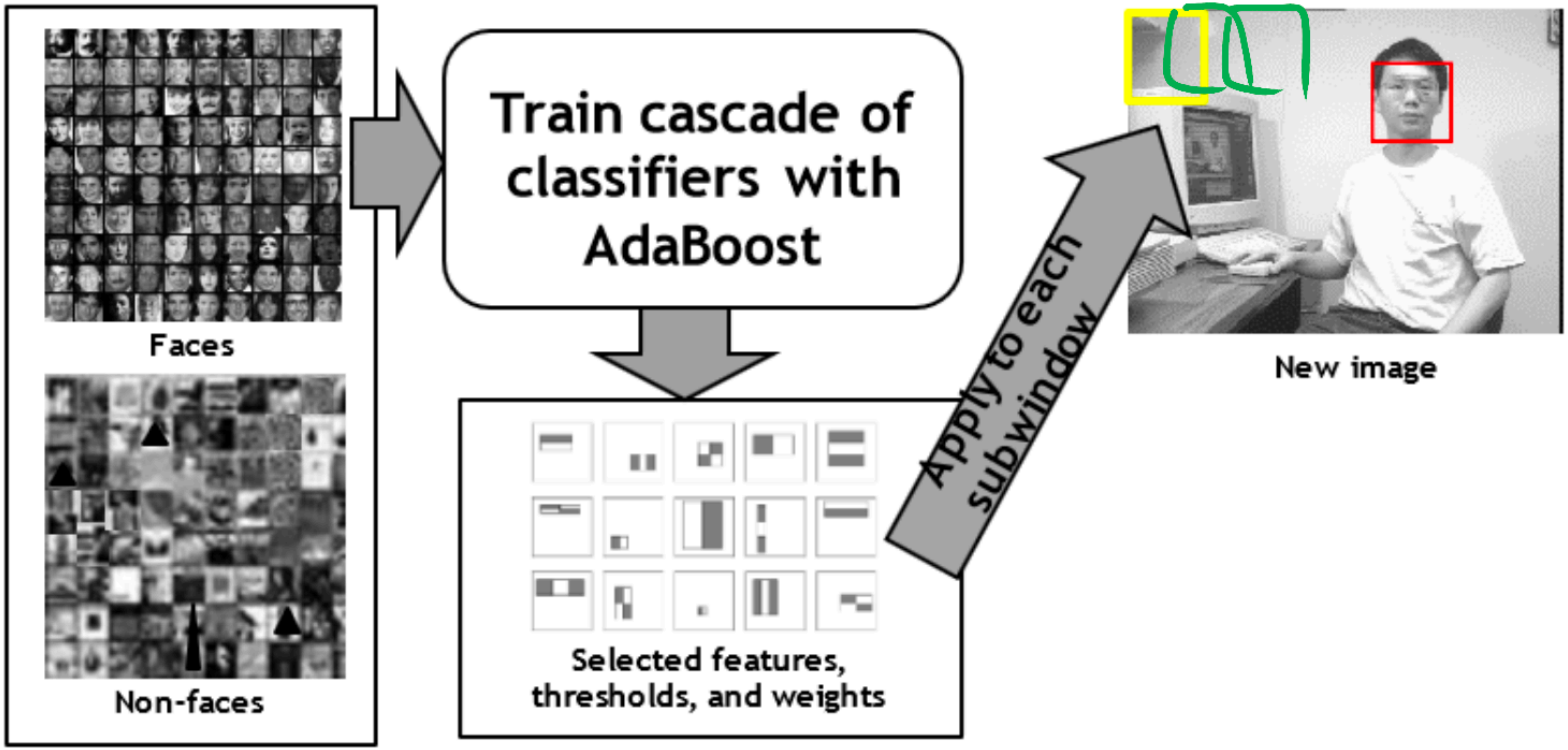


Figure credit: K. Grauman

Hard Negative Mining

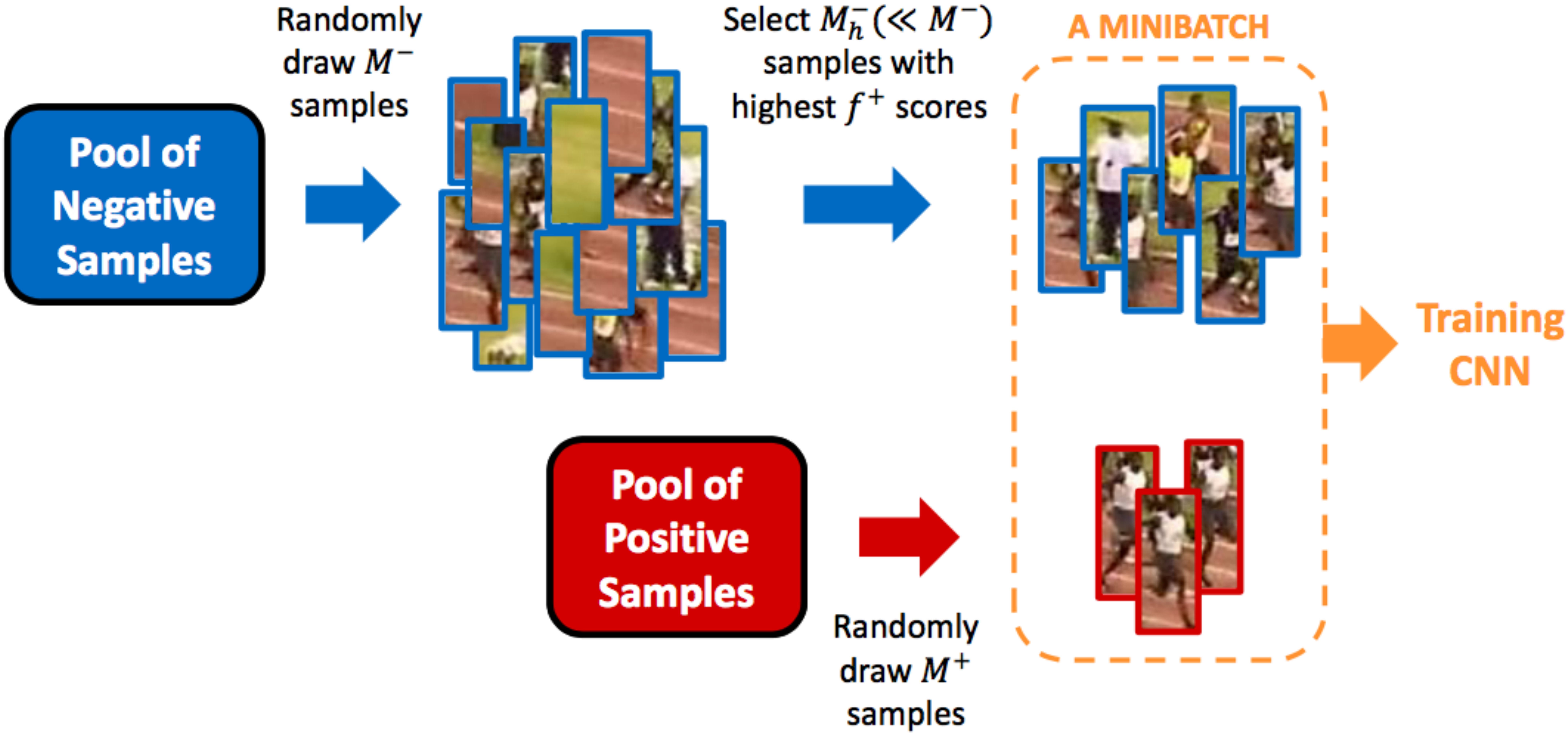


Image From: Jamie Kang

Recall: 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

Recall: 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 lot of possible windows! And most will not contain the object we are looking for.

Object Proposals

Object proposal algorithms generate a short list of regions that have generic object-like properties

— These regions are likely to contain some kind of foreground object instead of background texture

The object detector then considers these candidate regions only, instead of exhaustive sliding window search

Object Proposals

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

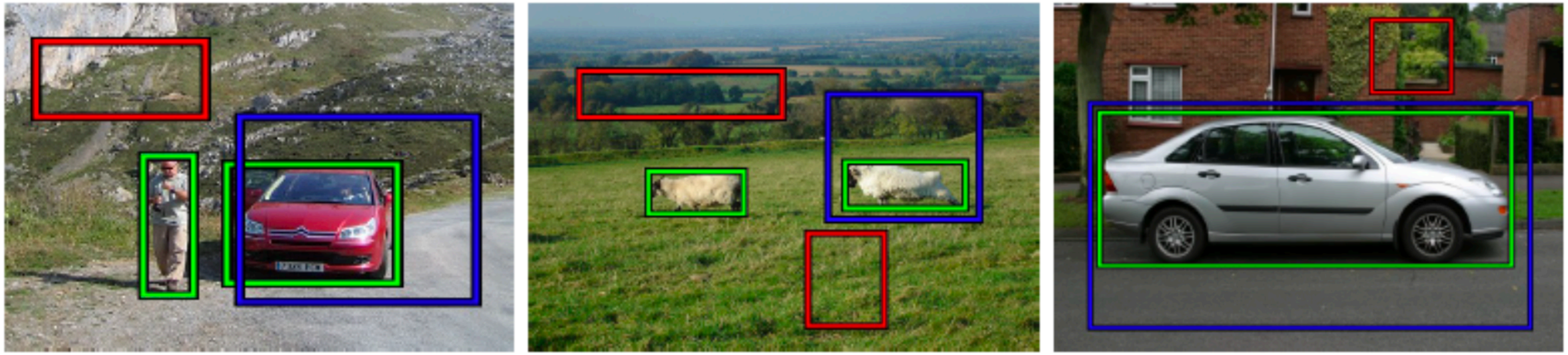


Figure credit: Alexe et al., 2012

Object Proposals

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

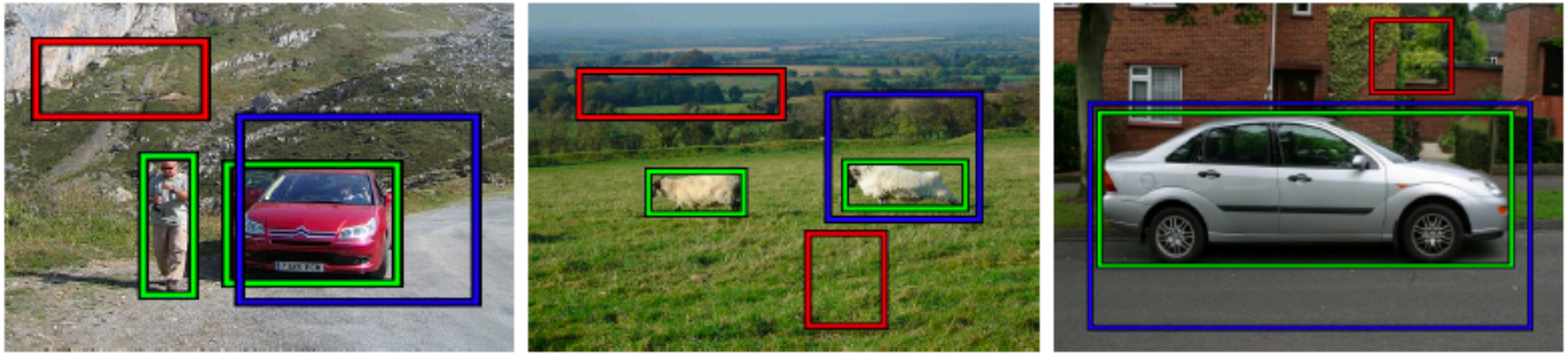


Figure credit: Alexe et al., 2012

This work argued that objects typically

— are unique within the image and stand out as salient

— have a contrasting appearance from surroundings and/or

— have a well-defined closed boundary in space

Object Proposals

Multiscale **Saliency**

— Favors regions with a unique appearance within the image

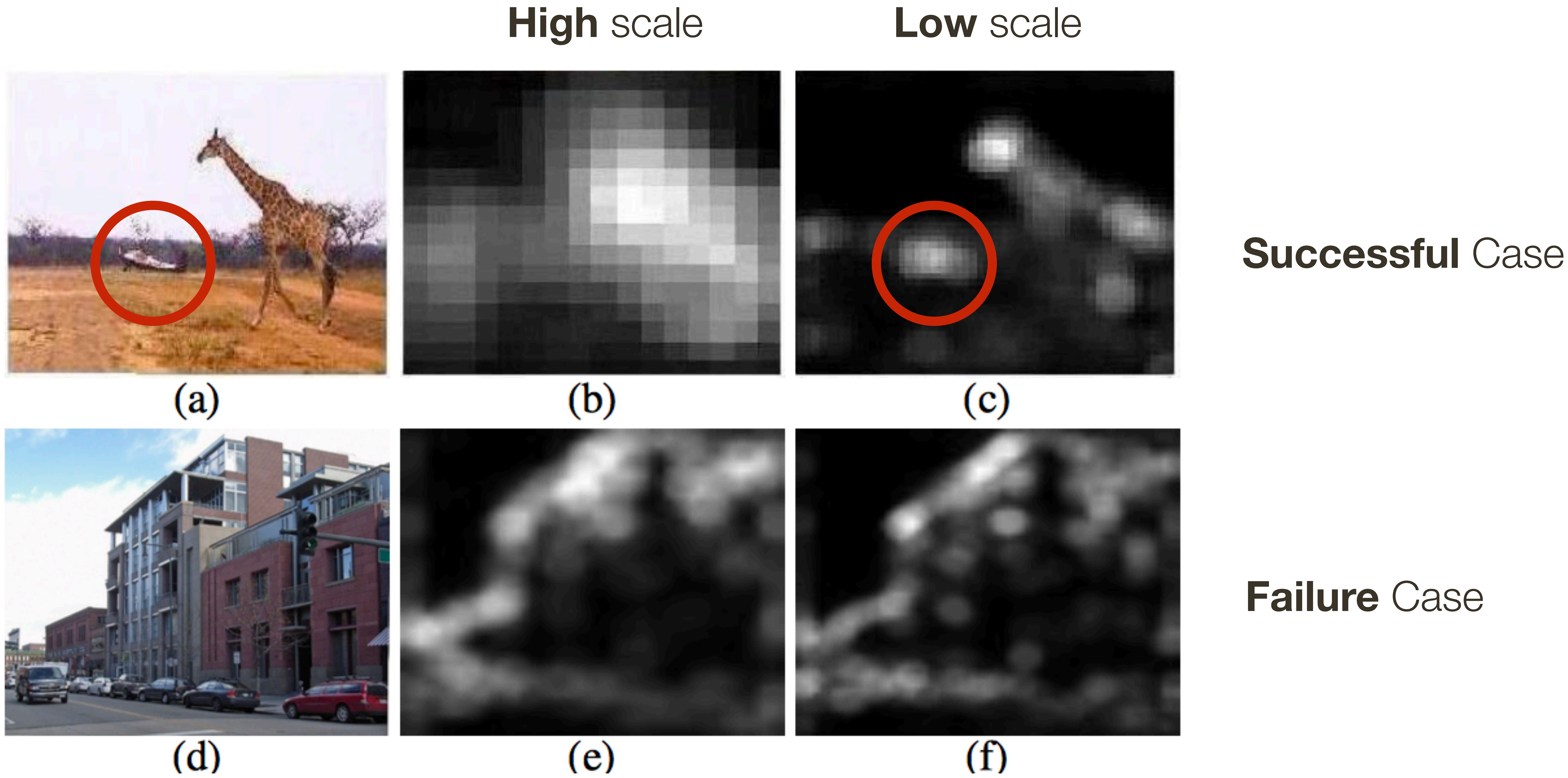
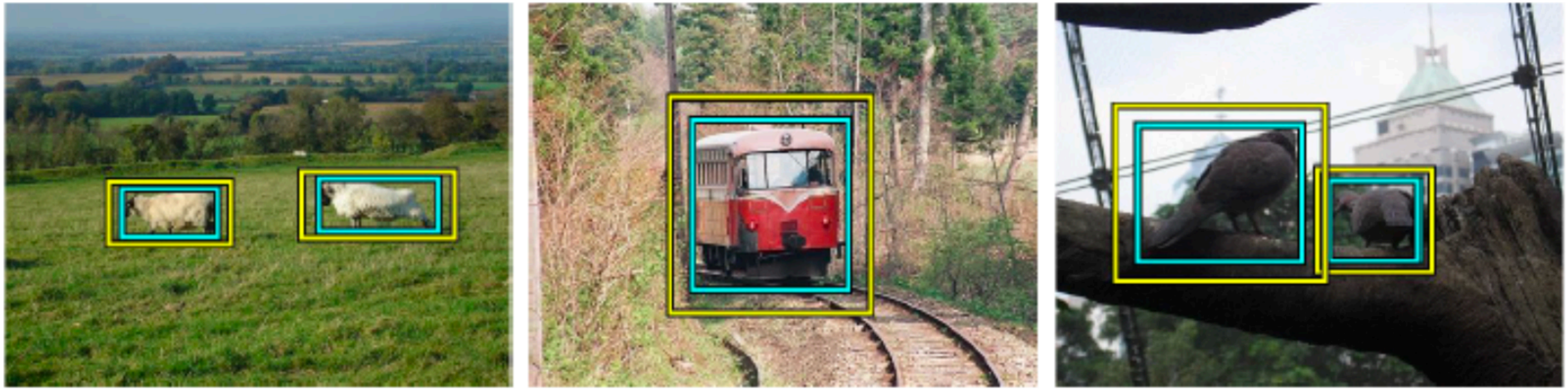


Figure credit: Alexe et al., 2012

Object Proposals

Colour Contrast

— Favors regions with a contrasting colour appearance from immediate surroundings



Successful Cases

Failure Case

Figure credit: Alexe et al., 2012

Object Proposals

Superpixels Straddling

— Favors regions with a well-defined closed boundary

— Measures the extent to which superpixels (obtained by image segmentation) contain pixels both inside and outside of the window

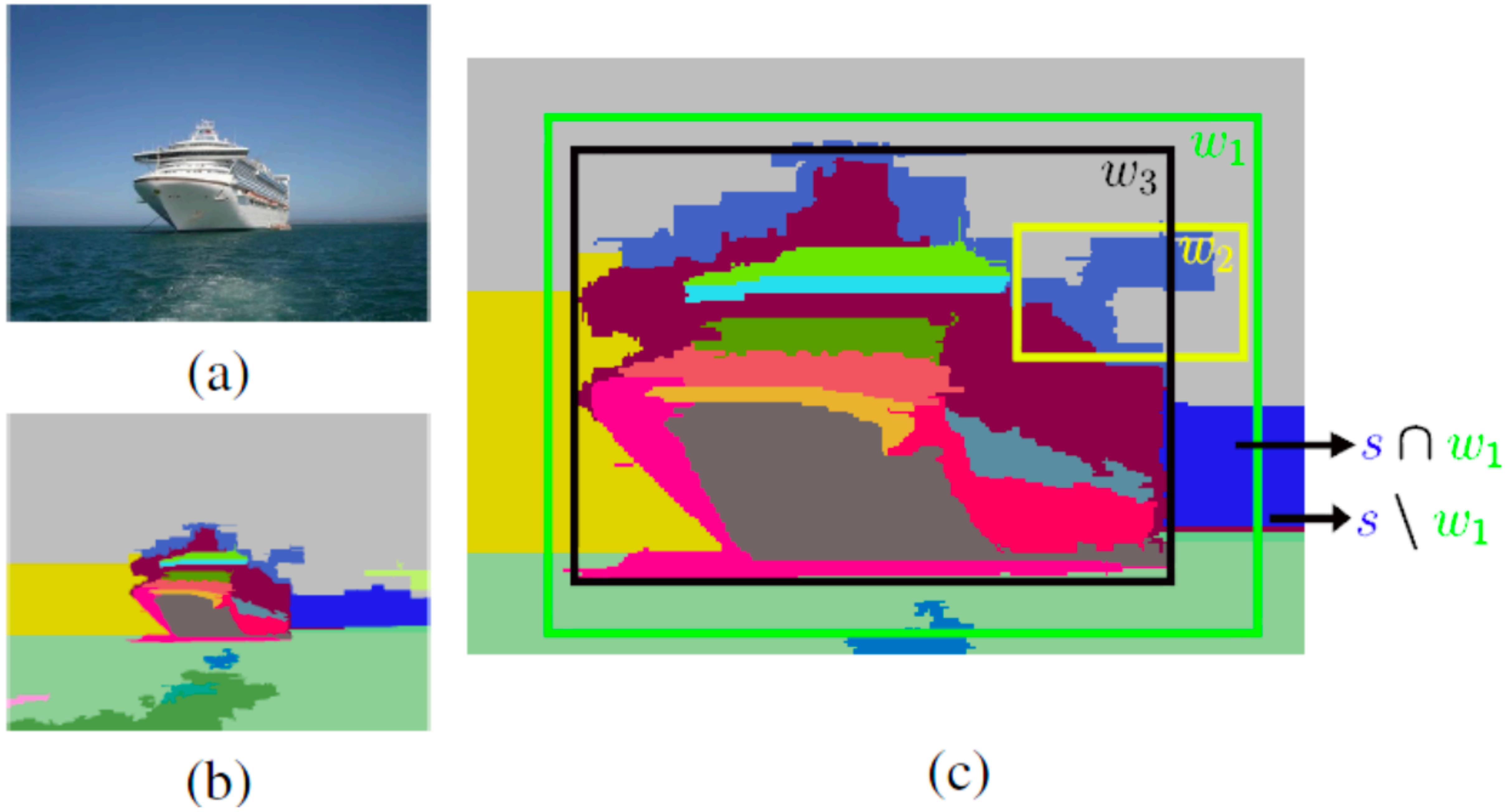


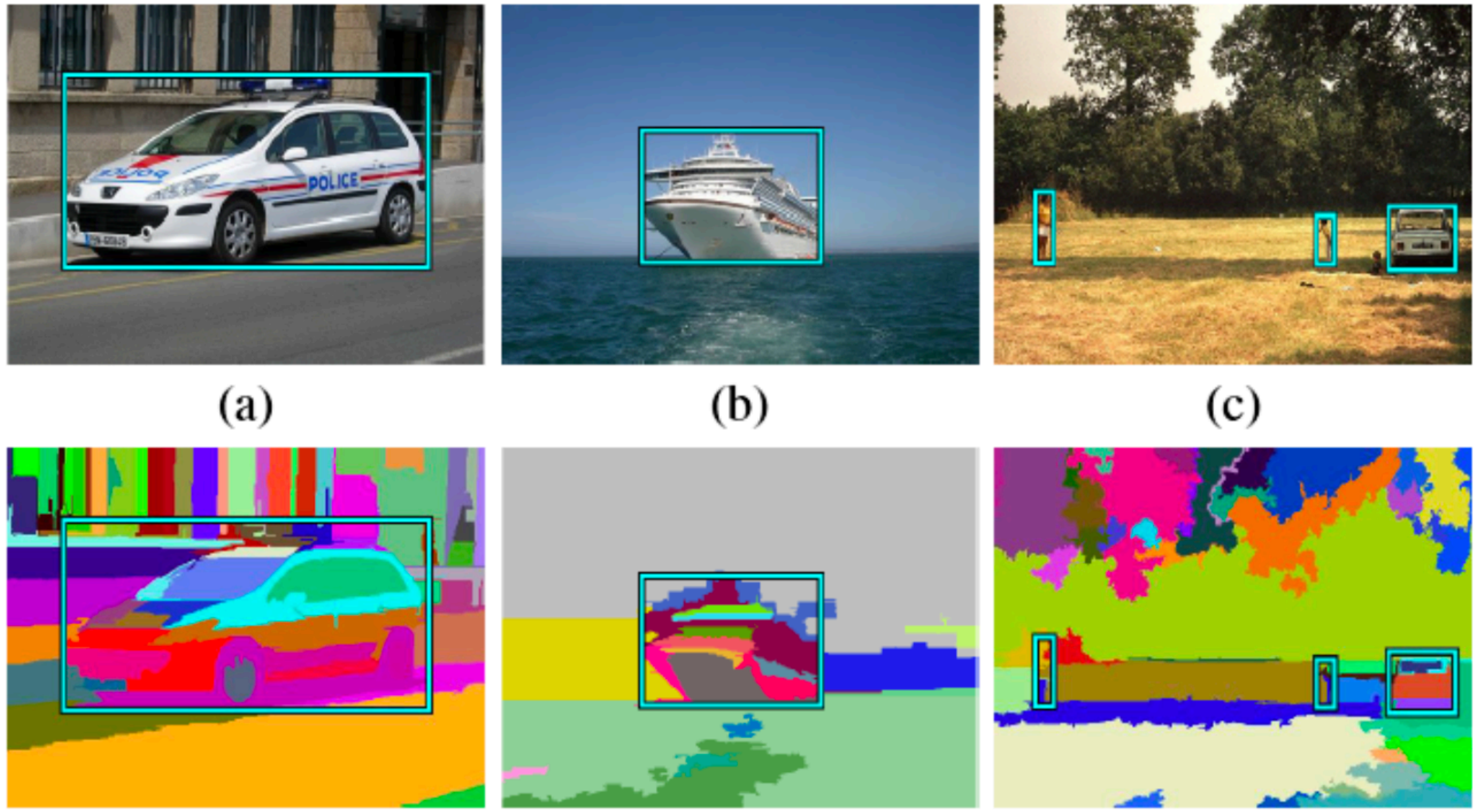
Figure credit: Alexe et al., 2012

Object Proposals

Superpixels Straddling

— Favors regions with a well-defined closed boundary

— Measures the extent to which superpixels (obtained by image segmentation) contain pixels both inside and outside of the window



Successful Cases **Failure** Case

Figure credit: Alexe et al., 2012

Object Proposals

TABLE 2: For each detector [11, 18, 33] we report its performance (left column) and that of our algorithm 1 using the same window scoring function (right column). We show the average number of windows evaluated per image #win and the detection performance as the mean average precision (mAP) over all 20 classes.

	[11]	OBJ- [11]	[18]	OBJ- [18]	ESS-BOW[33]	OBJ-BOW
mAP	0.186	0.162	0.268	0.225	0.127	0.125
#win	79945	1349	18562	1358	183501	2997

Table credit: Alexe et al., 2012

Speeding up [11] HOG pedestrian detector [18] Deformable part model detector [33] Bag of words detector

Summary

Detection scores in the deformable part model are based on both appearance and location

The deformable part model is trained iteratively by alternating the steps

1. Assume components and part locations given; compute appearance and offset models
2. Assume appearance and offset models given; compute components and part locations

An object **proposal** algorithm generates a short list of regions with generic object-like properties that can be evaluated by an object detector in place of an exhaustive sliding window search