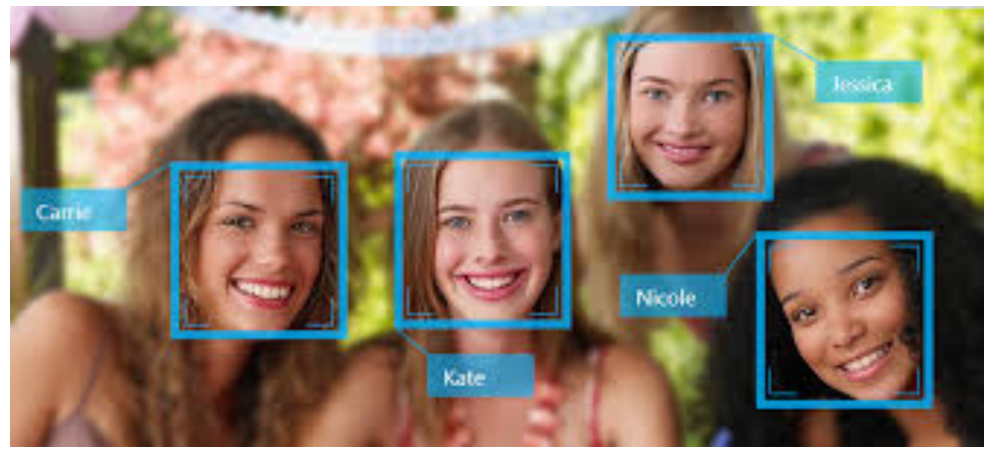
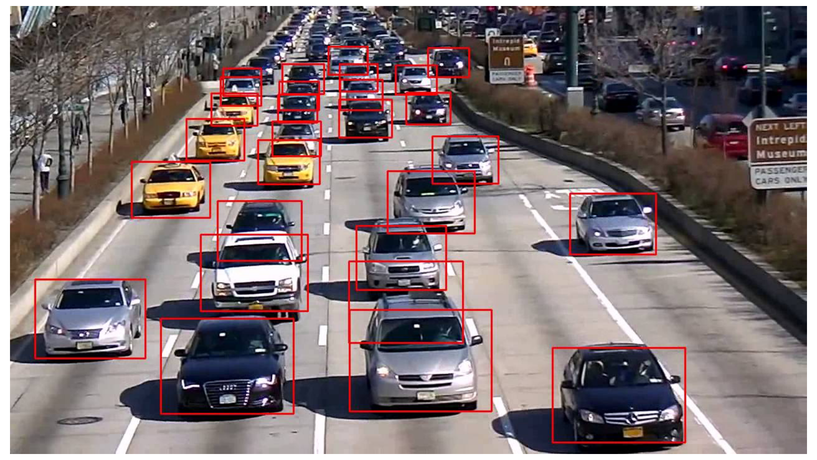
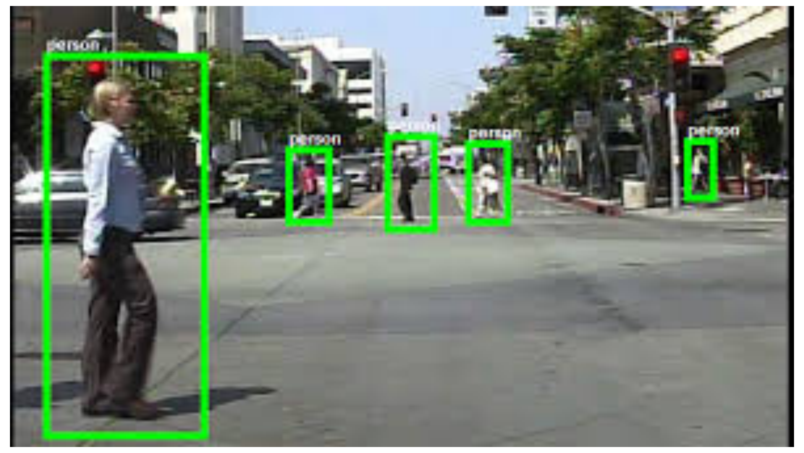
THE UNIVERSITY OF BRITISH COLUMBIA

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:

— **Assignment 5** is out — it will take **time** to run

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).

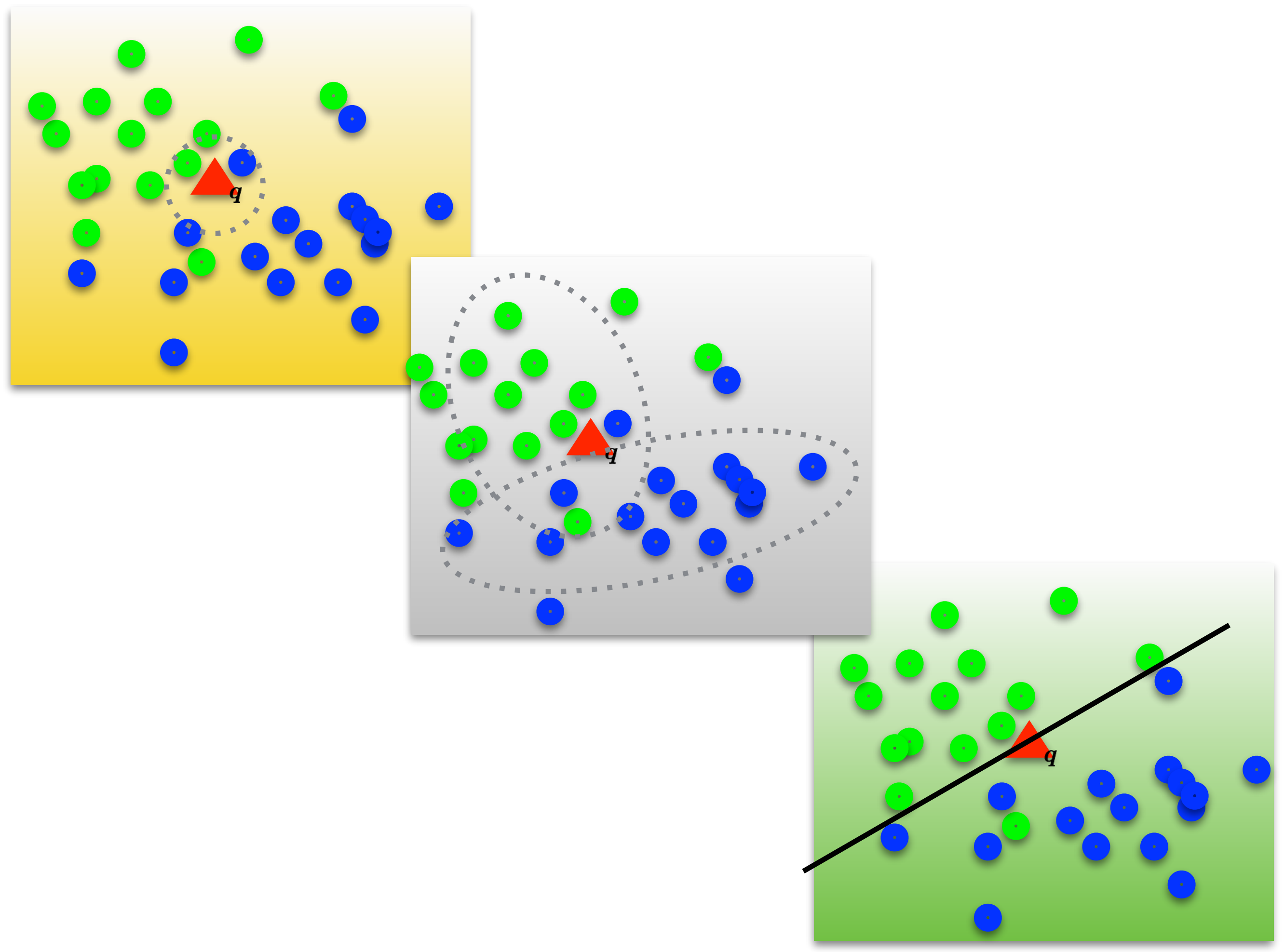
(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat

Image Classification

Classification Algorithms

- Bayes' Classifier
- Nearest Neighbor Classifier
- SVM Classifier



Representation of Images

- Image pixels directly
- Bag of Words

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

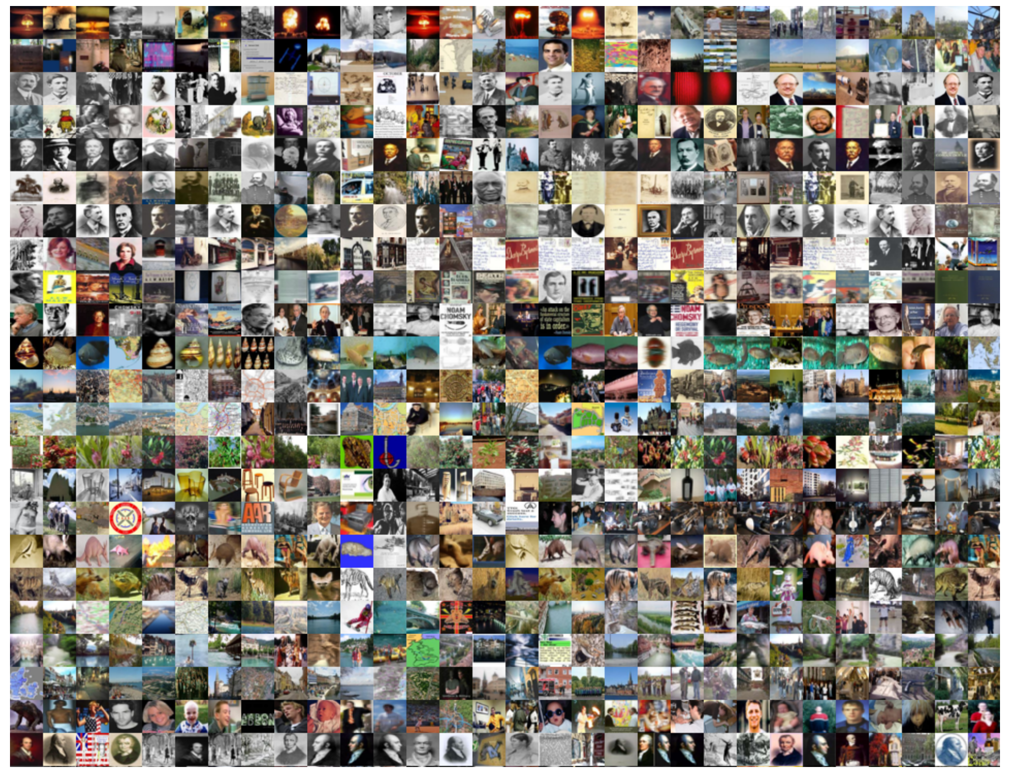
Encode:
build Bags-of-Words (BOW) vectors
for each image

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)

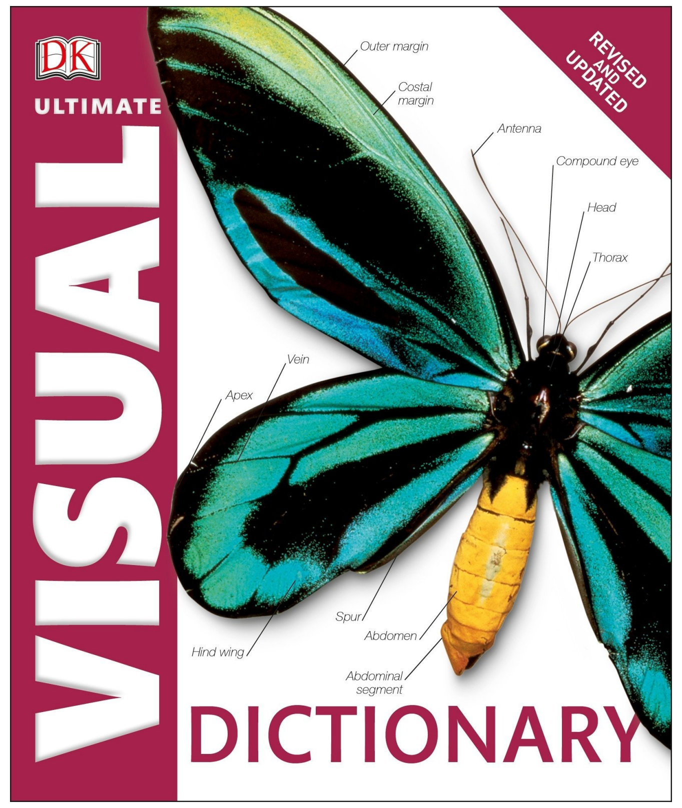


→

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)

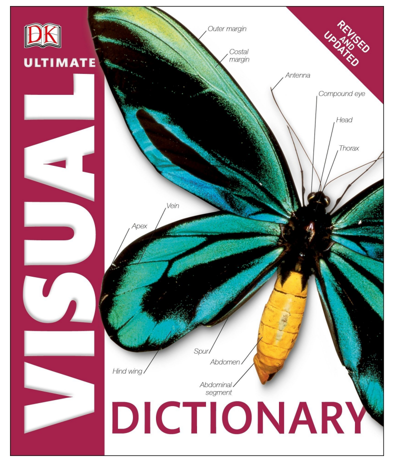
→ **Dictionary Learning**:
Learn Visual Words using clustering →

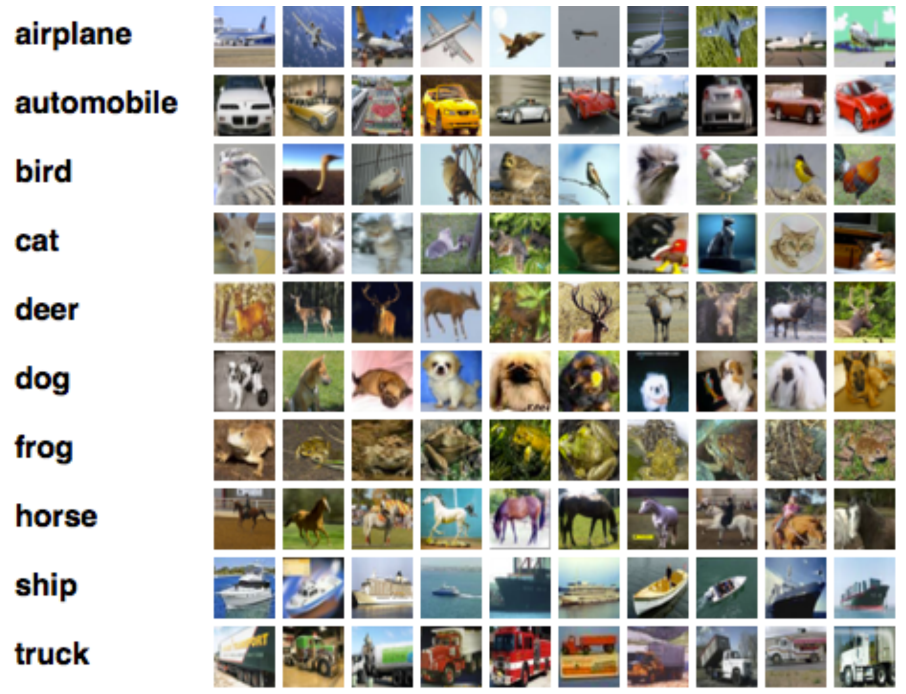
Output: dictionary of visual words

Input: training images, dictionary

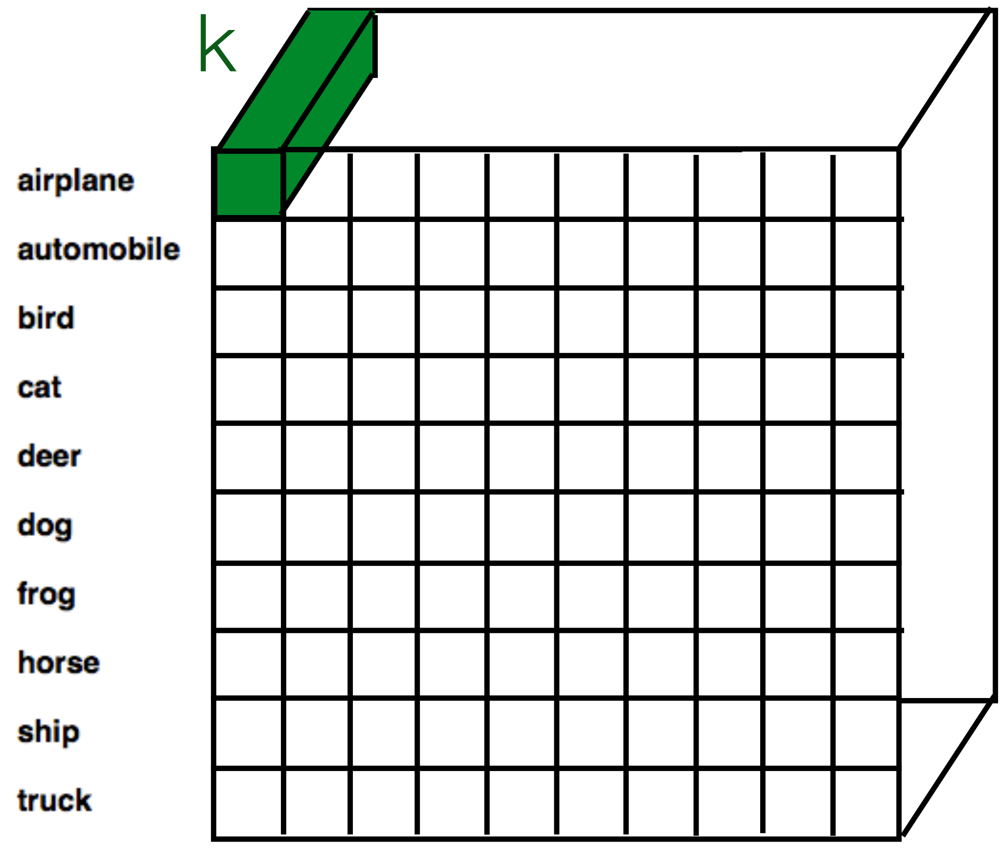
→ **Encode**:
build Bags-of-Words (BOW) vectors for each image →

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

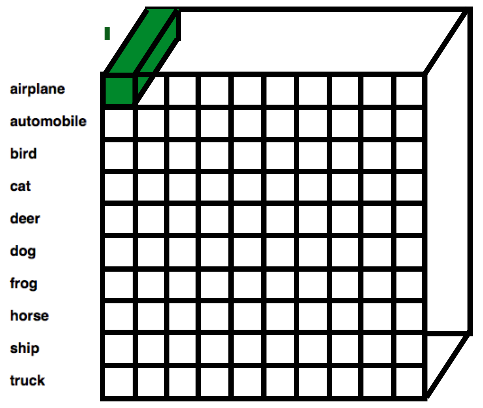
→ **Encode**:
build Bags-of-Words (BOW) vectors
for each image →

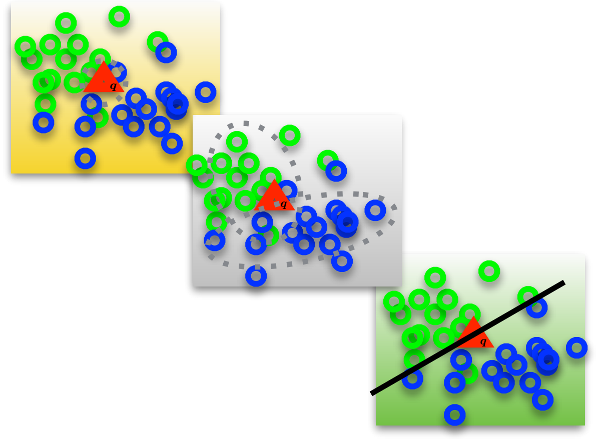
Output: histogram representation for each training image

Input: histogram representation for each training image + labels

→ **Classify**:
Train data using BOWs →

Output: parameters if the classifier



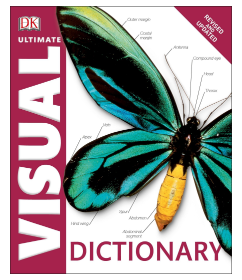


Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

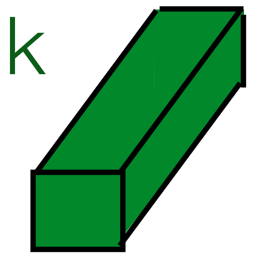
Standard **Bag-of-Words** Pipeline (for image classification) — **Testing**

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)

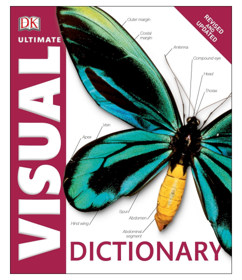
Standard **Bag-of-Words** Pipeline (for image classification) — **Testing**

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

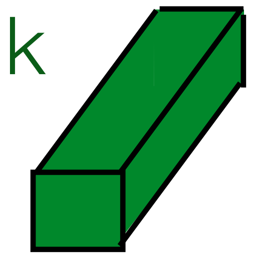
Input: test image, dictionary



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

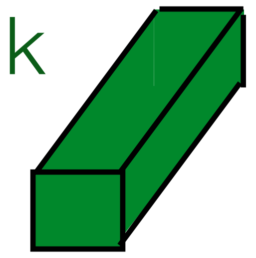
Output: histogram representation for test image

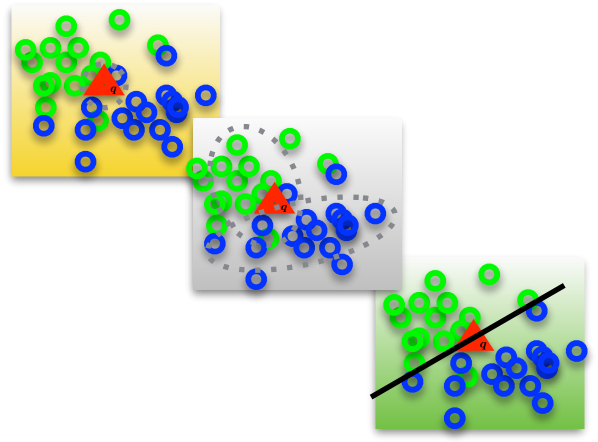
k



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)

Dictionary Learning:
Learn Visual Words using clustering

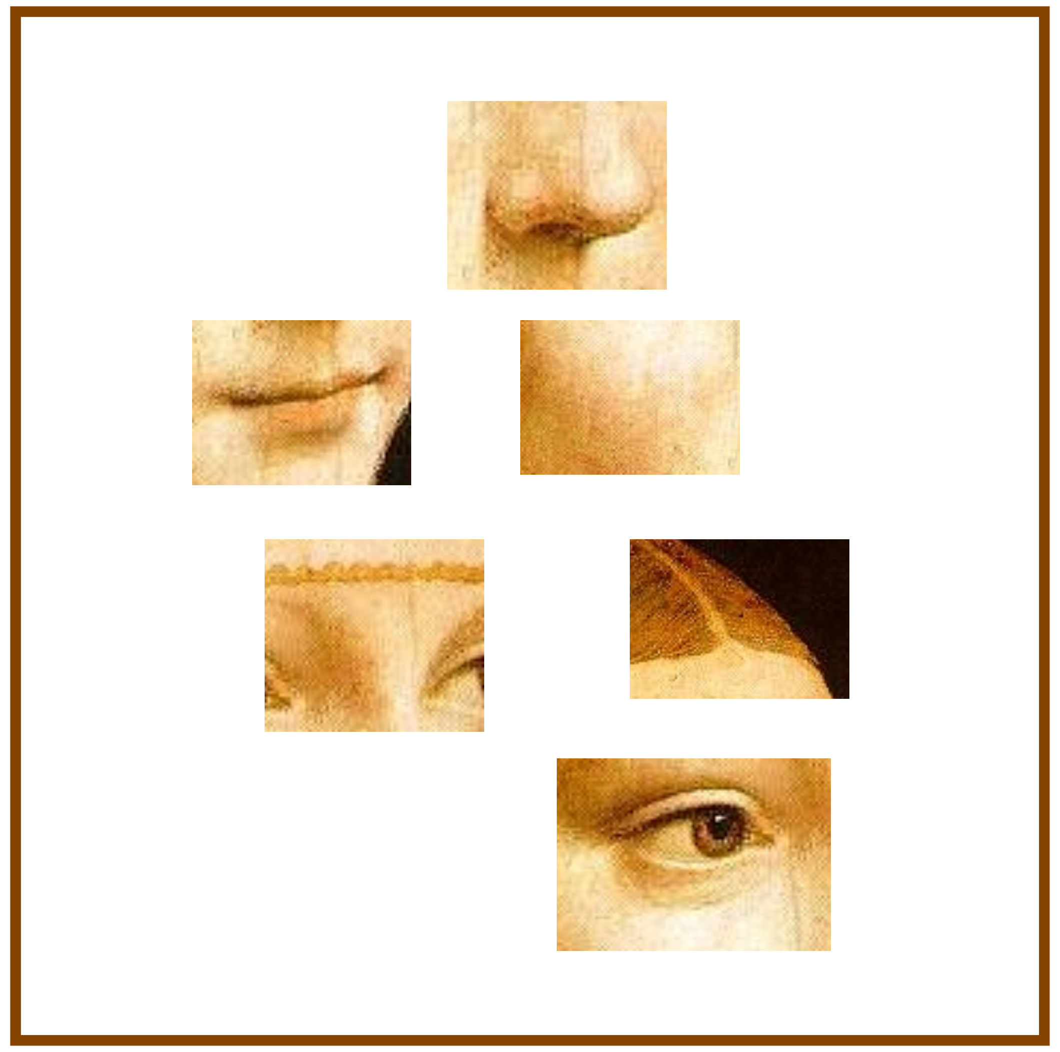
Encode:
build Bags-of-Words (BOW) vectors
for each image

Classify:
Train and test data using BOWs

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

1. **Dictionary Learning**: Learn Visual Words using Clustering

1. **extract features** (e.g., SIFT) from images



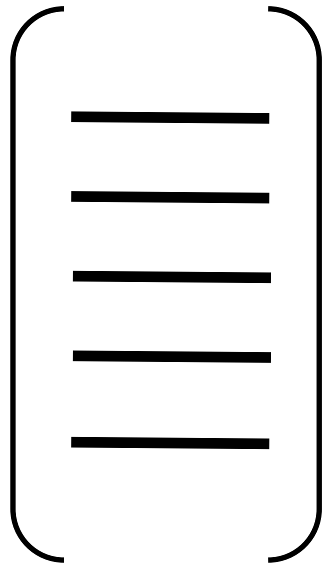
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

1. Dictionary Learning: Learn Visual Words using Clustering

2. **Learn visual dictionary** (e.g., K-means clustering)

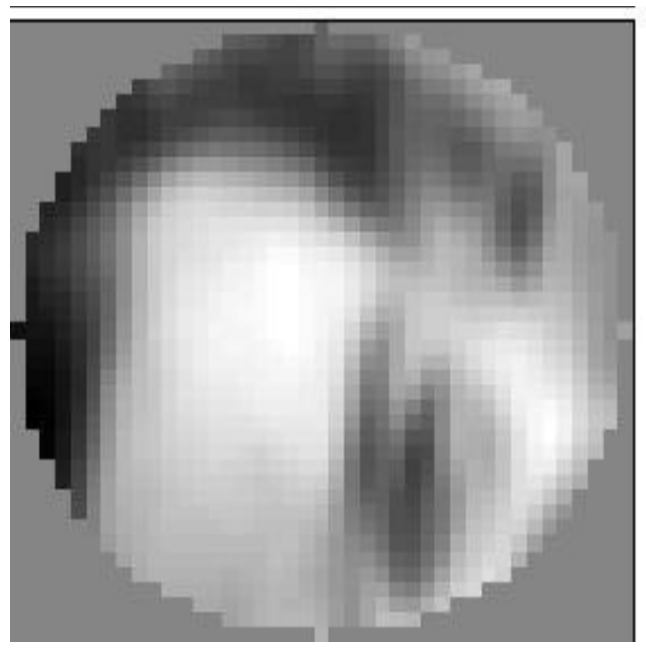
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Extracting **SIFT** Patches

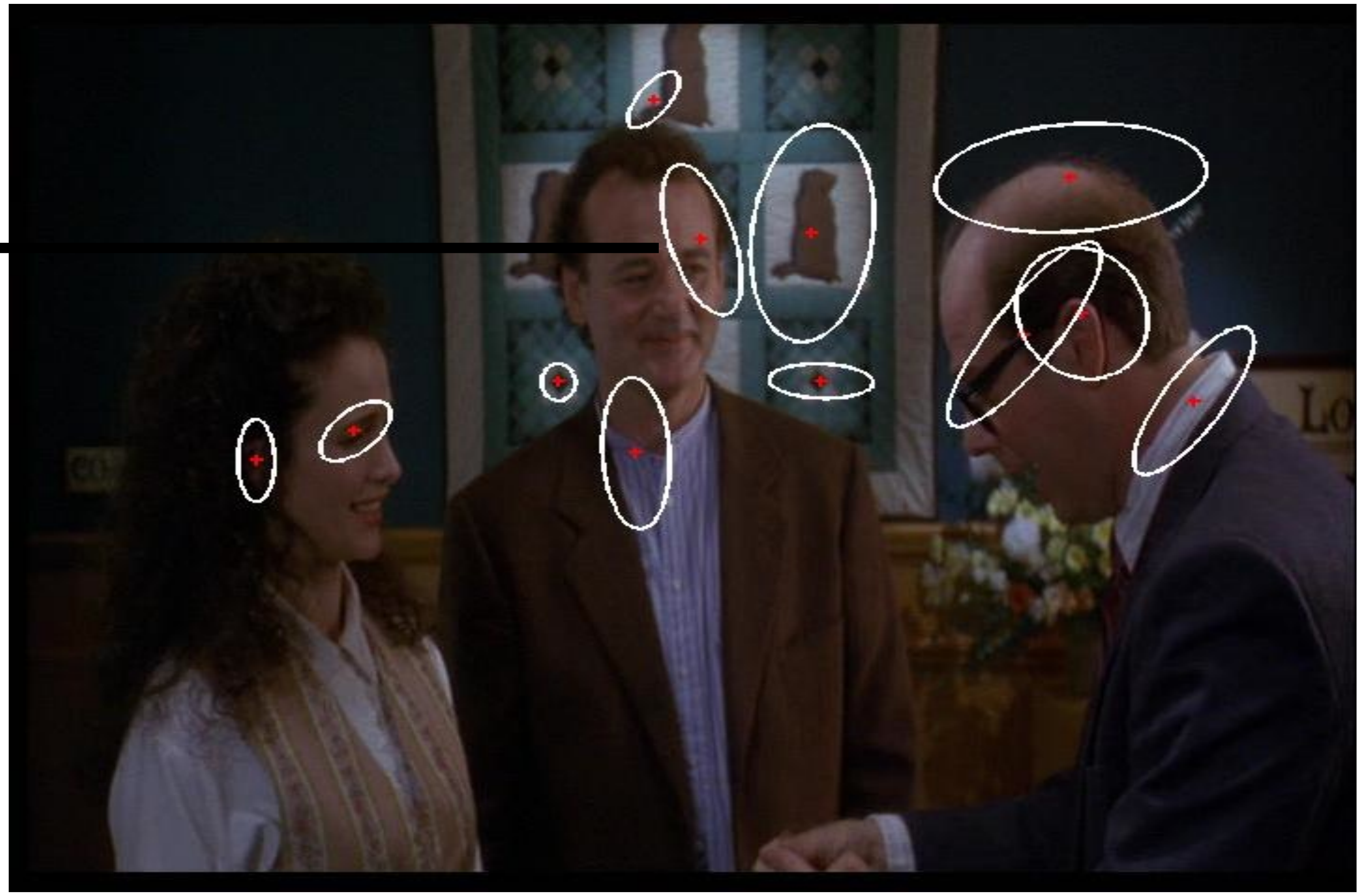


Compute SIFT descriptor

[Lowe'99]



Normalize patch



Detect patches

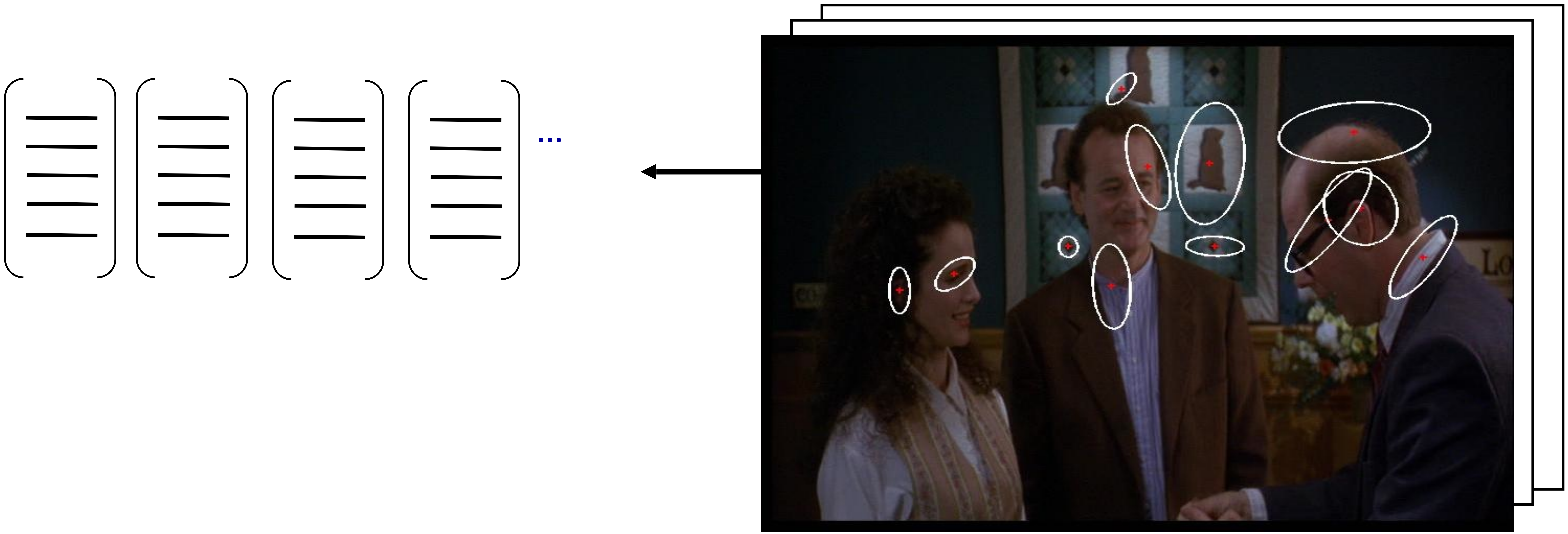
[Mikojaczyk and Schmid '02]

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

[Sivic & Zisserman, '03]

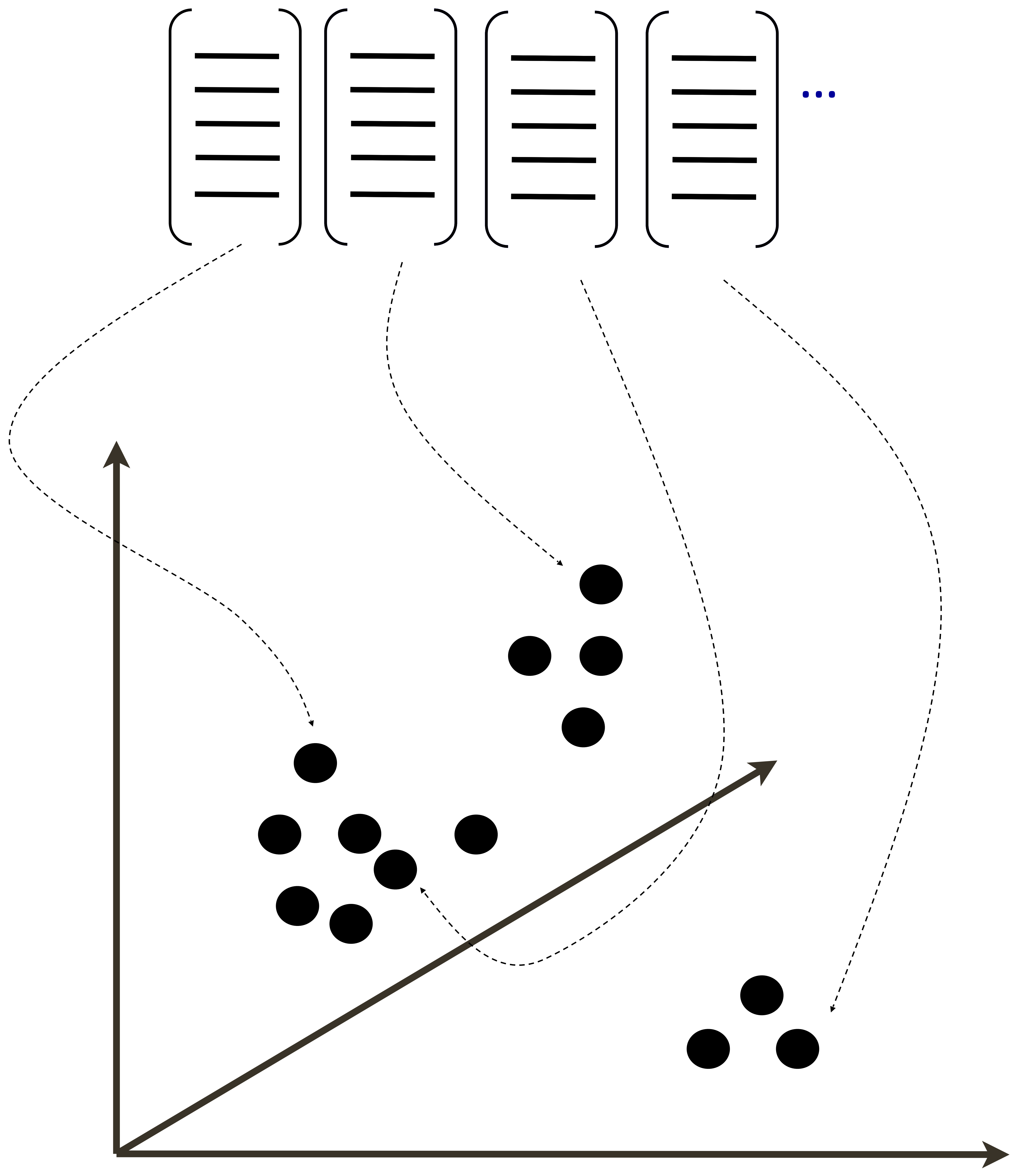
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

Extracting **SIFT** Patches



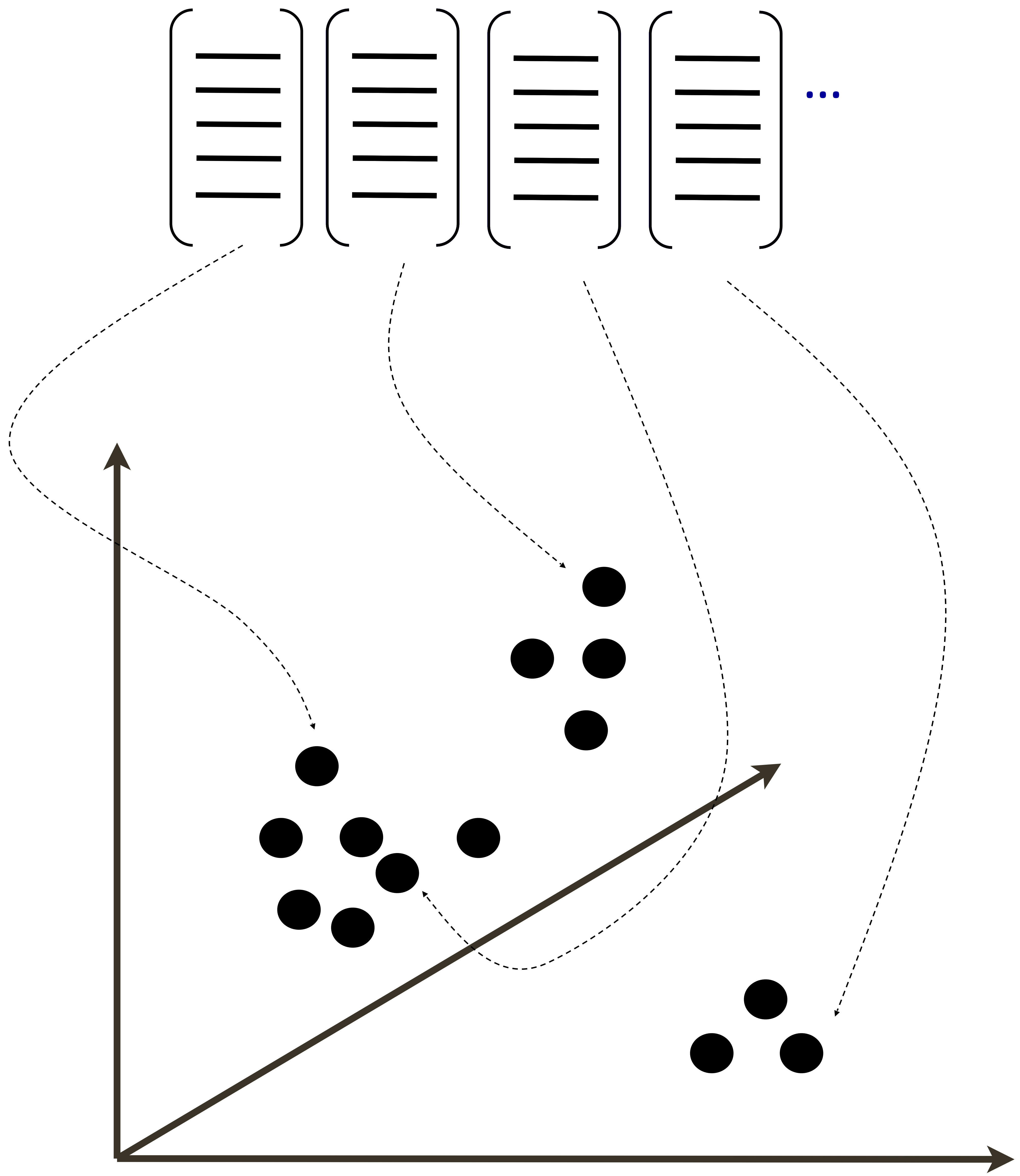
Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

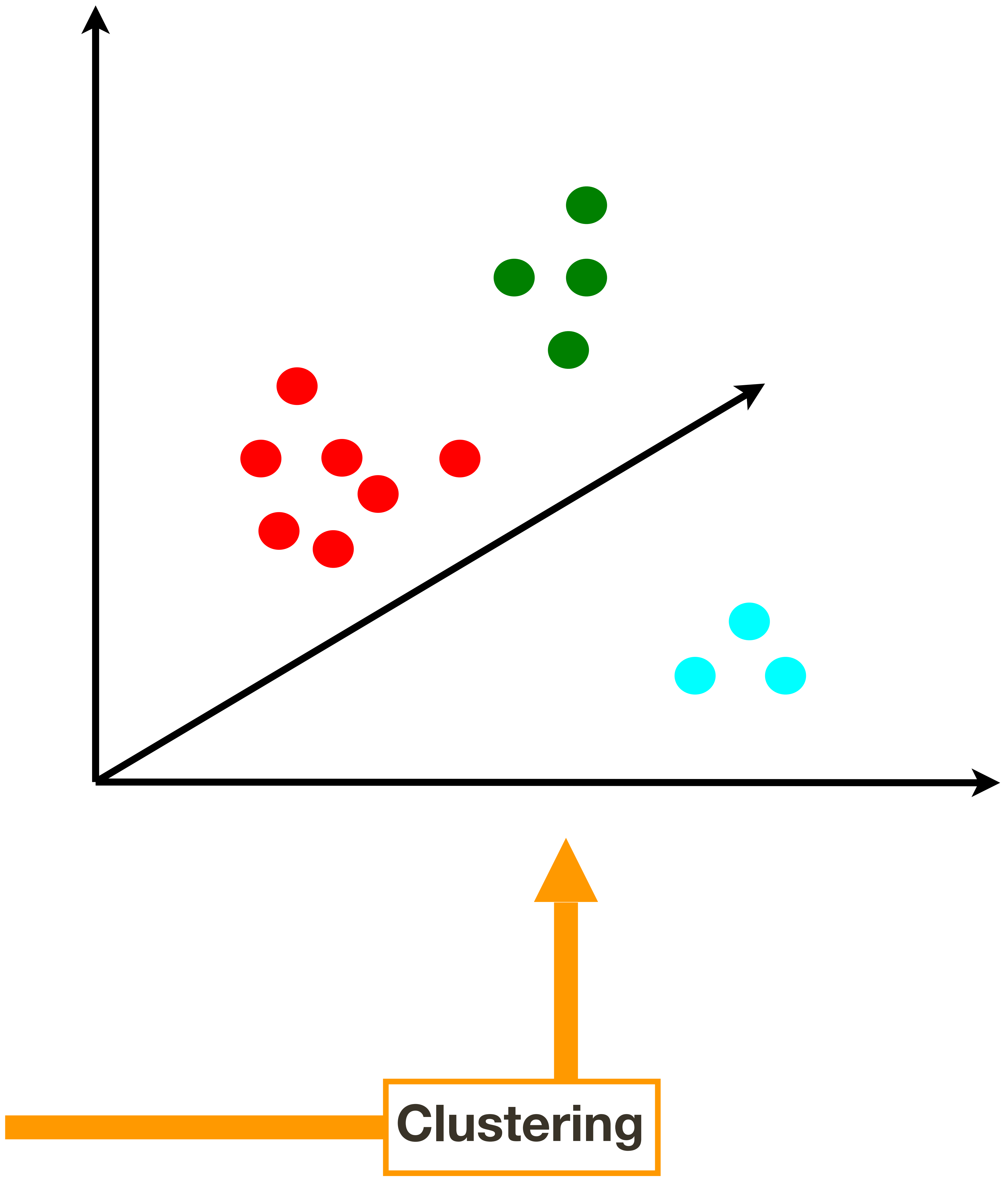
Creating **Dictionary**



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

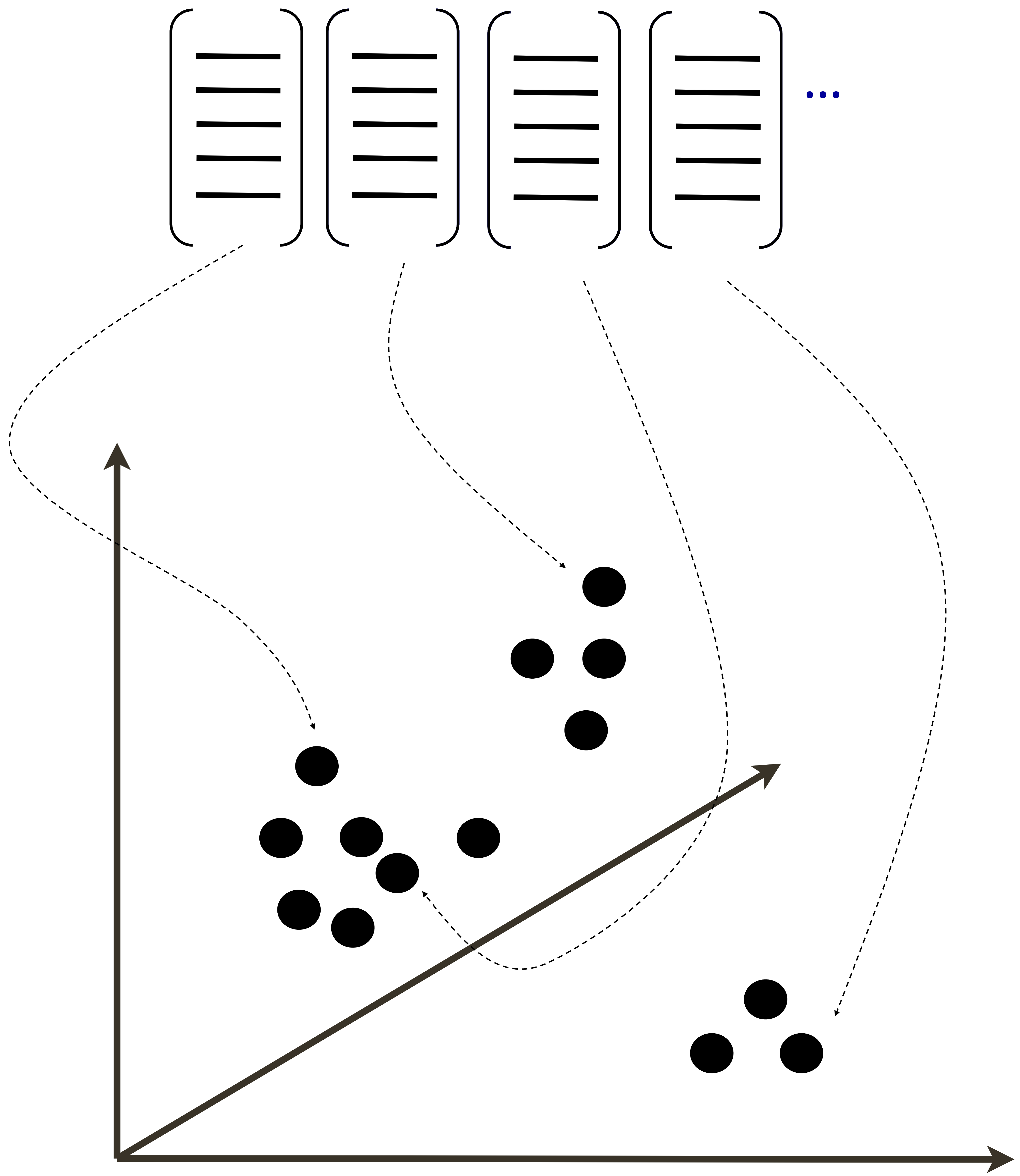
Creating **Dictionary**

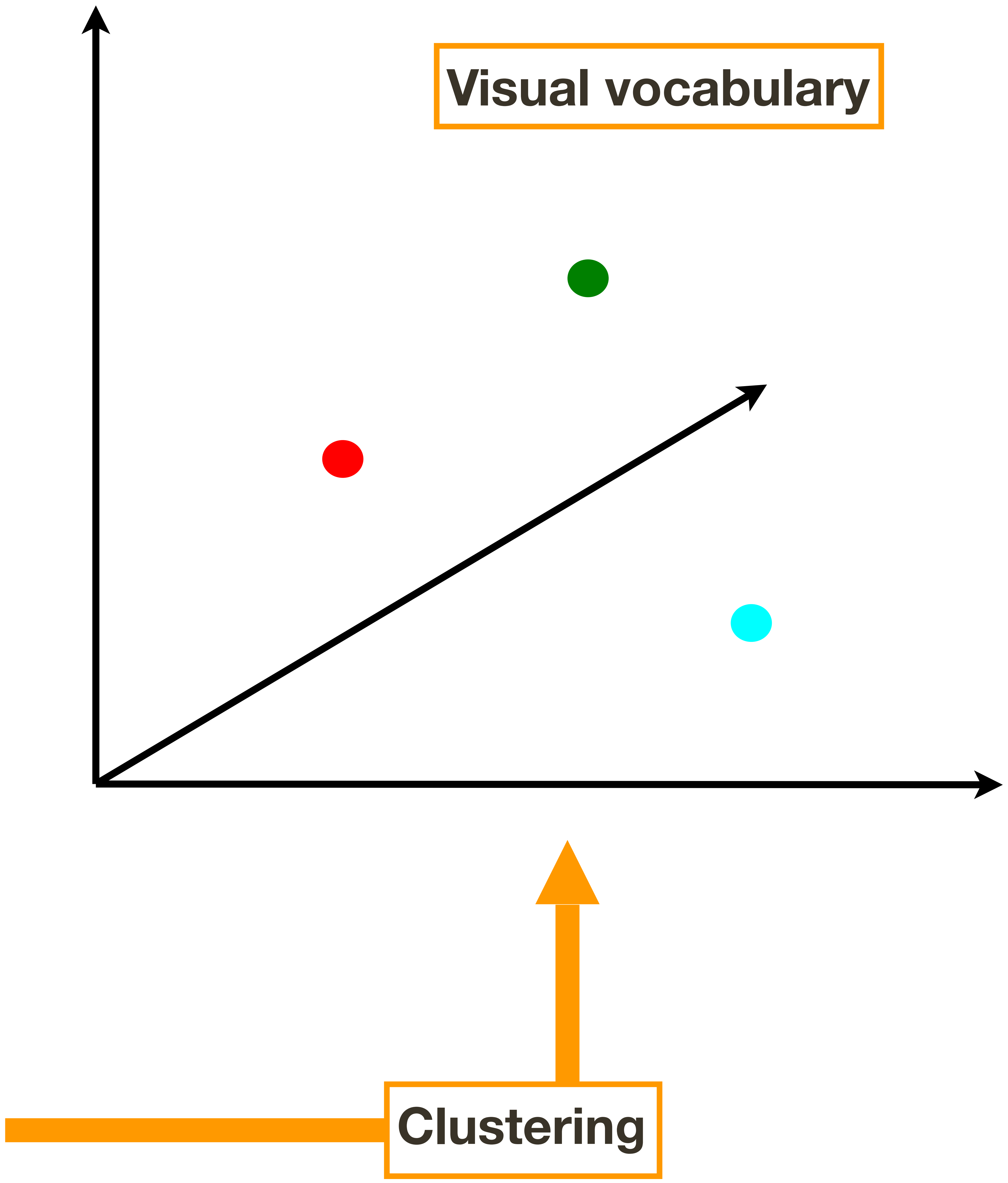




Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

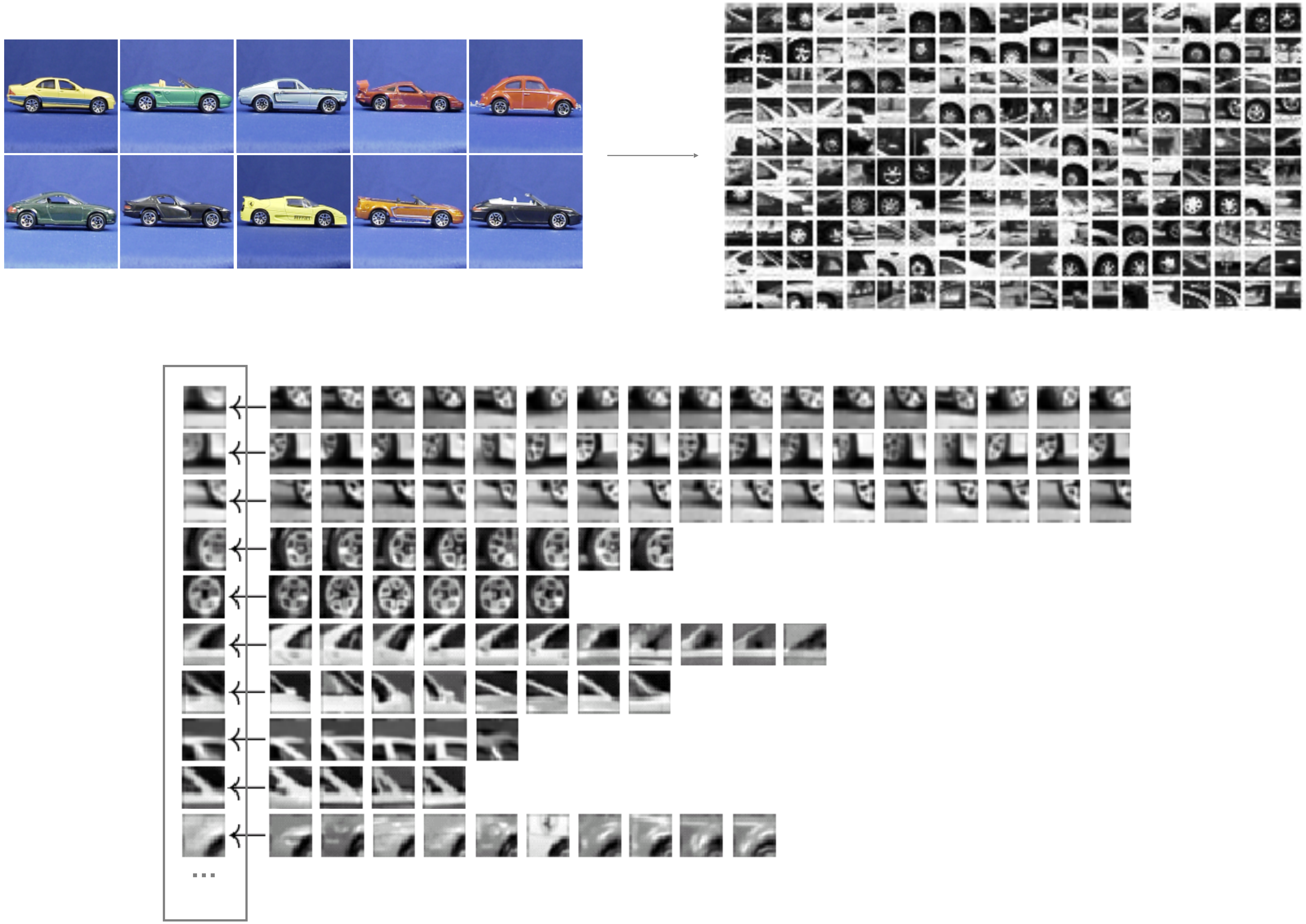
Creating **Dictionary**





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)

Dictionary Learning:
Learn Visual Words using clustering

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

Classify:
Train and test data using BOWs

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

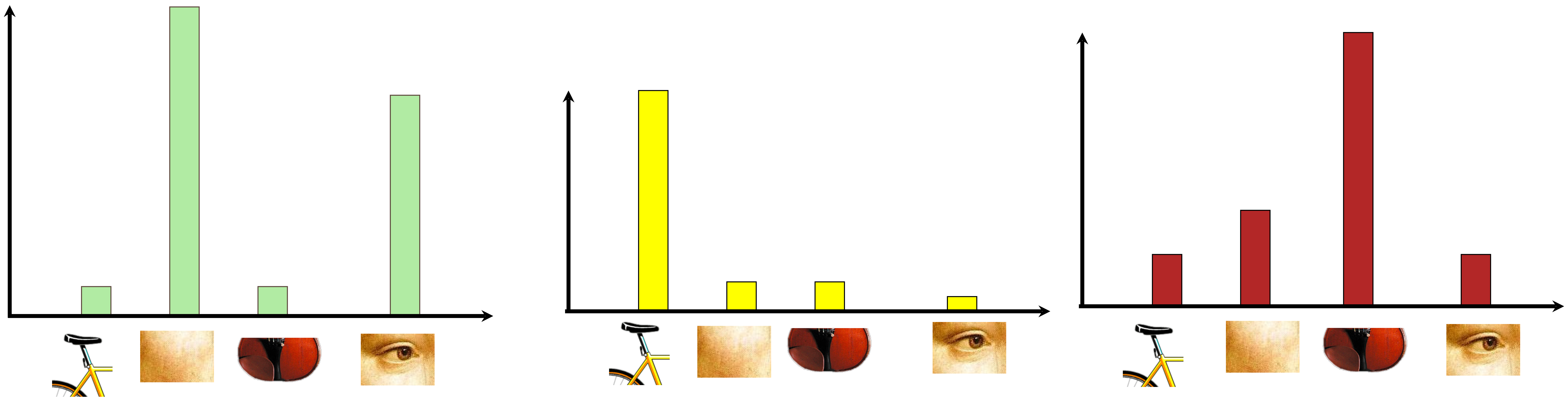
2. **Encode:** build Bag-of-Words (BOW) vectors for each image

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

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

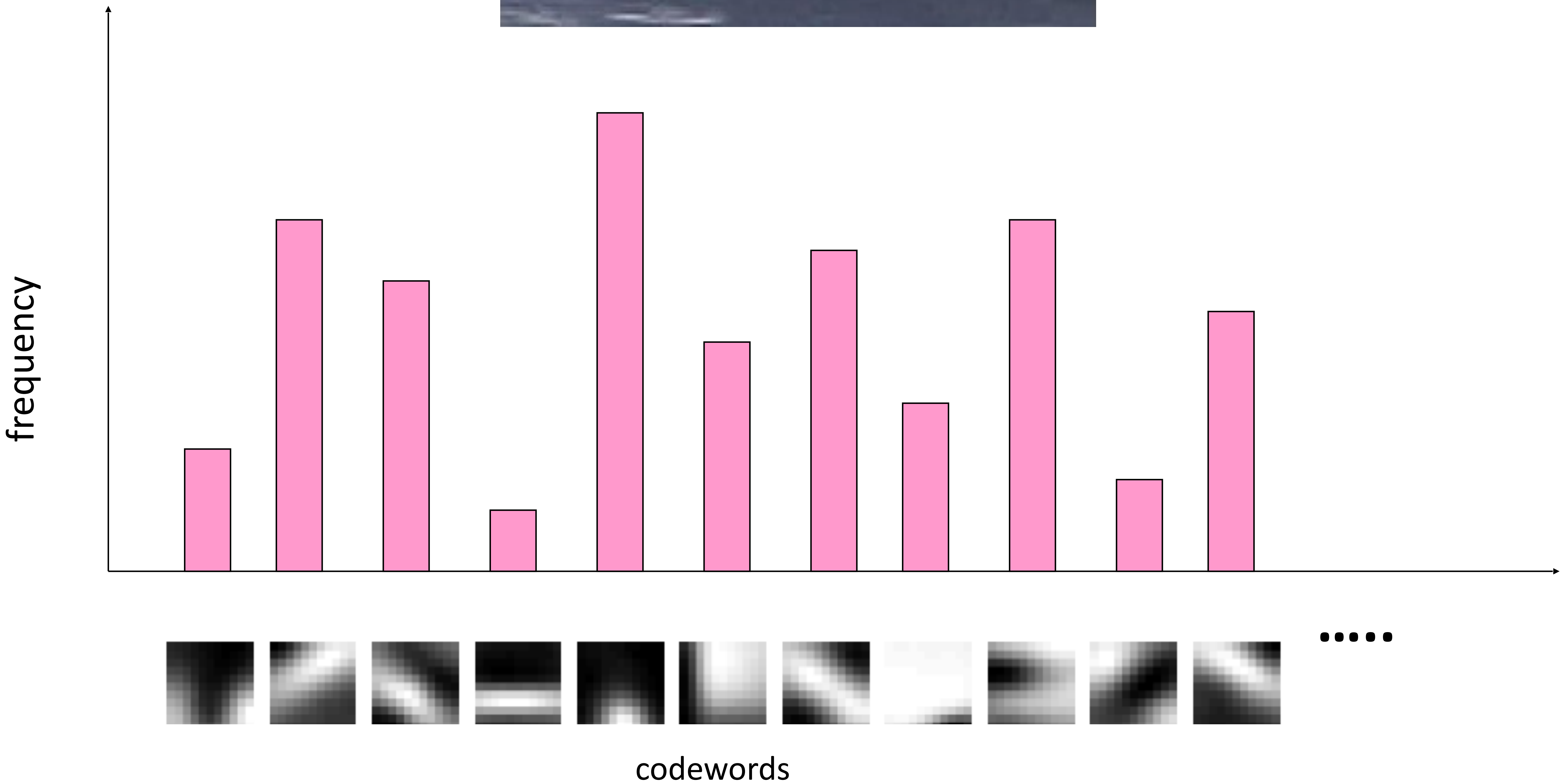
2. Encode: build Bag-of-Words (BOW) vectors for each image

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)

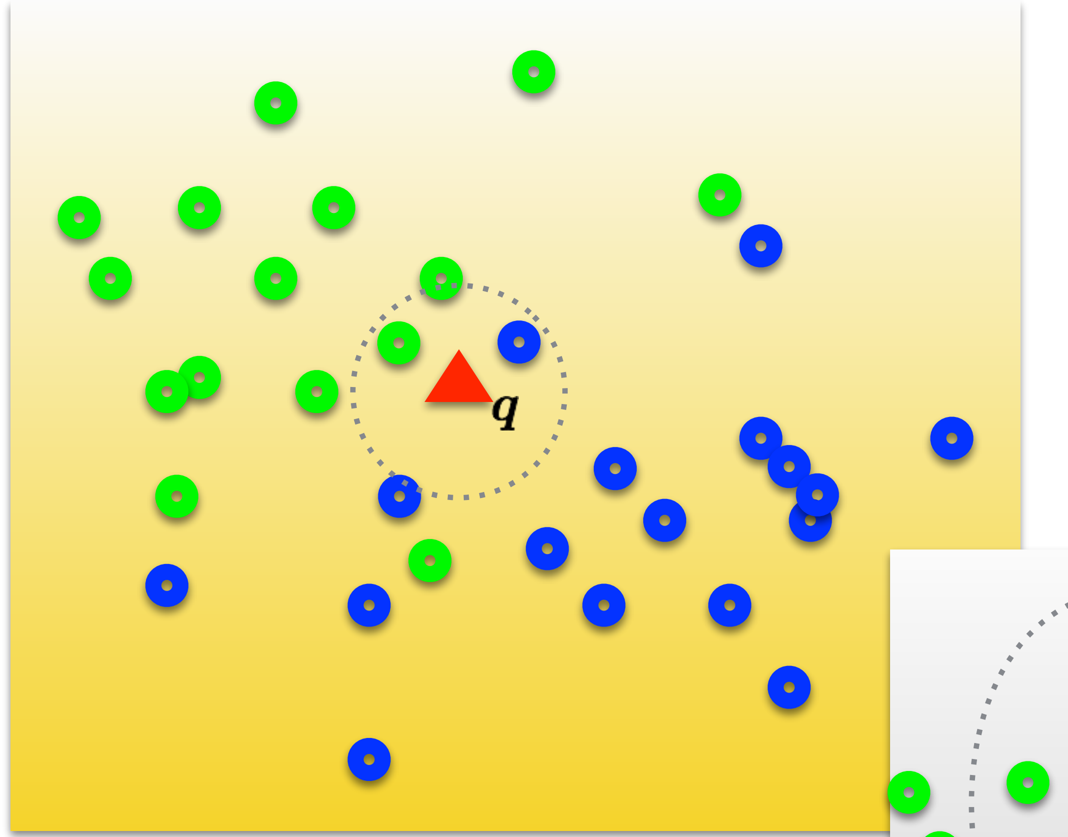
Dictionary Learning:
Learn Visual Words using clustering

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

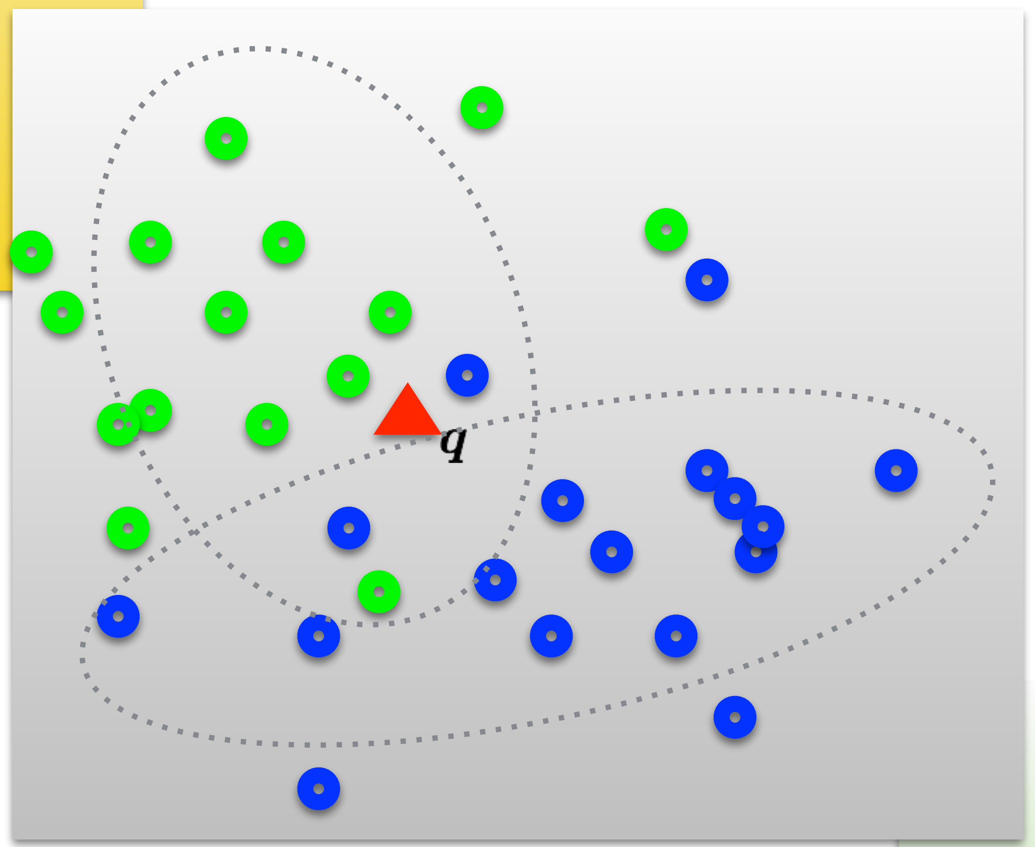
Classify:
Train and test data using BOWs

Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

3. **Classify**: Train and text classifier using BOWs

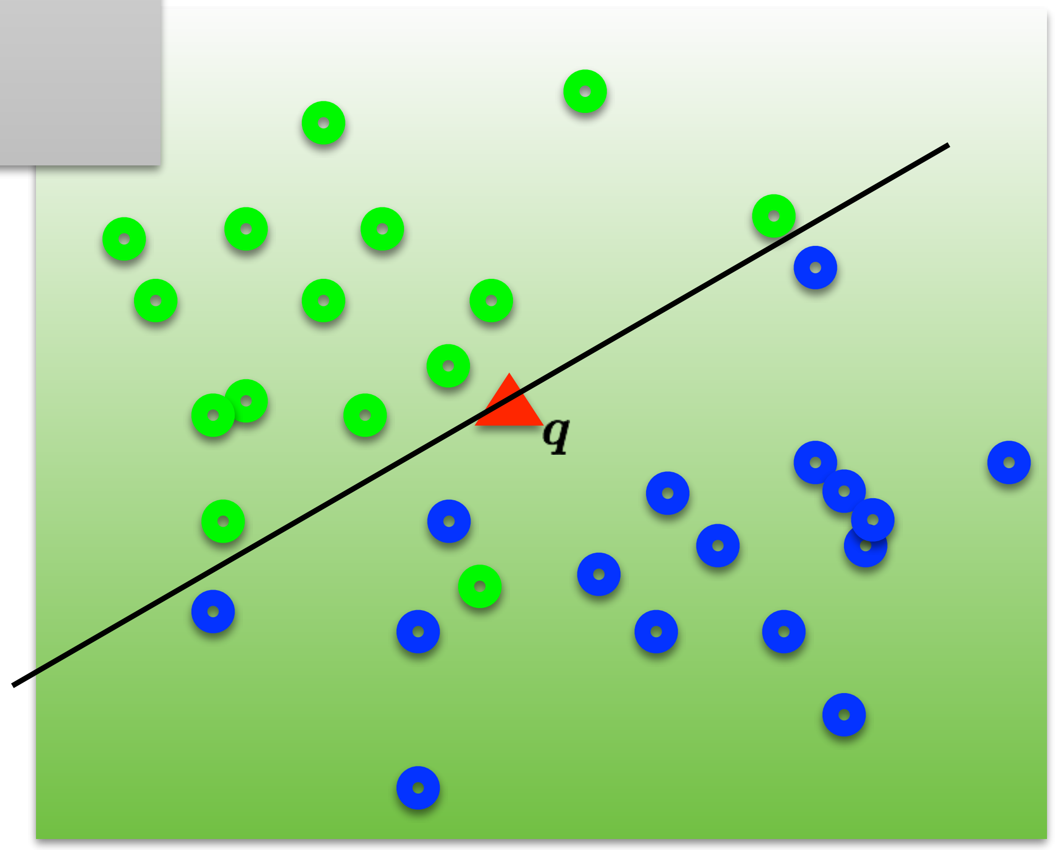


K nearest neighbors



Naïve Bayes

Support Vector Machine



Slide Credit: Ioannis (Yannis) Gkioulekas (CMU)

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

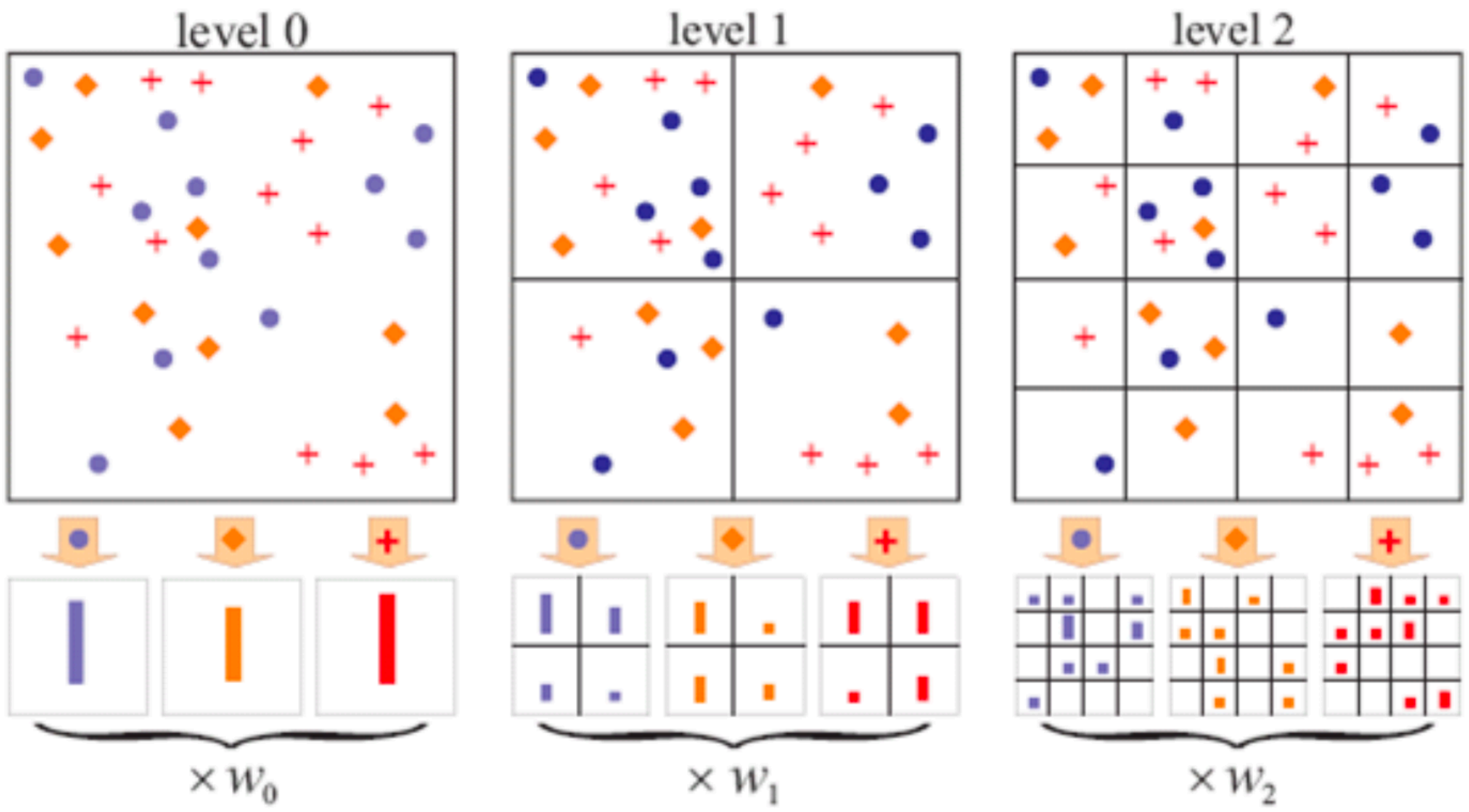


Fig. 16.8 in Forsyth & Ponce (2nd ed.).
Original credit: Lazebnik et al., 2006

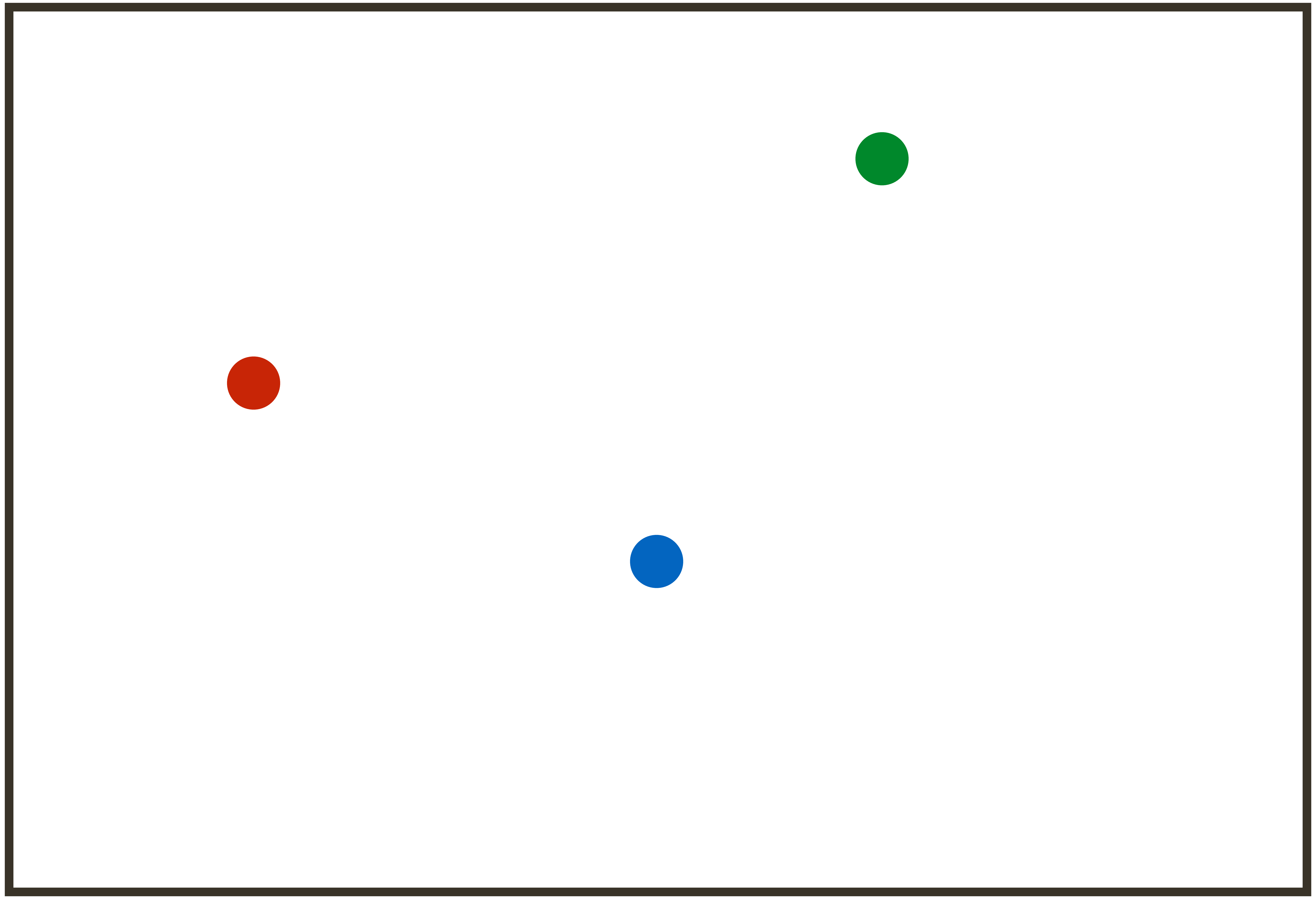
VLAD (Vector of Locally Aggregated Descriptors)

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

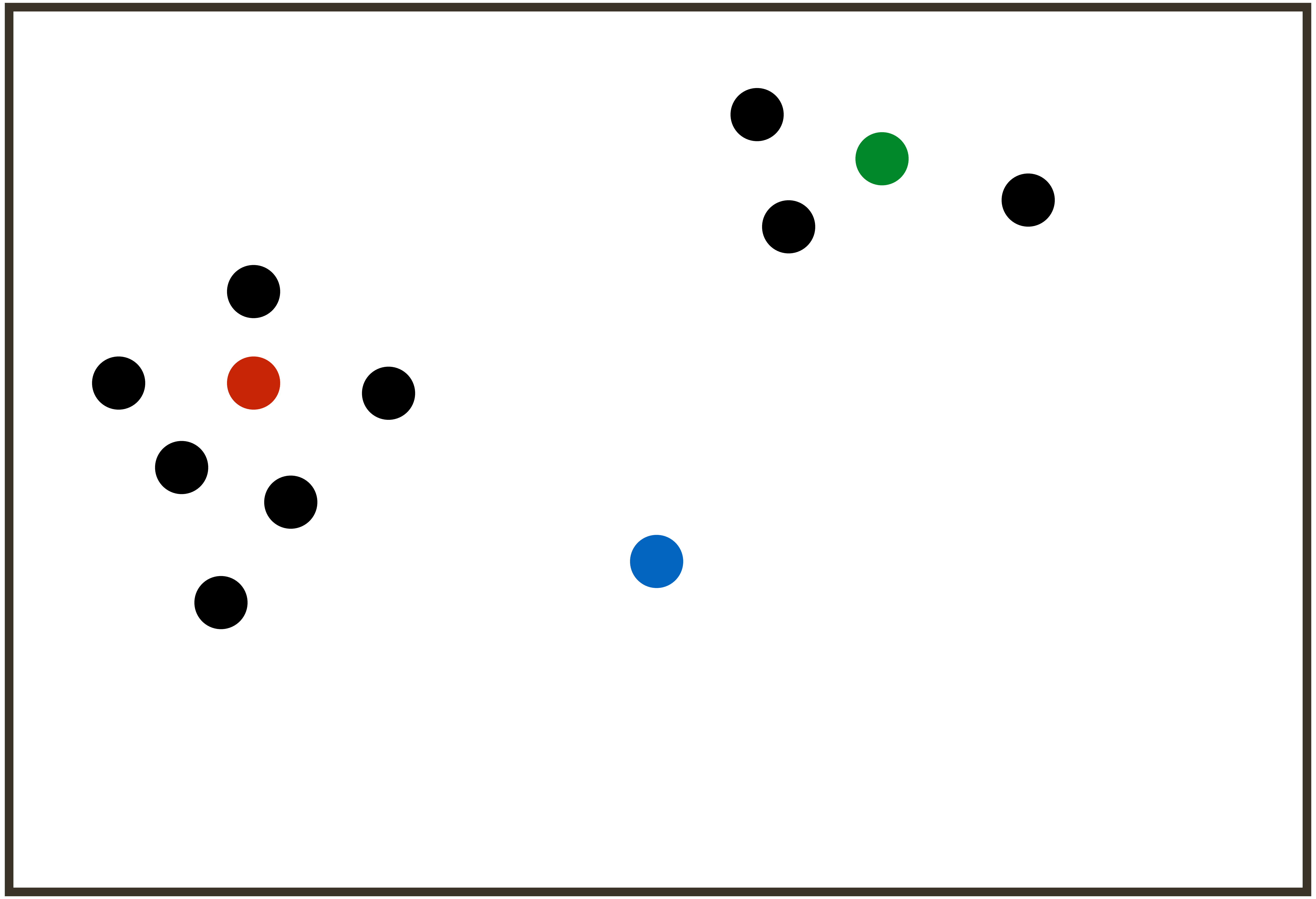
For example, it might be useful to describe how local descriptors are quantized to their visual words

In the VLAD representation, instead of incrementing the histogram bin by one, we increment it by the **residual** vector ***x – c(x)***

Example: VLAD

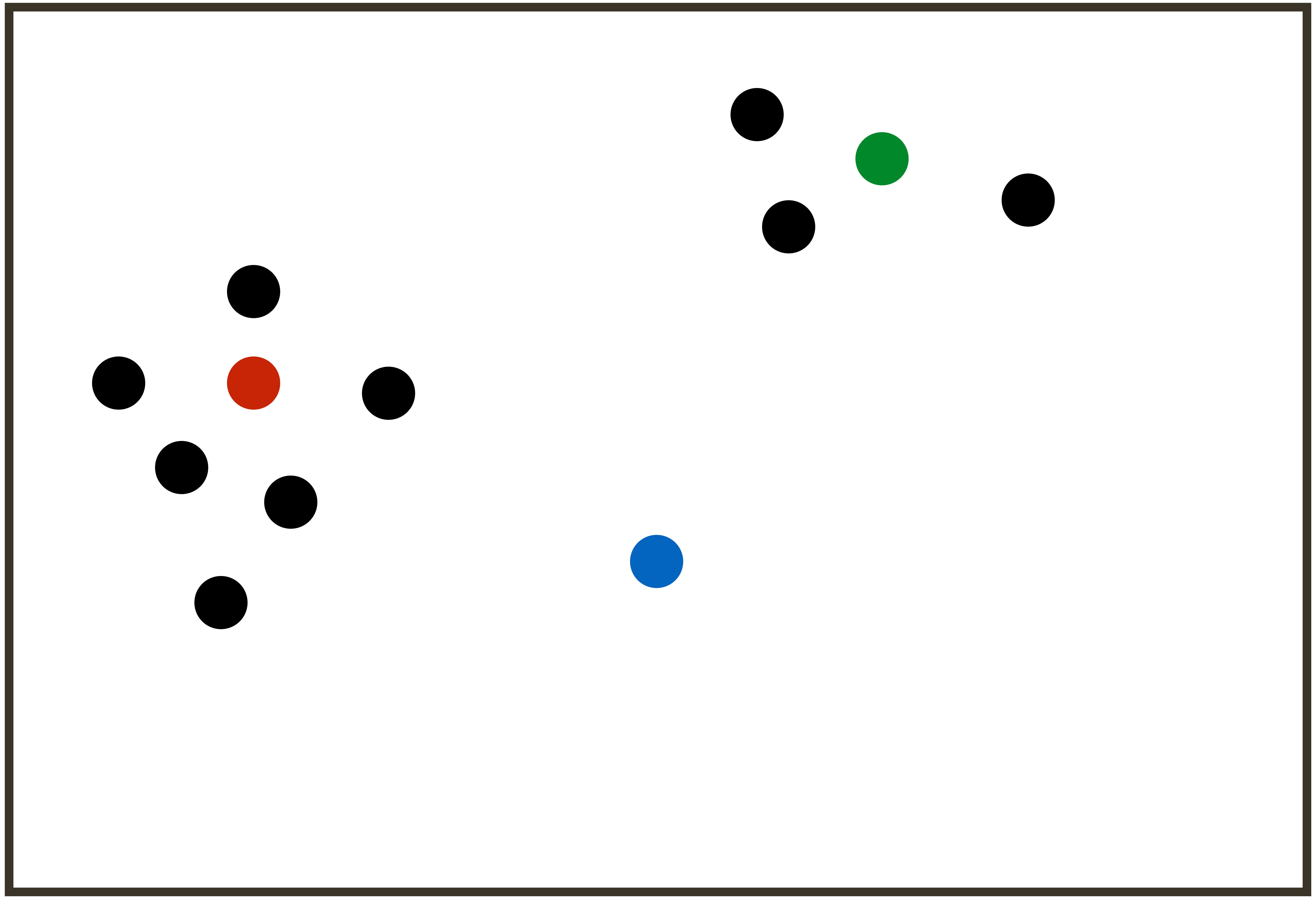


Example: VLAD

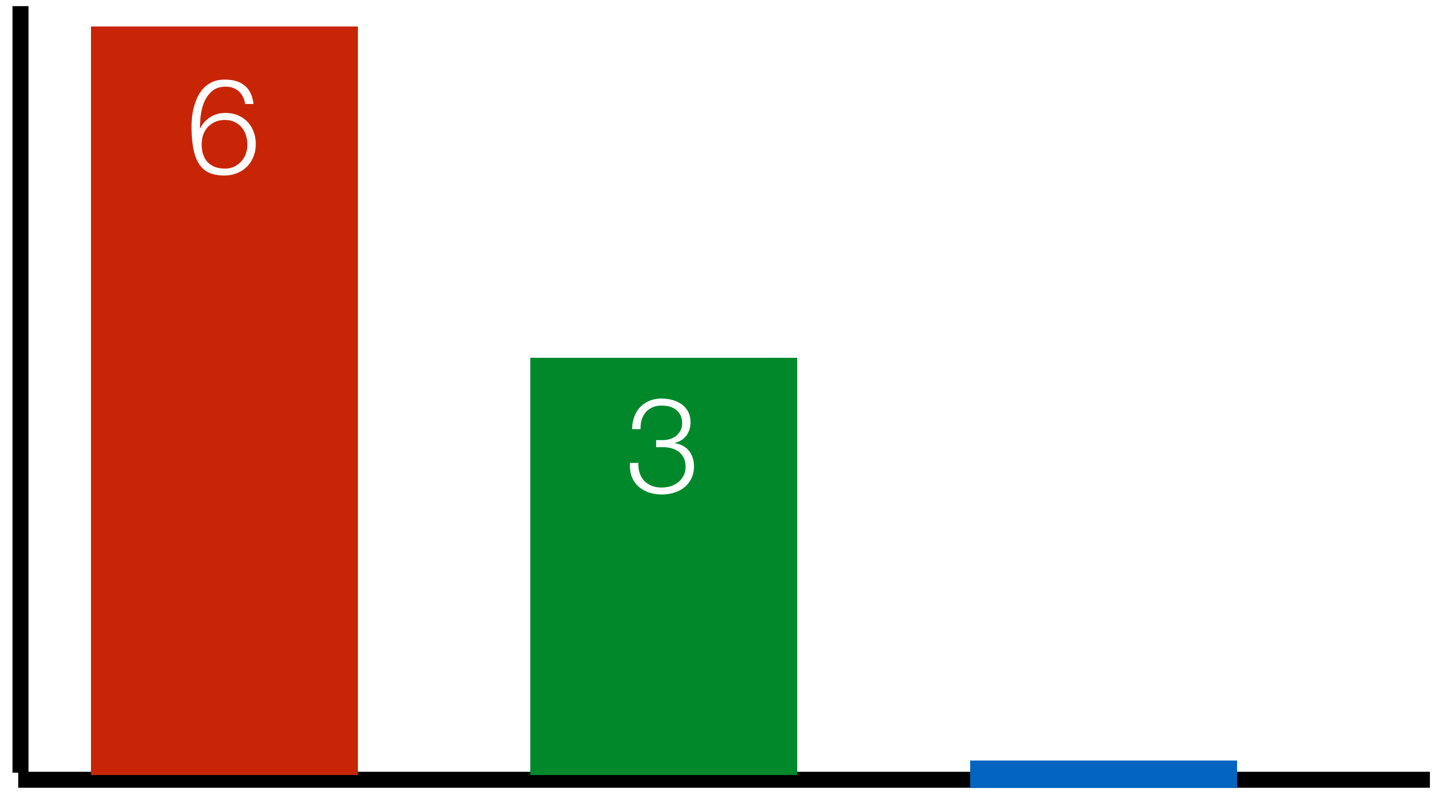


Bag of Word

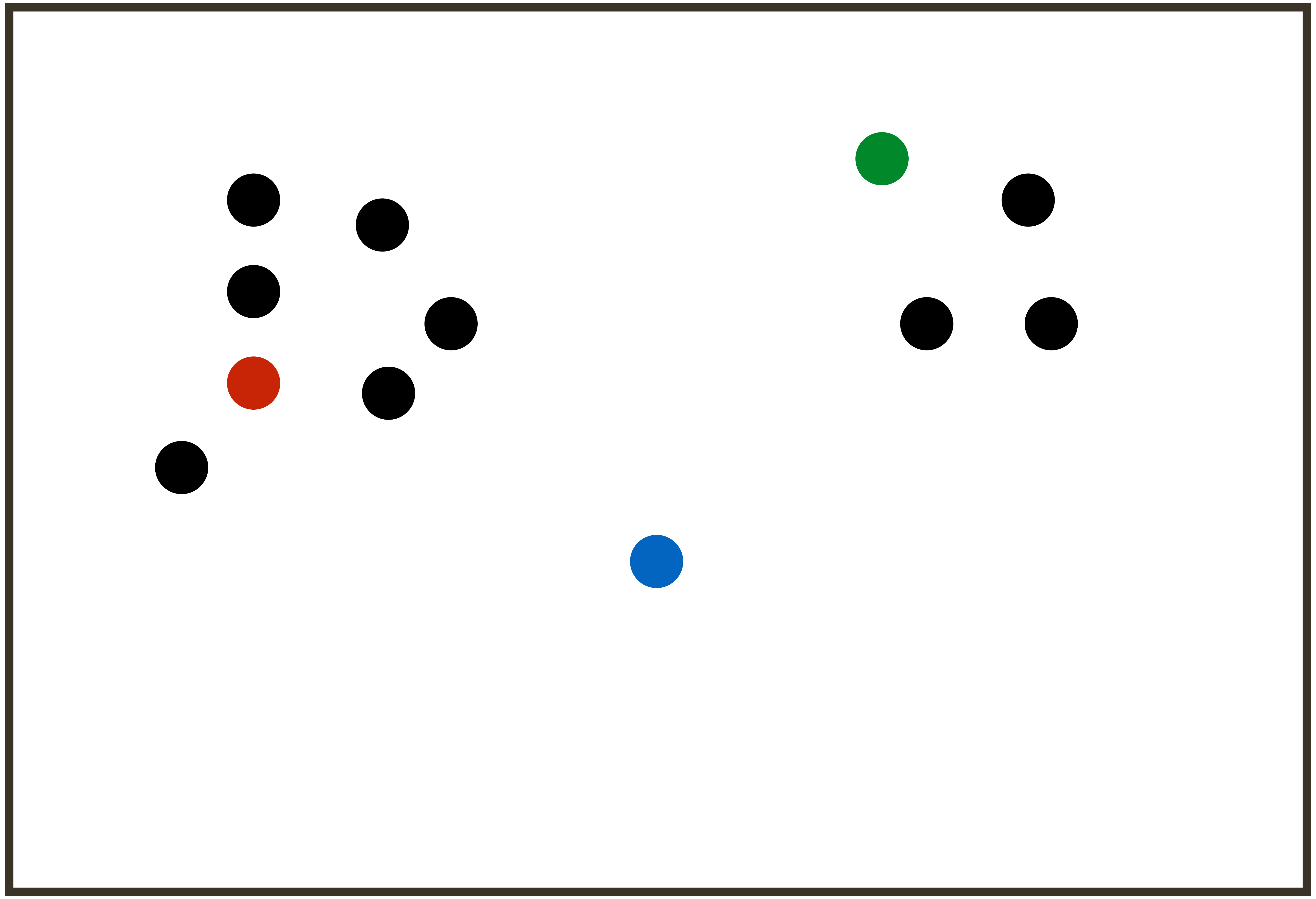
Example: VLAD

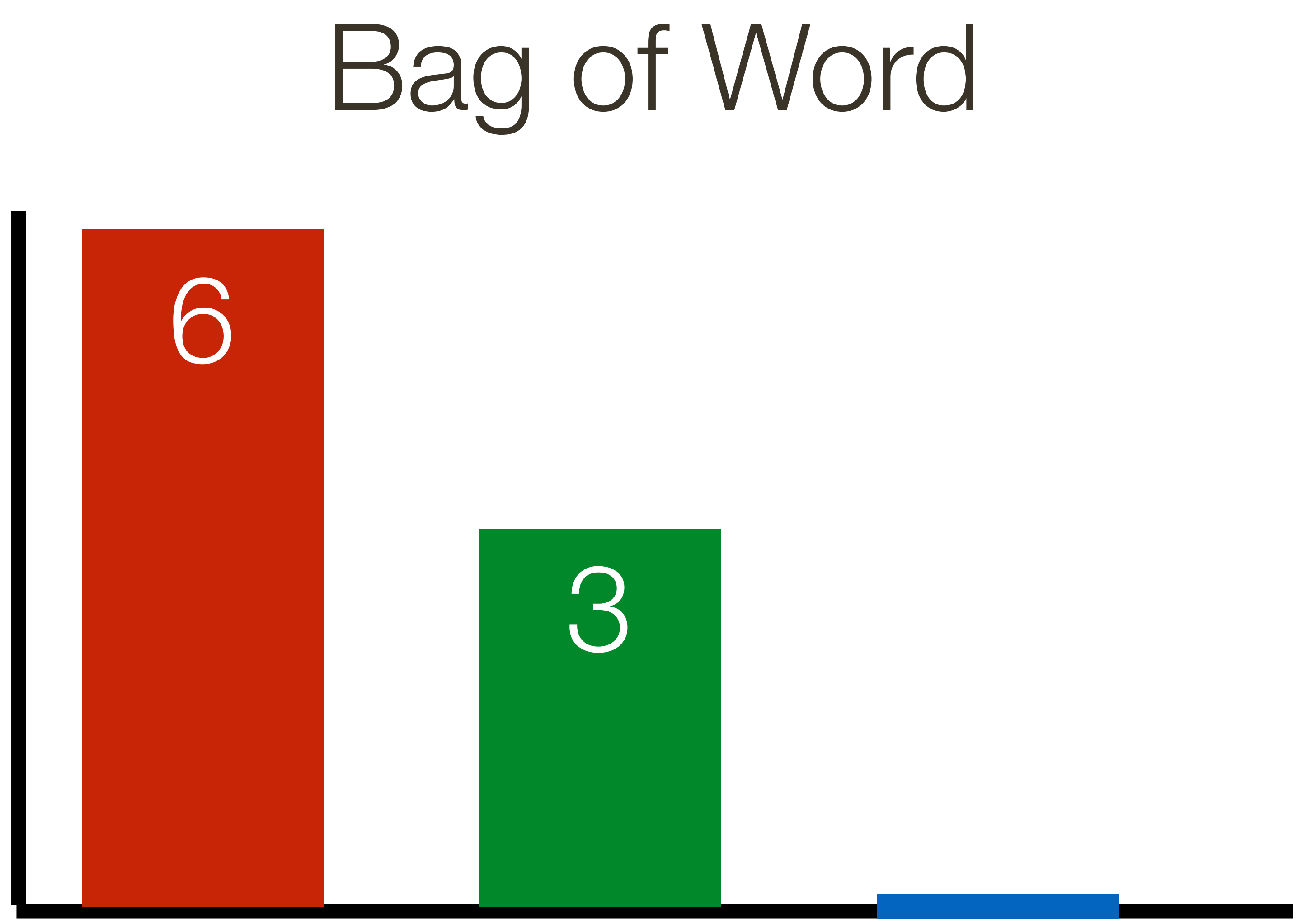


Bag of Word

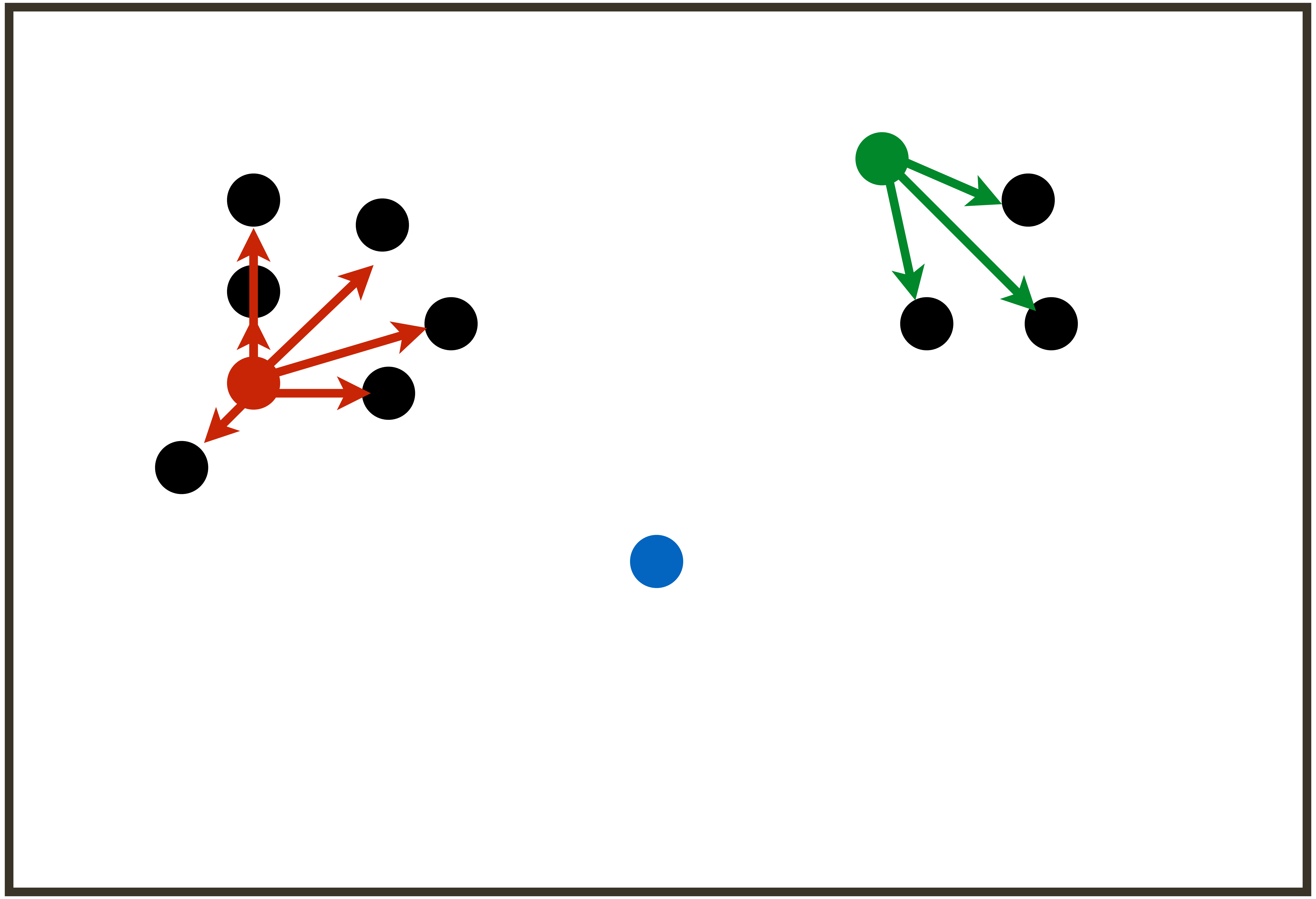


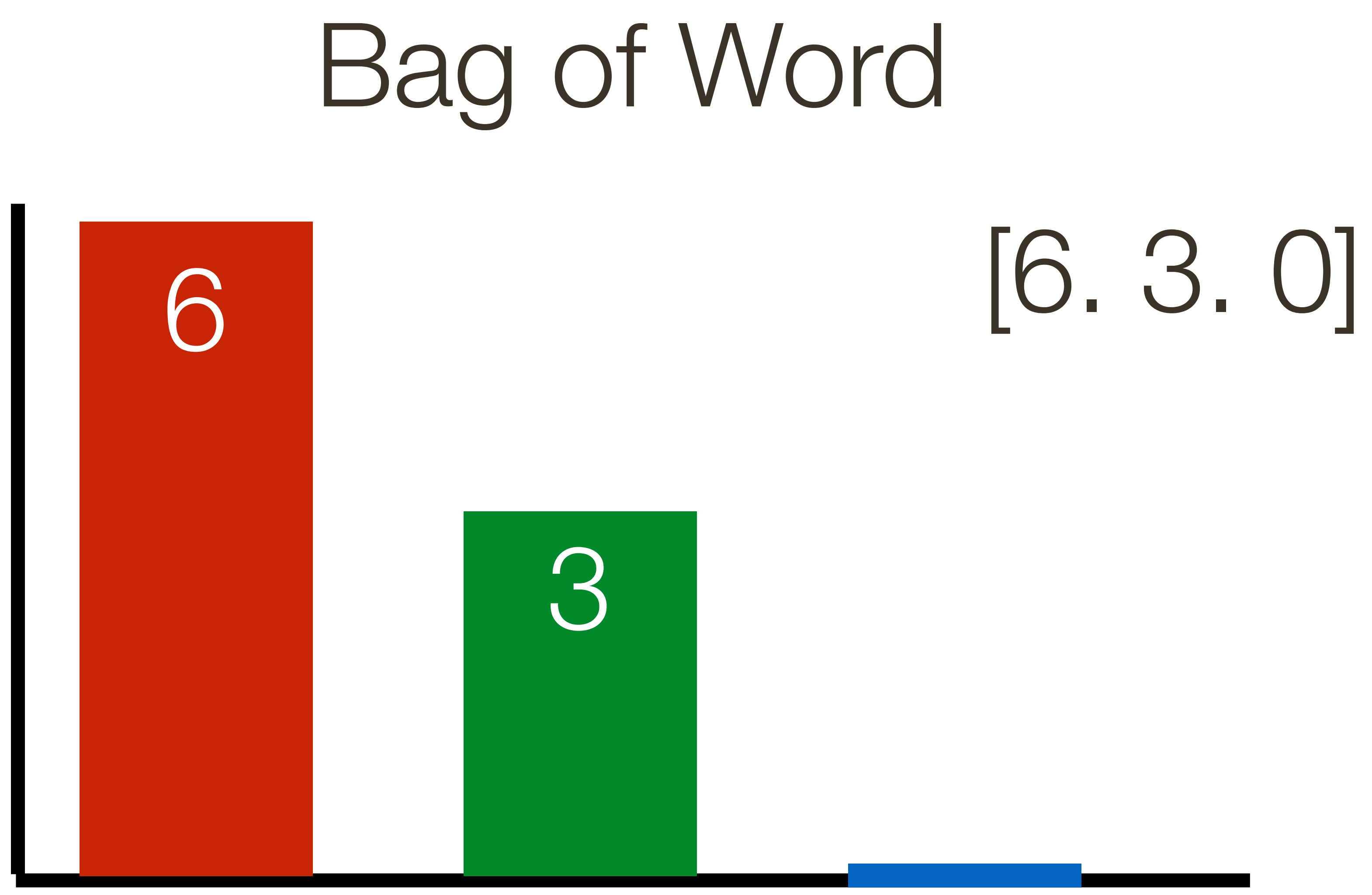
Example: VLAD

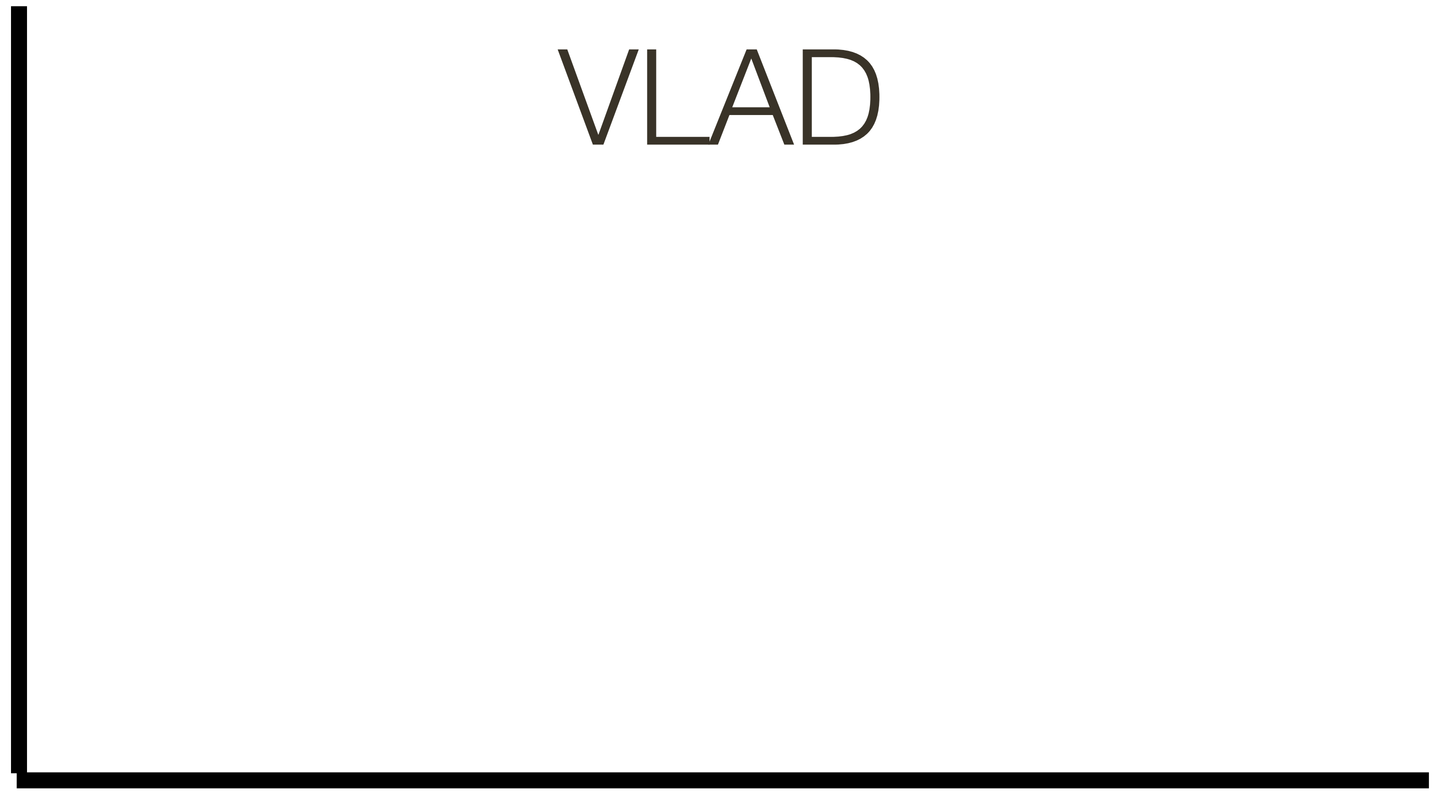




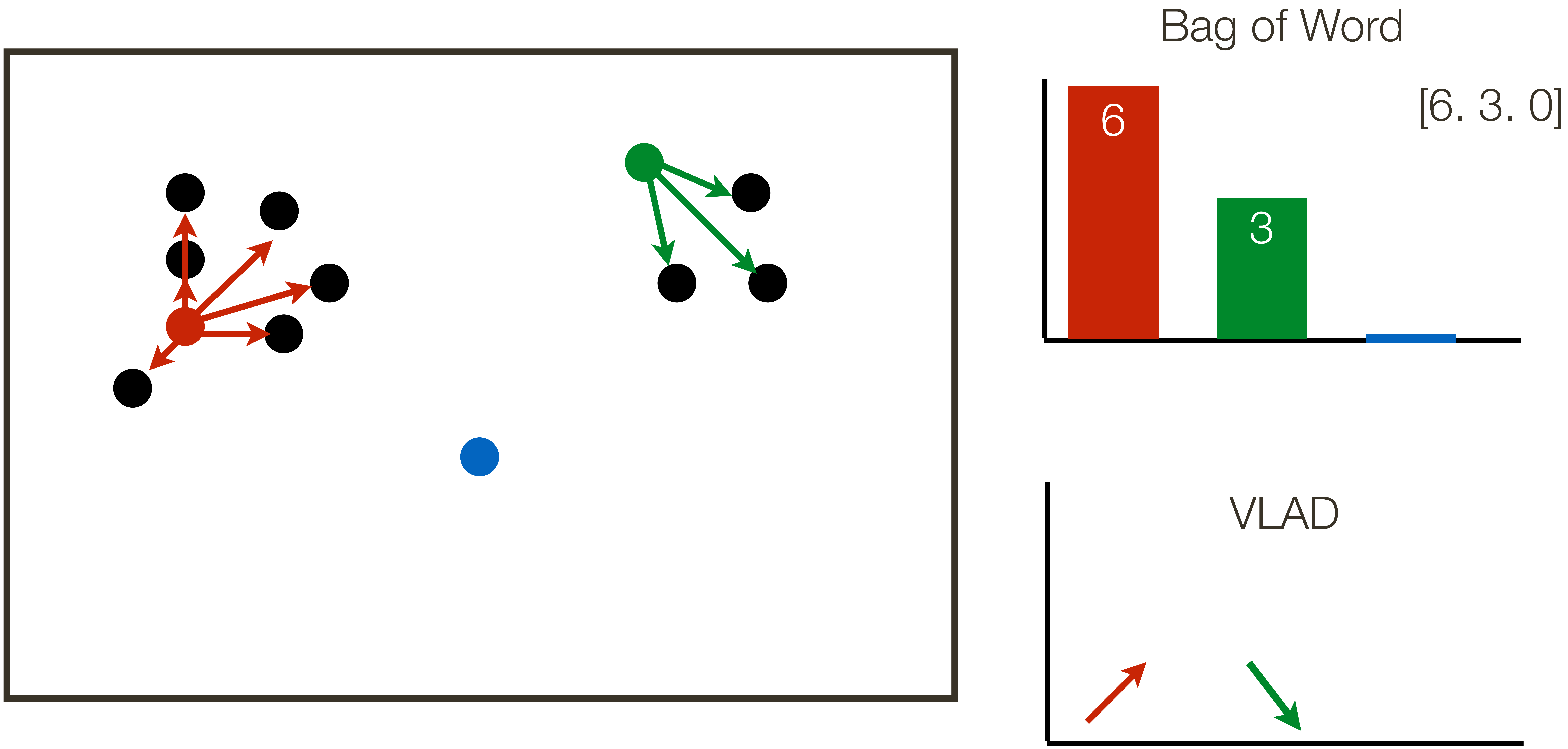
Example: VLAD







Example: VLAD



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 —> 2023

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

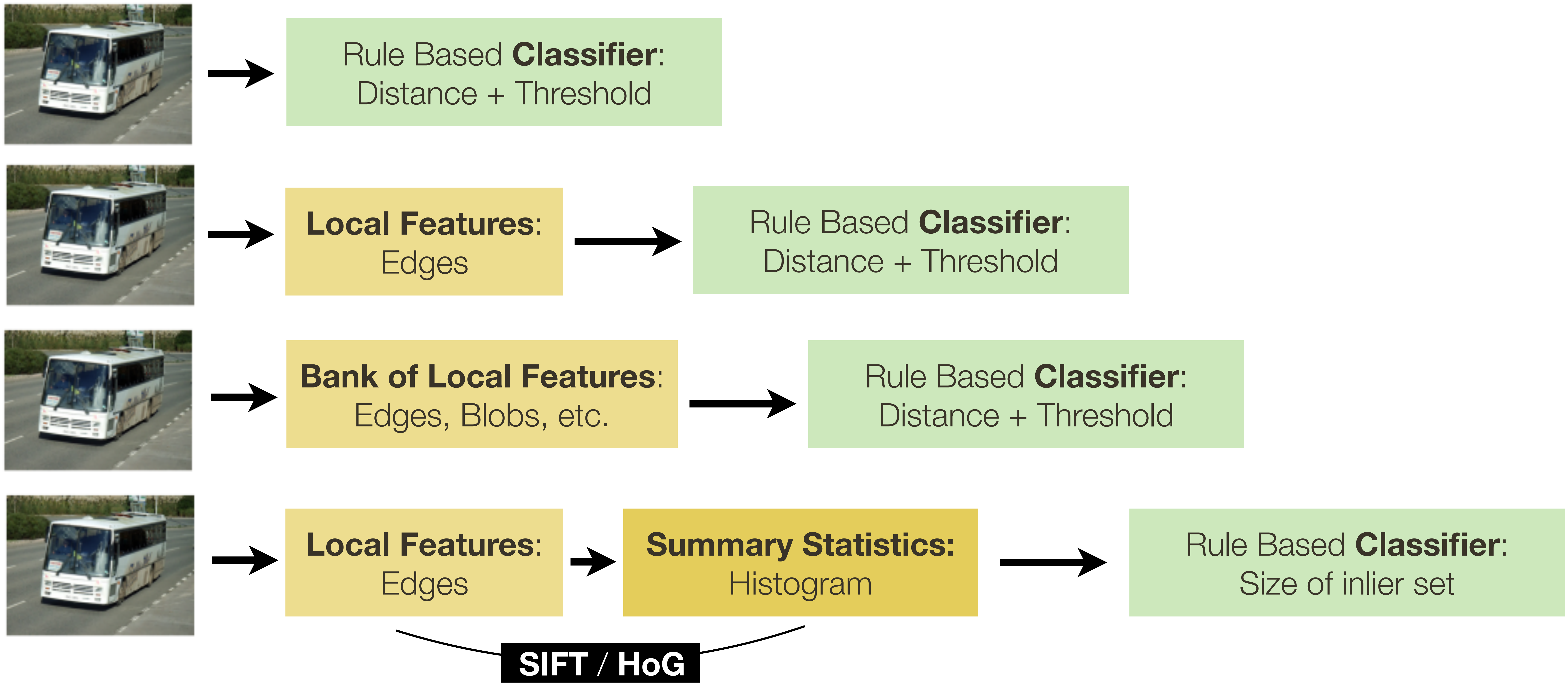
There is nothing really to "**learn**" (no need for training data), just measure similarity using favorite distance and choose threshold based on validation set

→ **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

→ **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

Recognition Overview: Learning

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

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

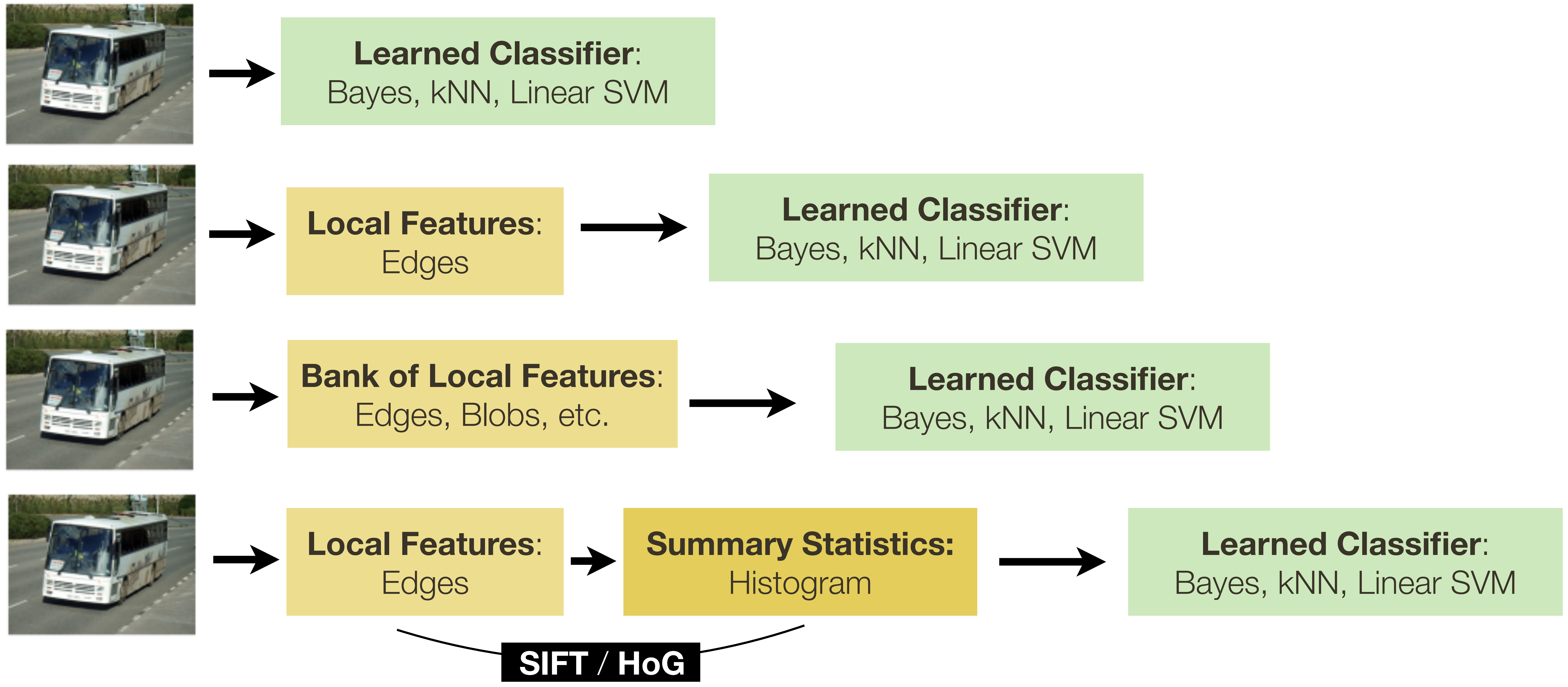
→ **Bank of Local Features**: Edges, Blobs, etc. → **Learned Classifier**: Bayes, kNN, Linear SVM

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

$\underbrace{\text{Local Features: Edges} \rightarrow \text{Summary Statistics: Histogram}}_{\text{SIFT / HoG}}$

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

Recognition Overview: Learning

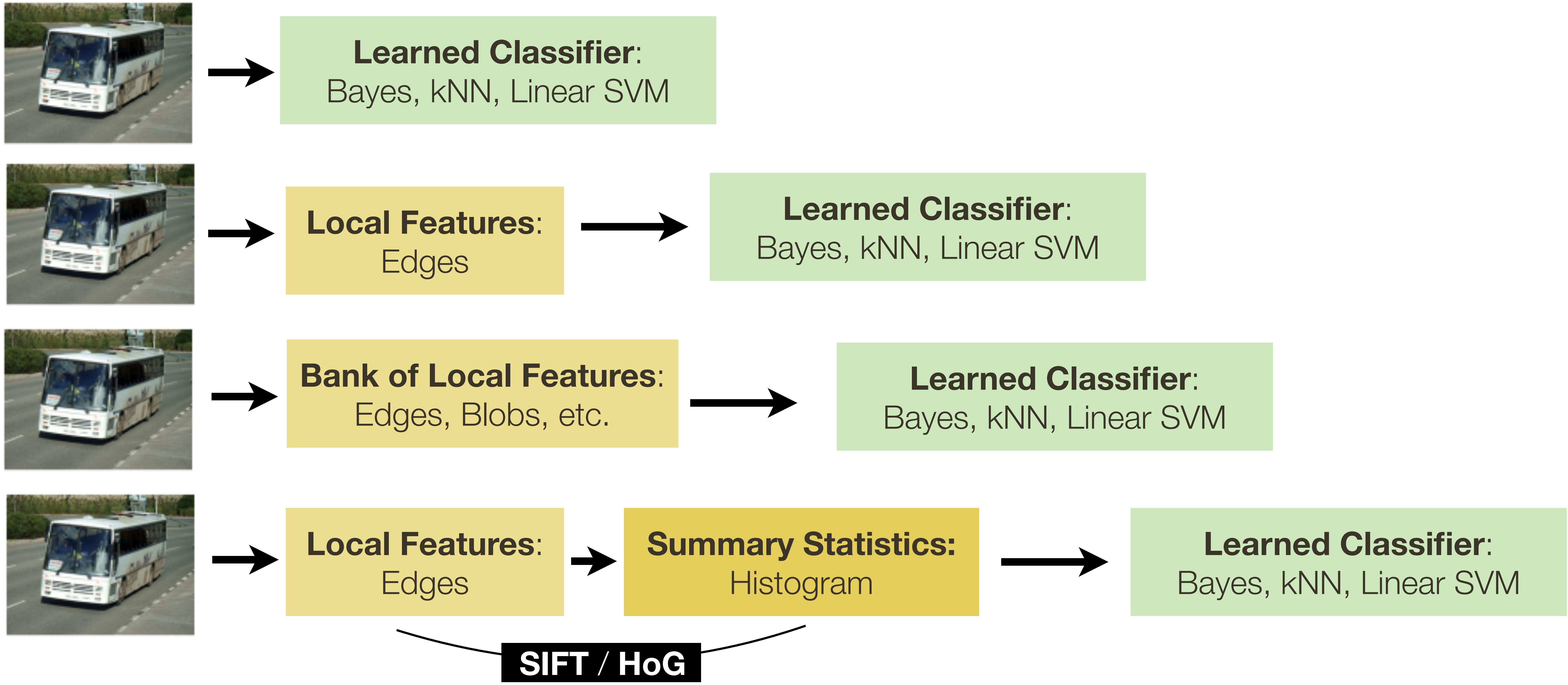


Bayes — estimate *parametric* form of distribution (requires training data) for each class

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

More expressive

Recognition Overview: Learning



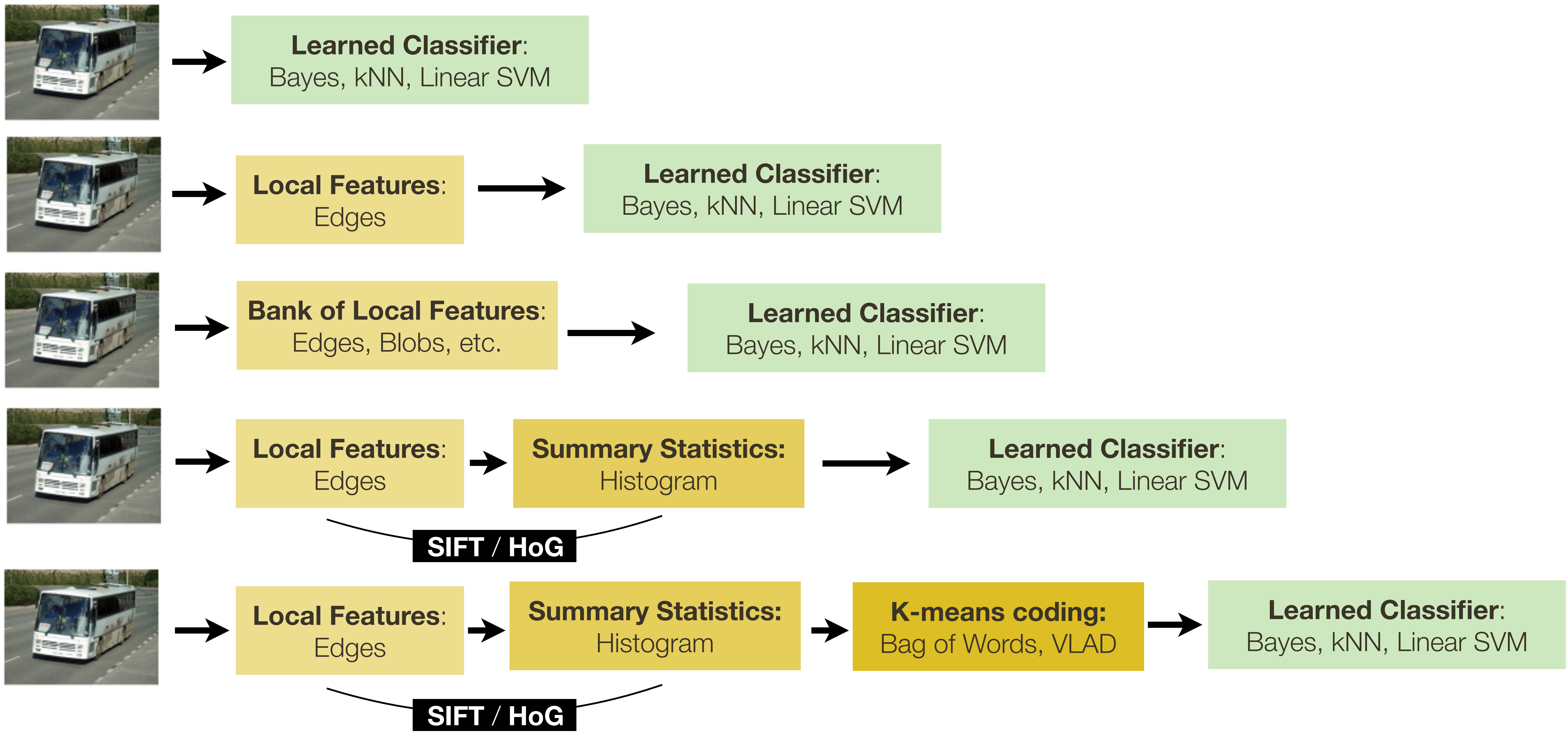
Bayes — estimate *parametric* form of distribution (requires training data) for each class

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



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

$\underbrace{\text{Local Features} \rightarrow \text{Summary Statistics}}_{\textbf{SIFT / HoG}}$

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

$\underbrace{\text{Local Features} \rightarrow \text{Summary Statistics}}_{\textbf{SIFT / HoG}}$

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

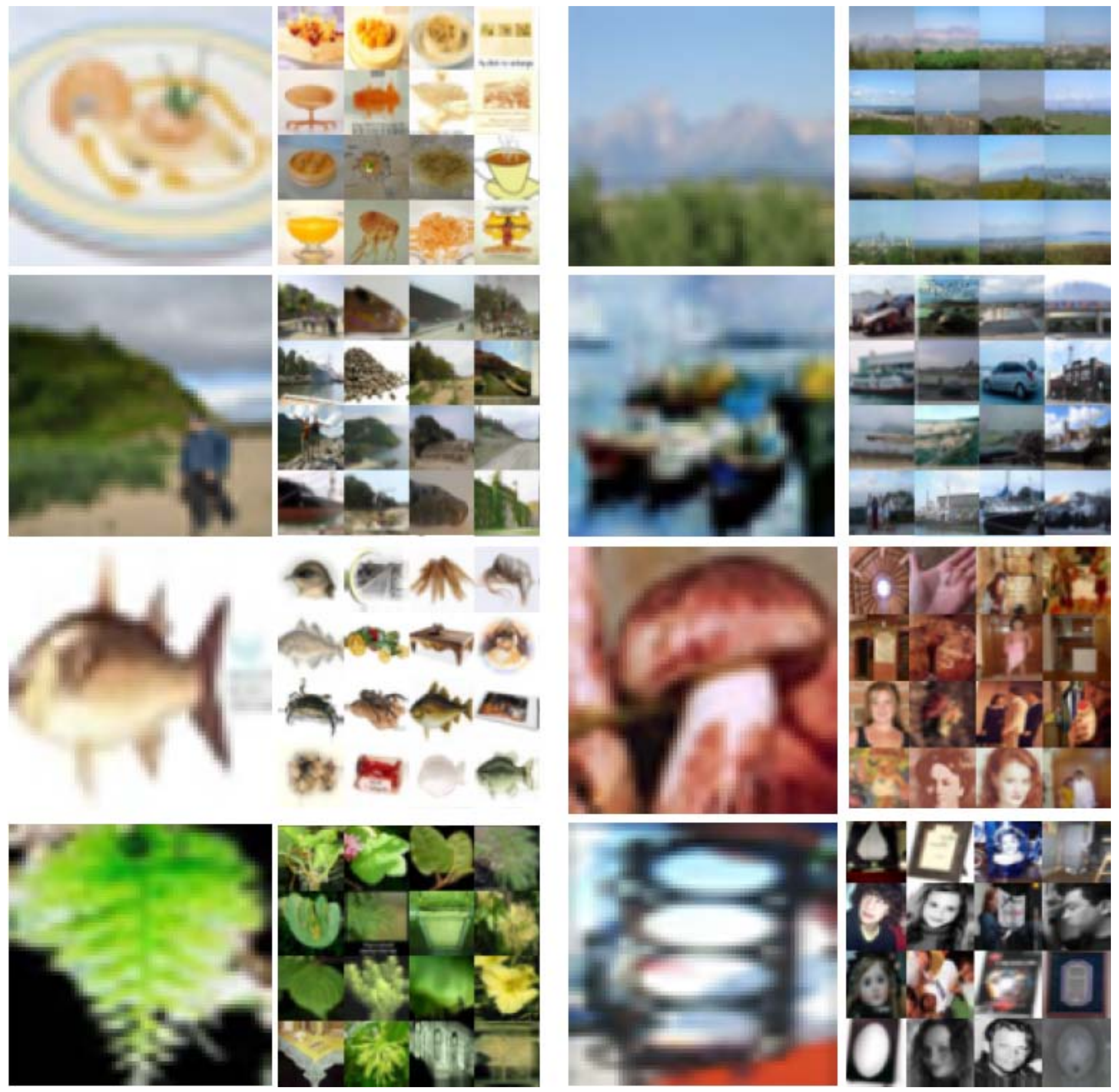
 → $\underbrace{\textbf{Local Features: } \text{Edges} \rightarrow \textbf{Summary Statistics: } \text{Histogram}}_{\textbf{SIFT / HoG}}$ → **K-means coding:** Bag of Words, VLAD → **Learned Classifier**: Bayes, kNN, Linear SVM

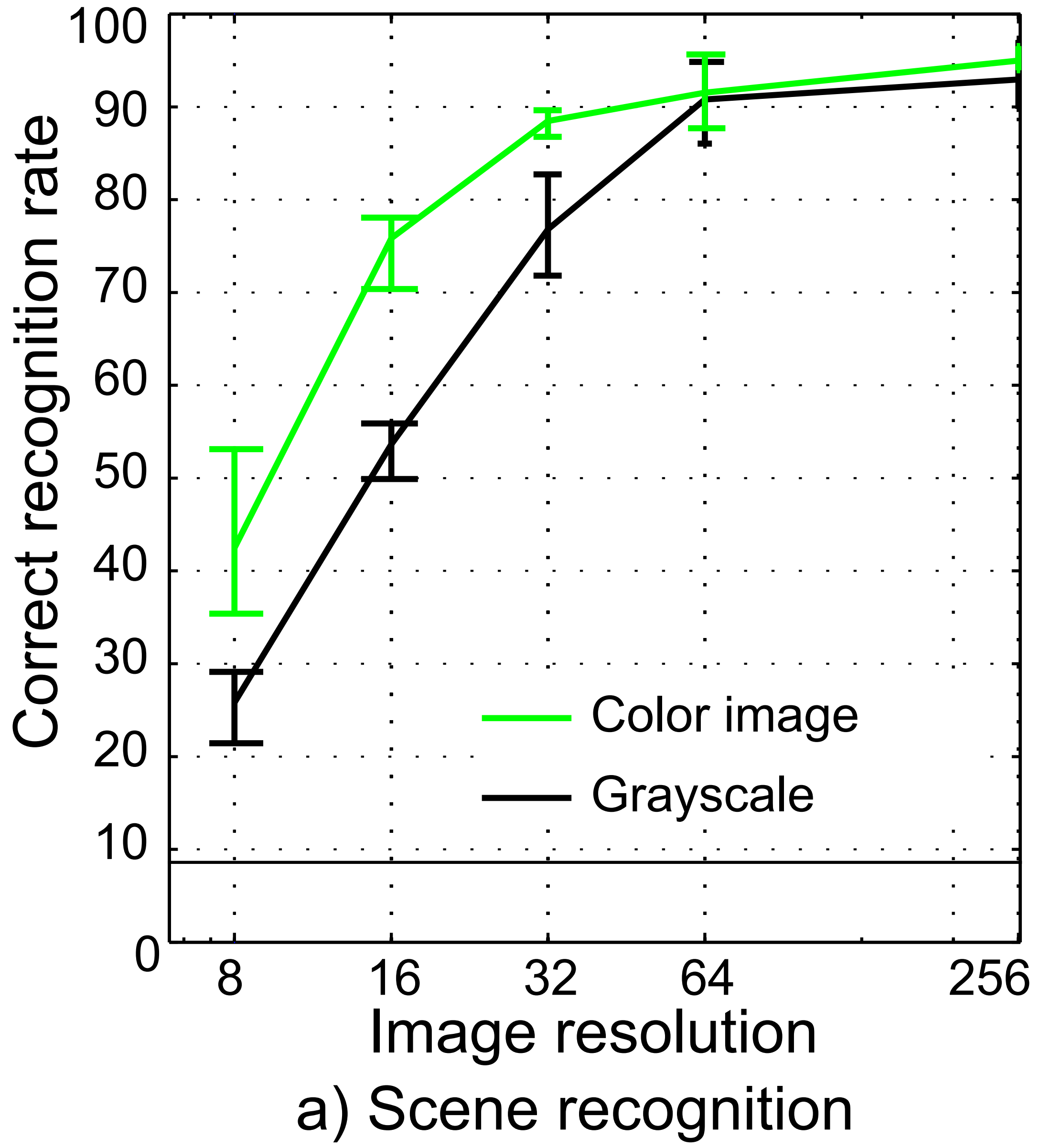
Tiny Image Dataset

— 80 million images collected via image search using 75,062 noun synsets from WordNet (labels are noisy)

— Very small images (32x32xRGB) used to minimise storage

— Note human performance is still quite good at this scale!



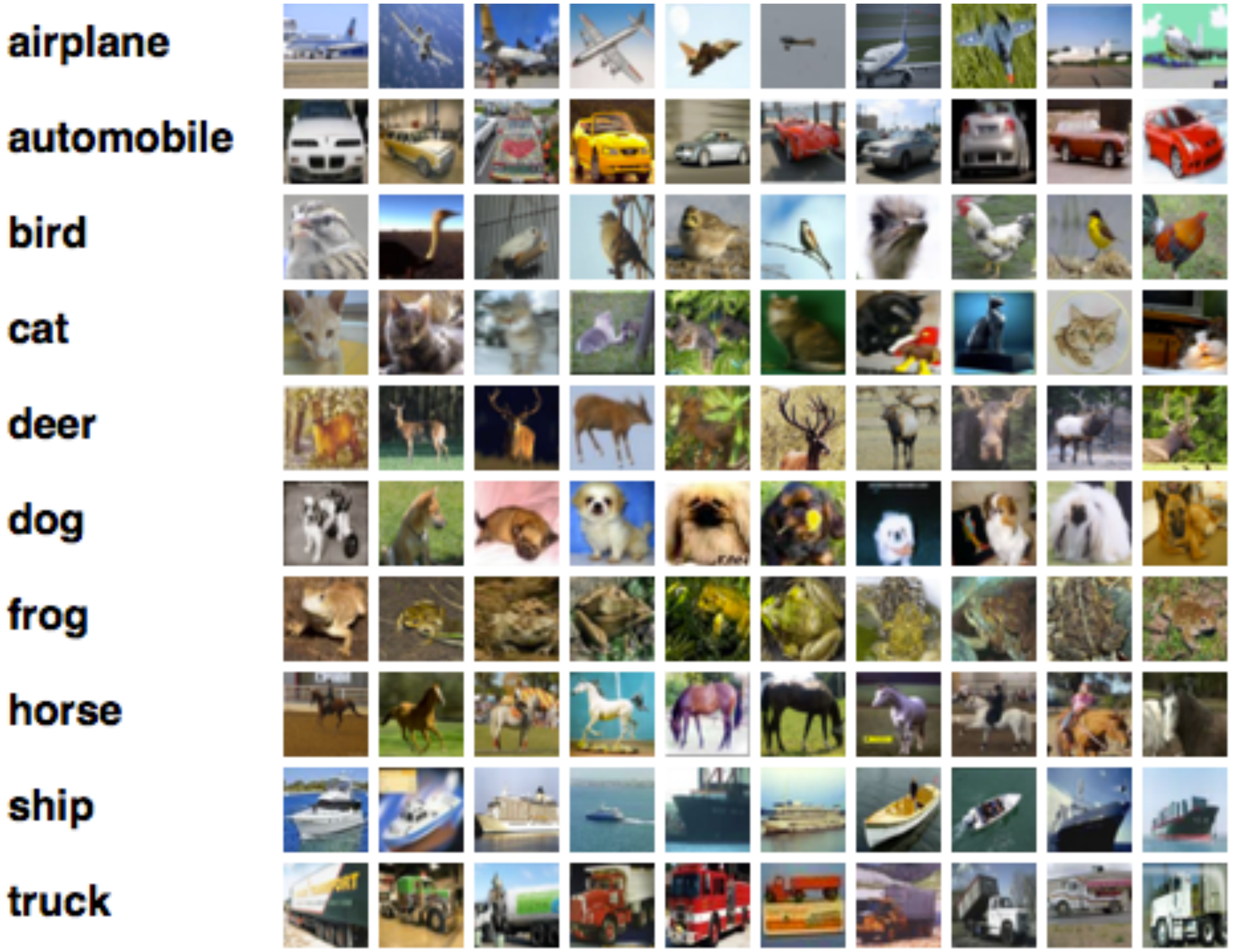


[Torralba Freeman Fergus 2008]

CIFAR10 Dataset

— Hand labelled set of 10 categories from Tiny Images dataset

— 60,000 32x32 images in 10 classes (50k train, 10k test)

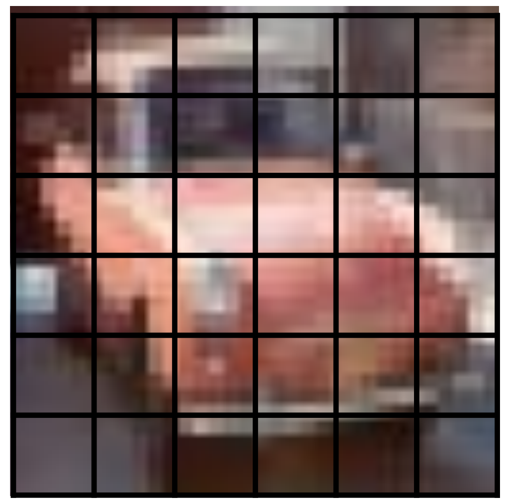


Good test set for visual recognition problems

CIFAR10 Classification

Let's build an image classifier

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



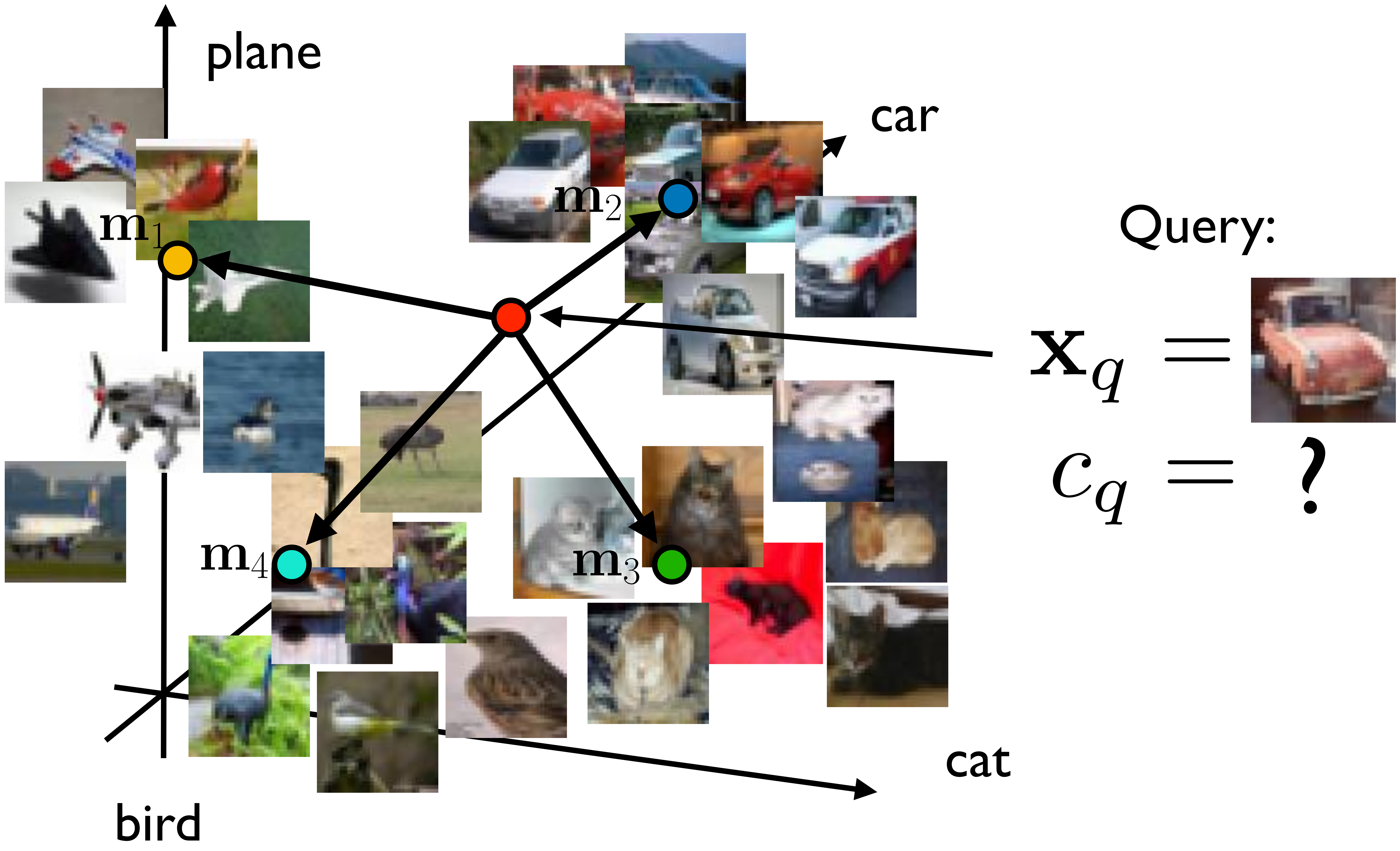
32 x 32 x RGB (8 bit) image →

x = [65 102 33 57 54 …]

x = 3072 element vector of 0-255

Nearest Mean Classifier

Compute a single "average" template per class

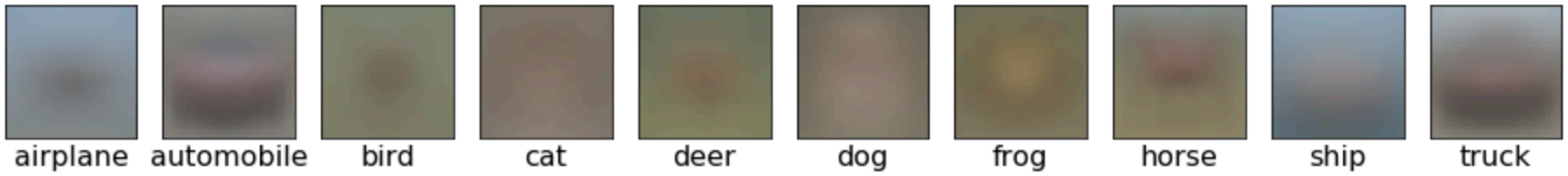


Nearest Mean Classifier

Find the nearest mean and assign class:

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

CIFAR10 class means:

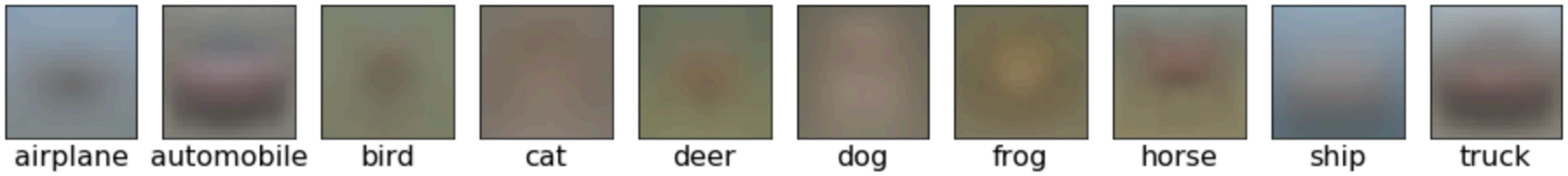


Nearest Mean Classifier

Find the nearest mean and assign class:

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

CIFAR10 class means:

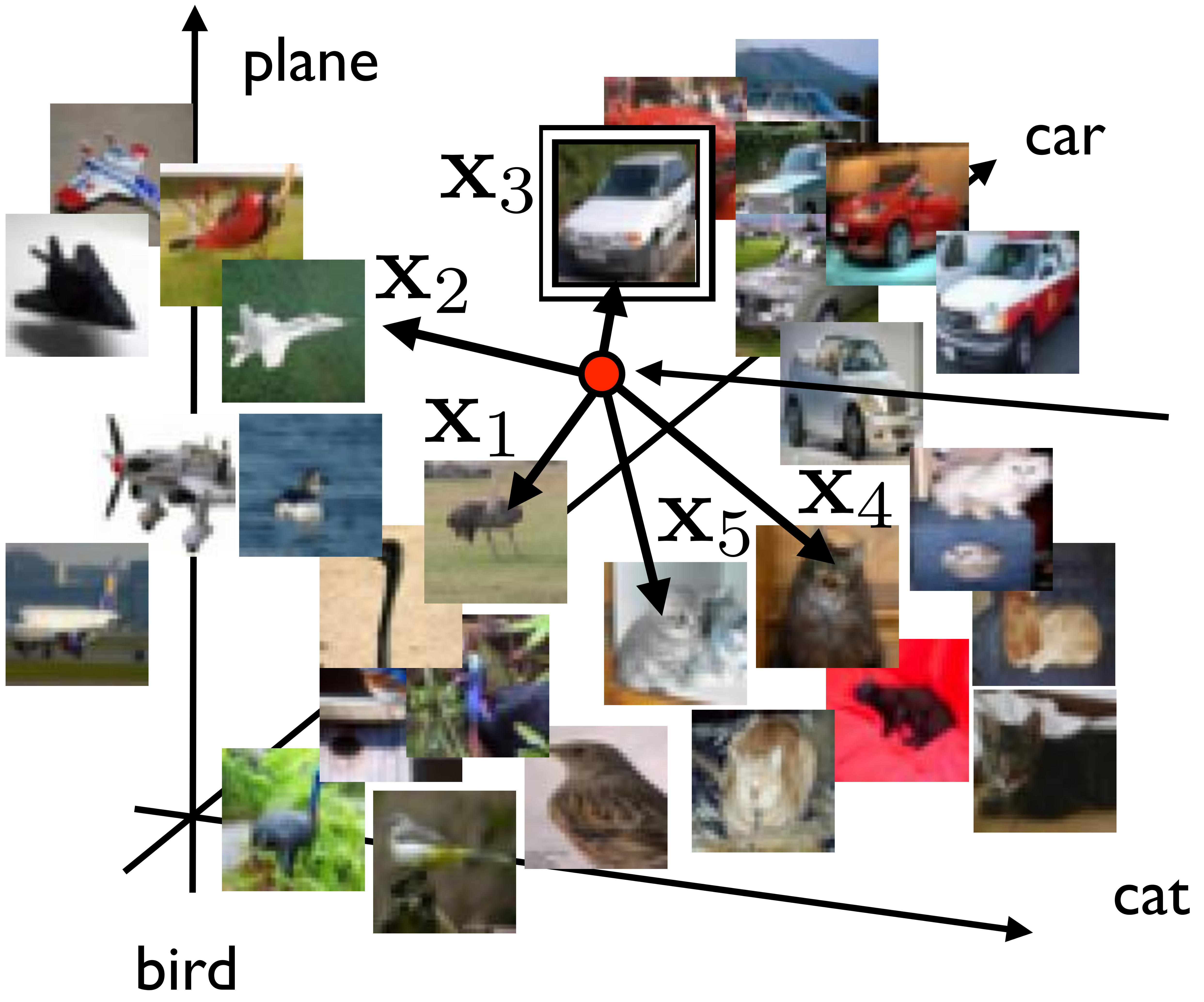


Performance:

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



Query:

$$\mathbf{x}_q = $$

$$c_q = ?$$

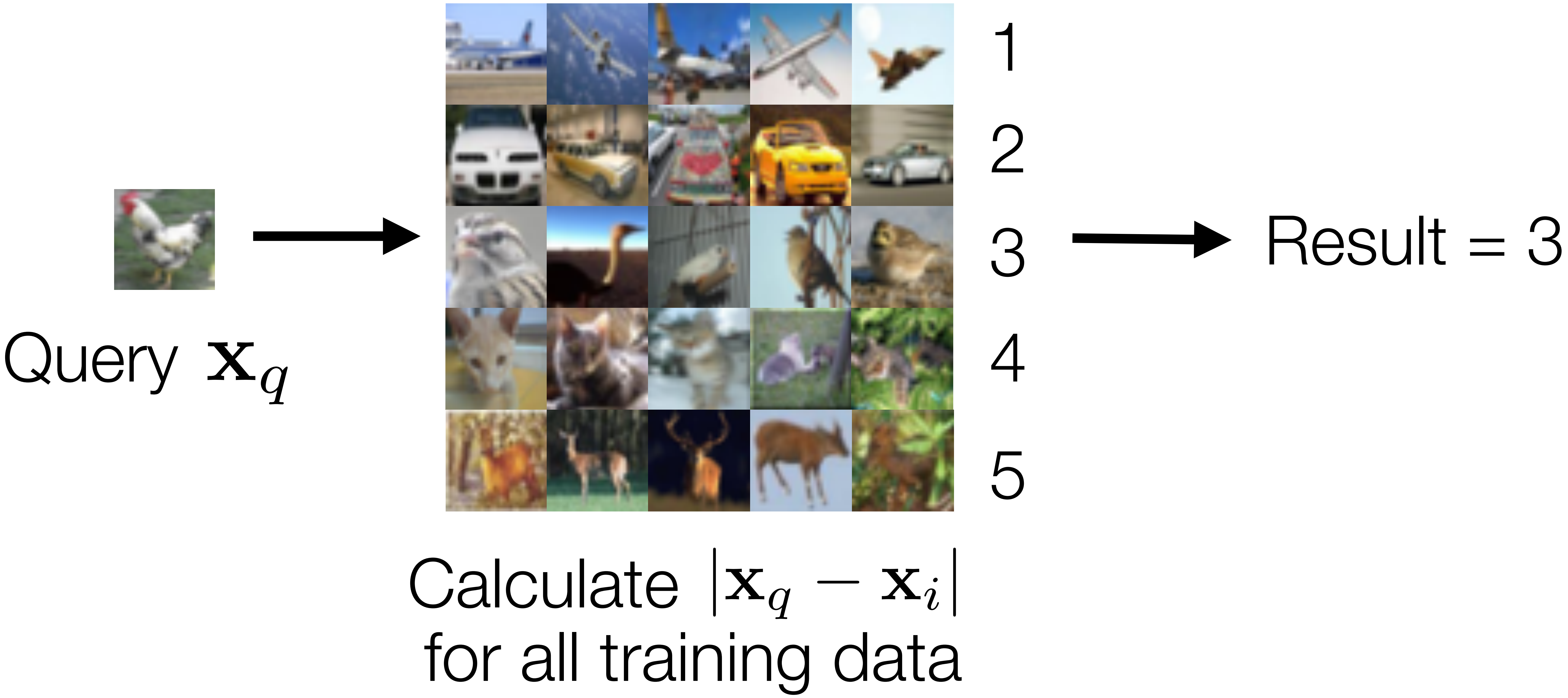
Nearest Neighbor Classifier

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:

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



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:

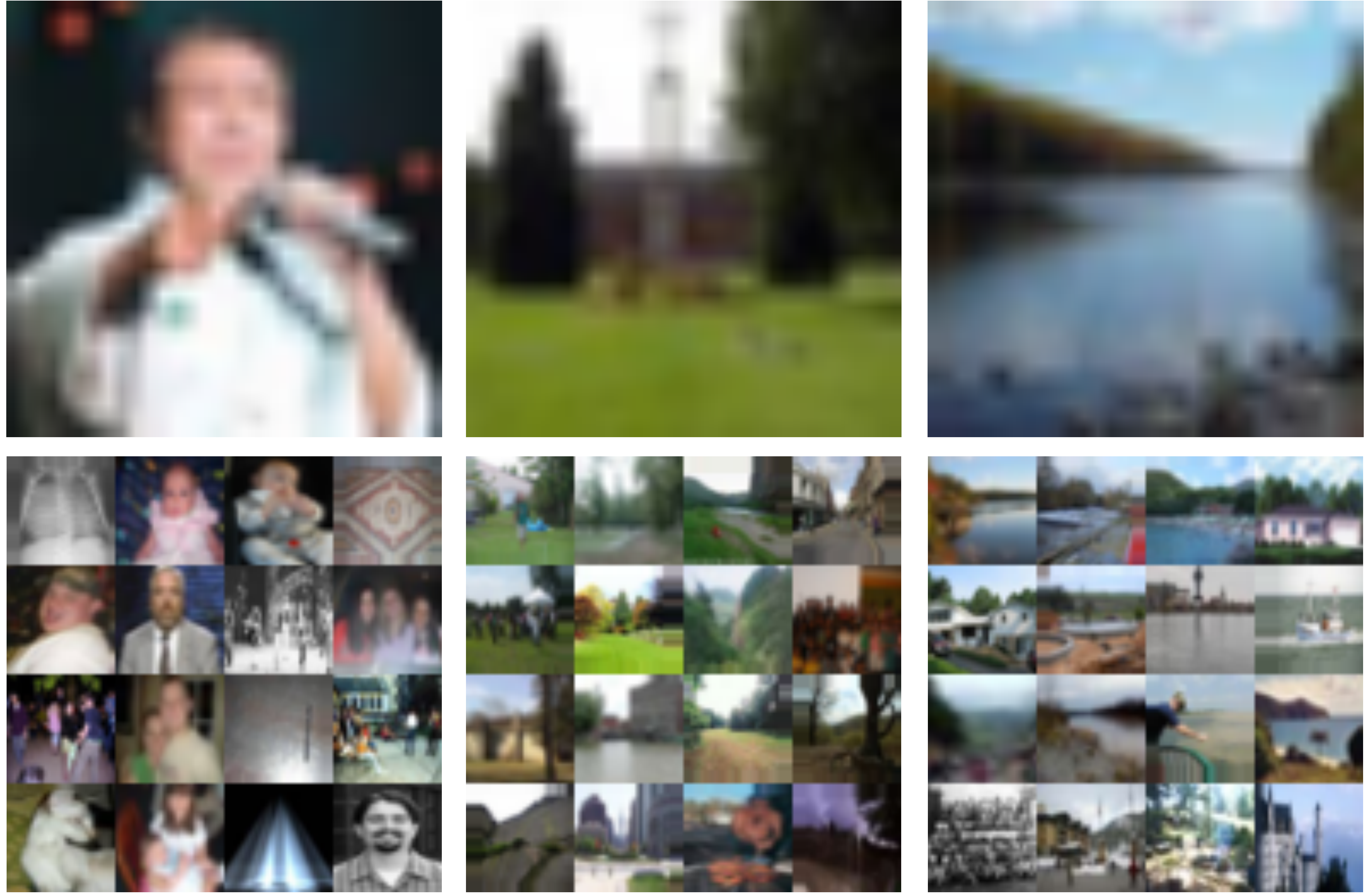
$$\hat{y}(\mathbf{x}_q) = y(\mathbf{x}_{i_{NN}})$$

Performance:

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

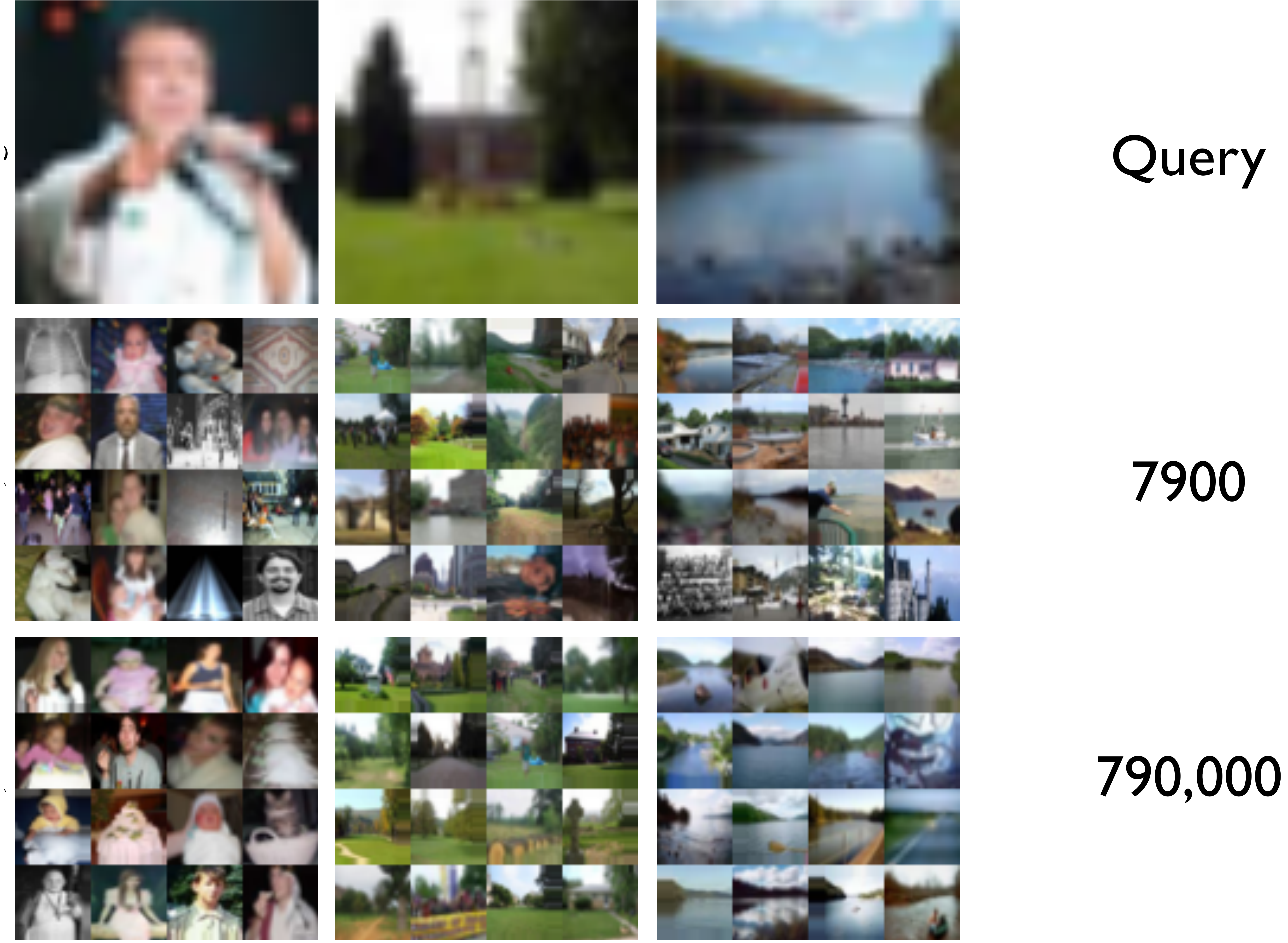
Source: https://cran.r-project.org/web/packages/KernelKnn/vignettes/image_classification_using_MNIST_CIFAR_data.html

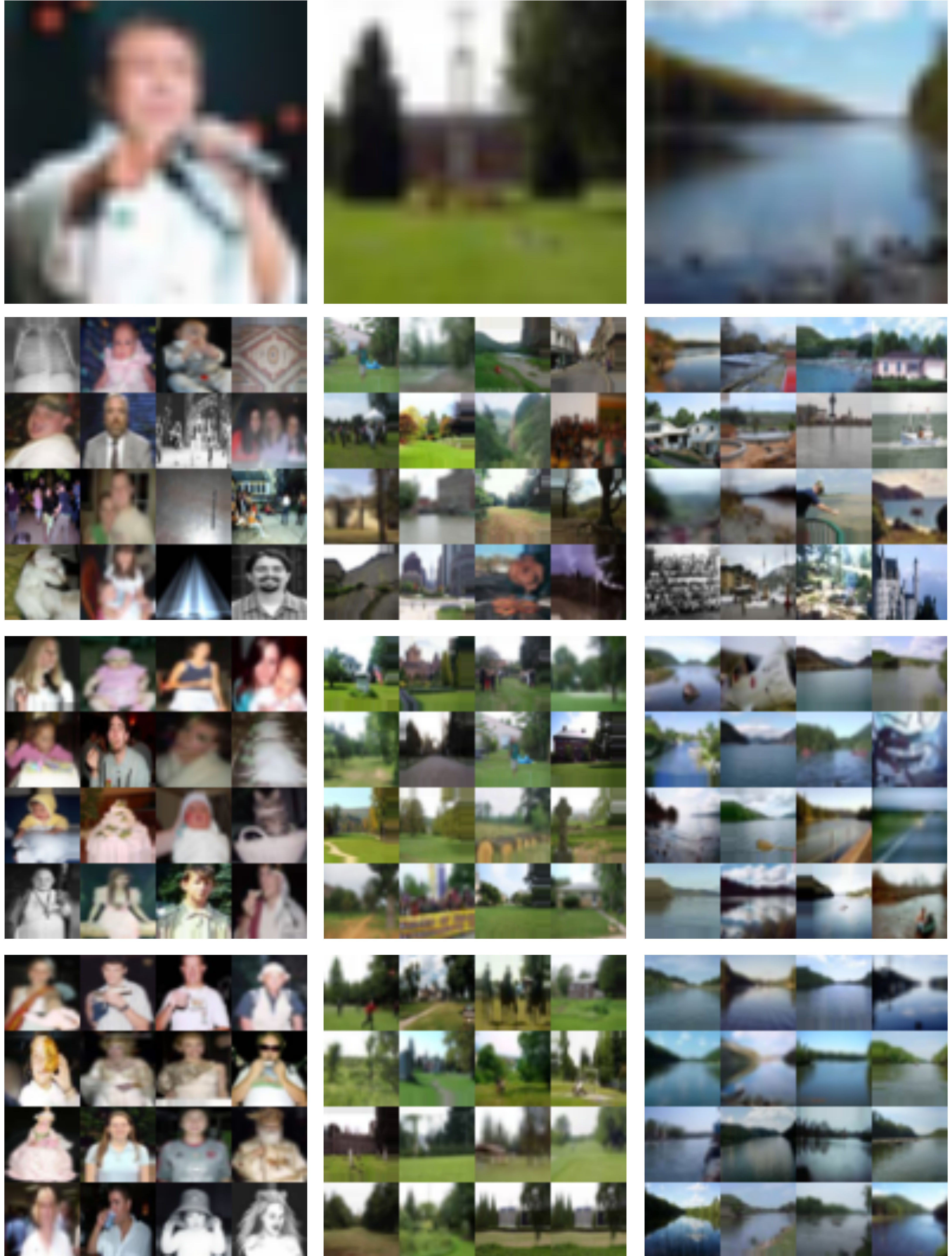
Query



Query

7900


Query
7900
790,000



Query

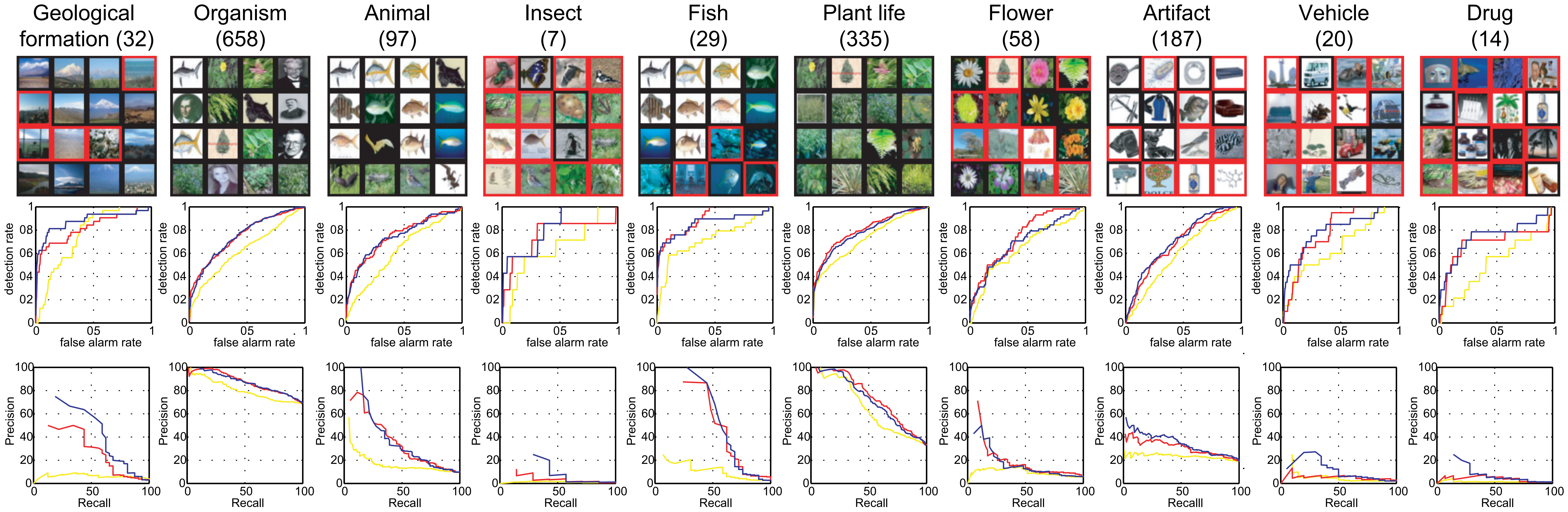
7900

790,000

79,000,000

Tiny Image Recognition

[Torralba, Fergus, Freeman '08]

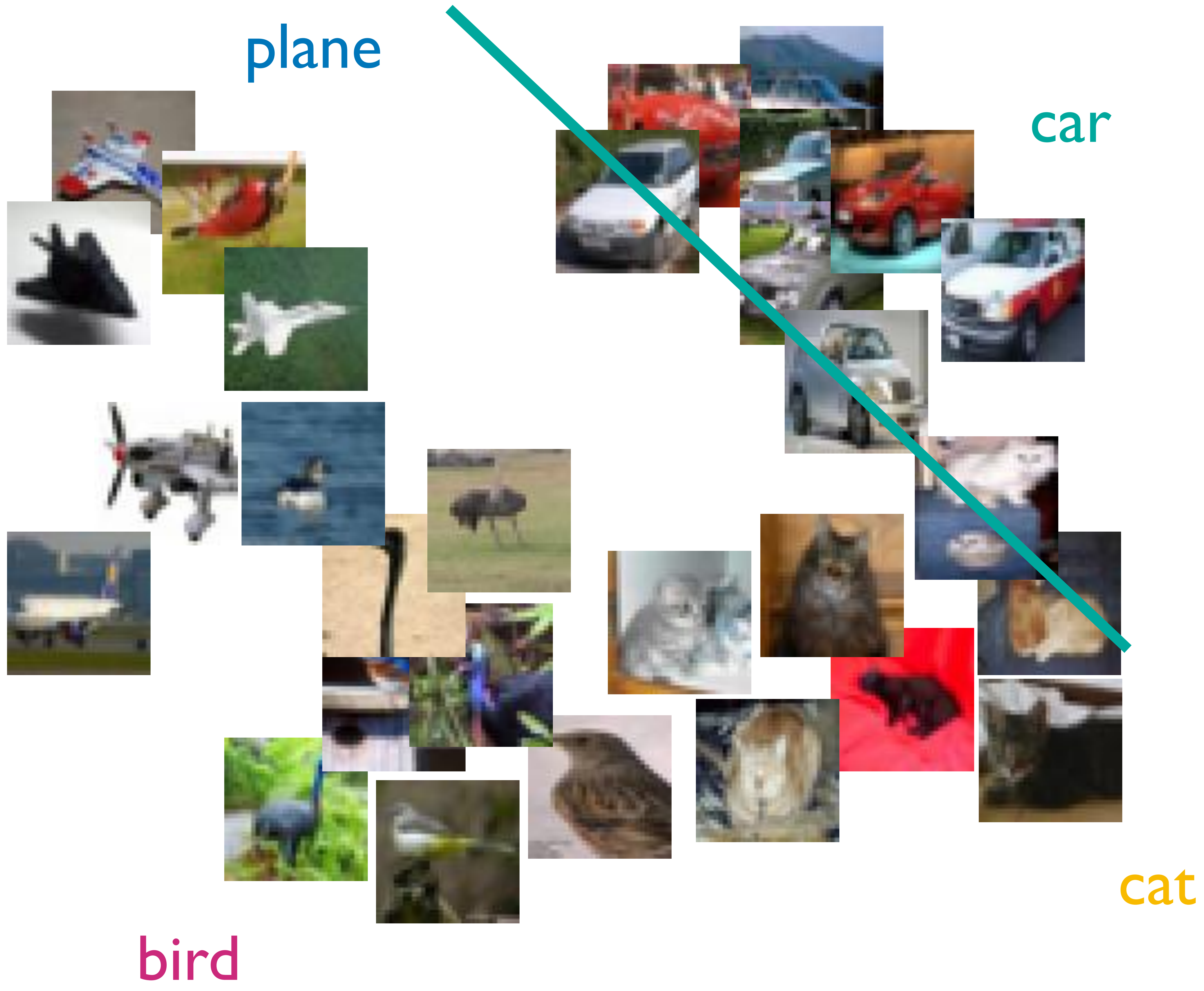


yellow = 7900, red = 790,000, blue = 79,000,000

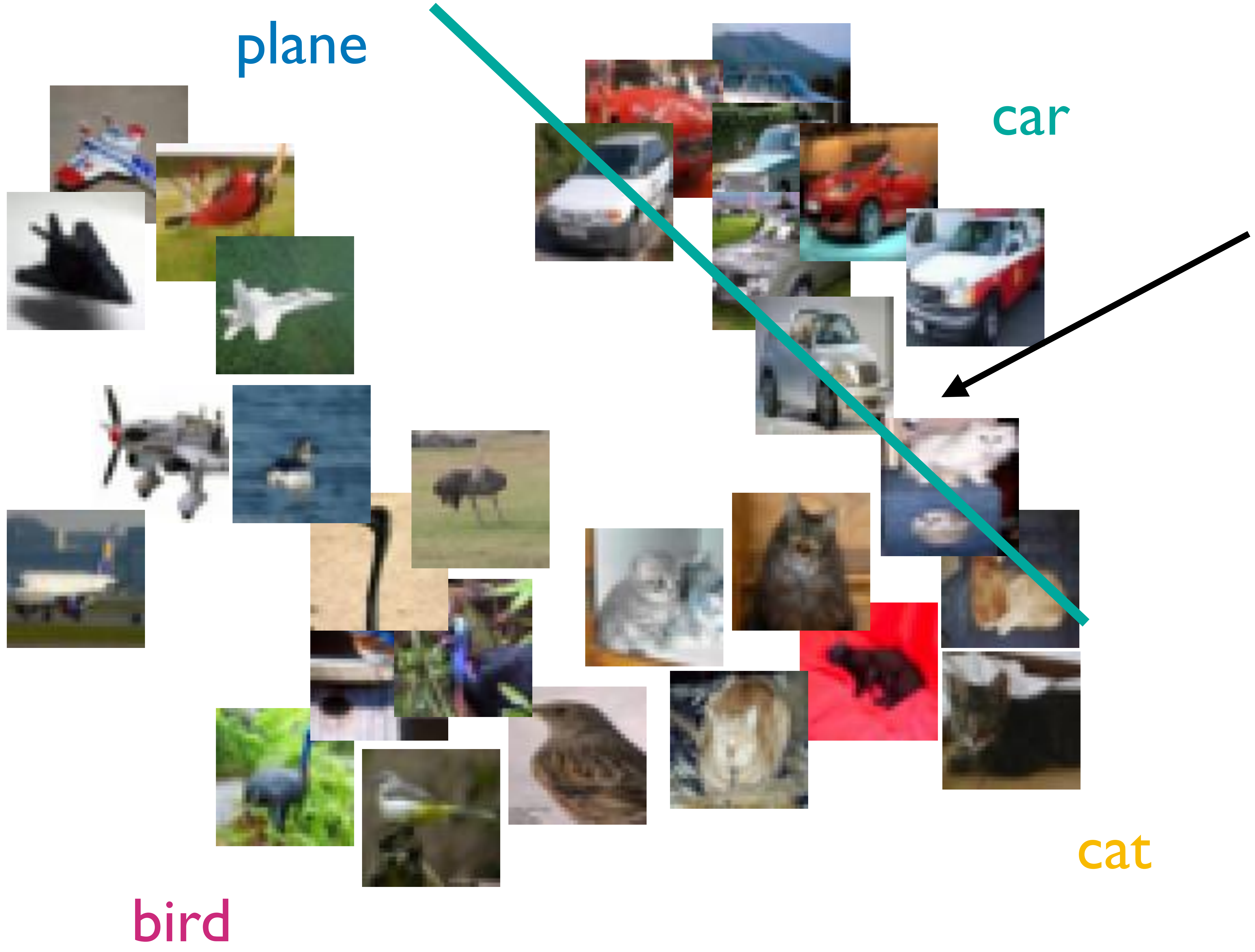
Nearest neighbour becomes increasingly accurate as N increases, but do we need to store a dataset of 80 million images?

1-vs-All **Linear SVM**

1-vs-All **Linear SVM**

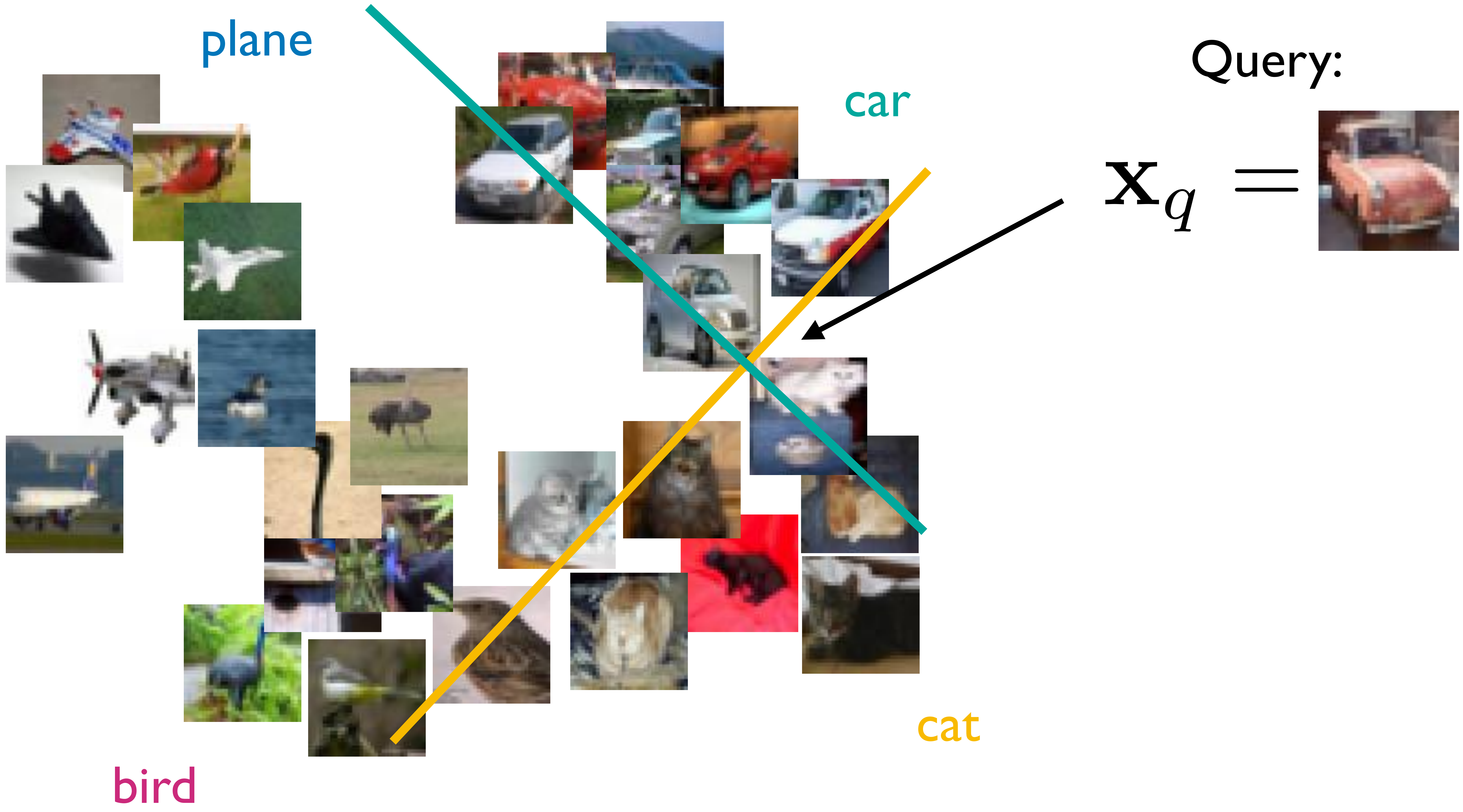


1-vs-All **Linear SVM**

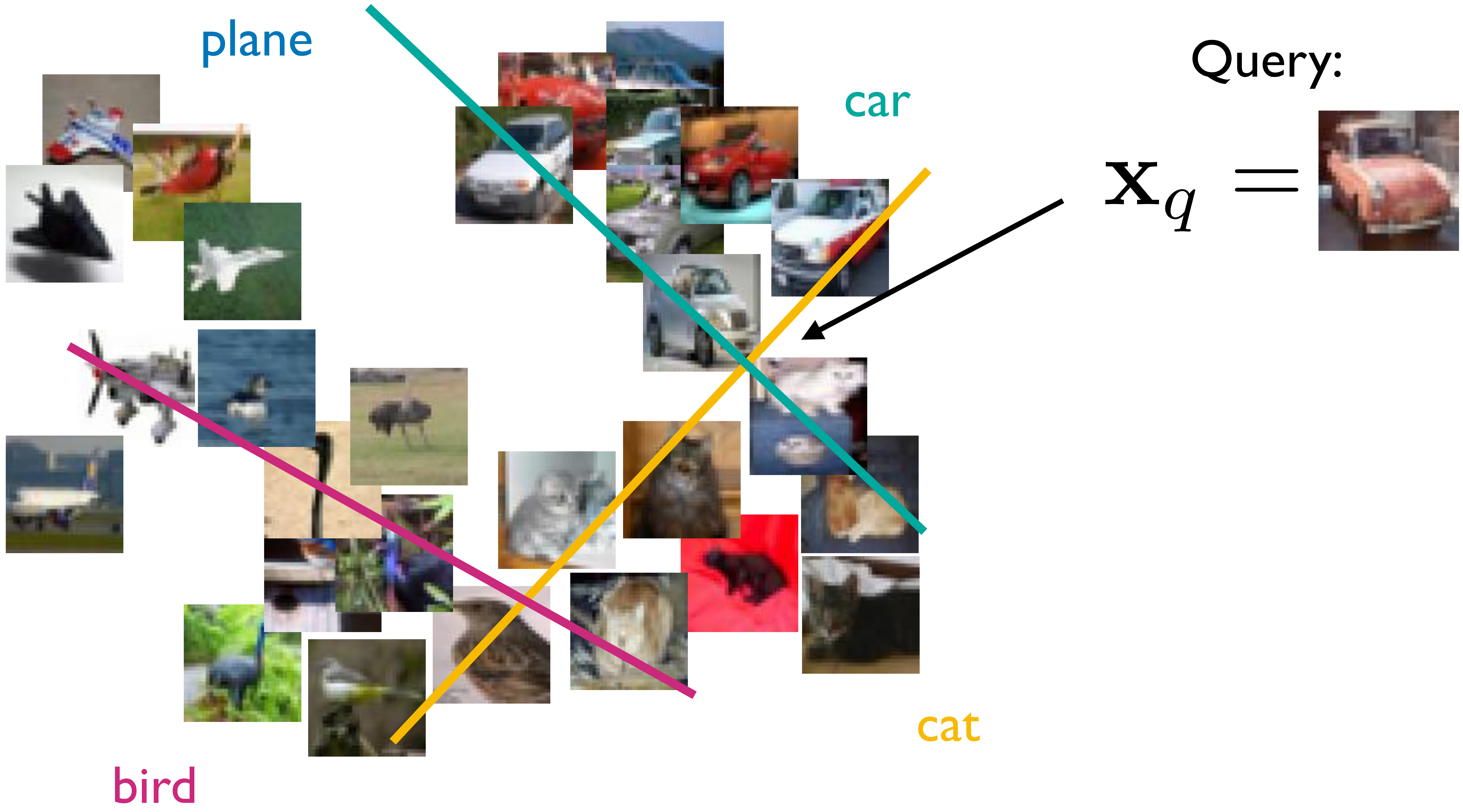


$$\mathbf{x}_q =$$

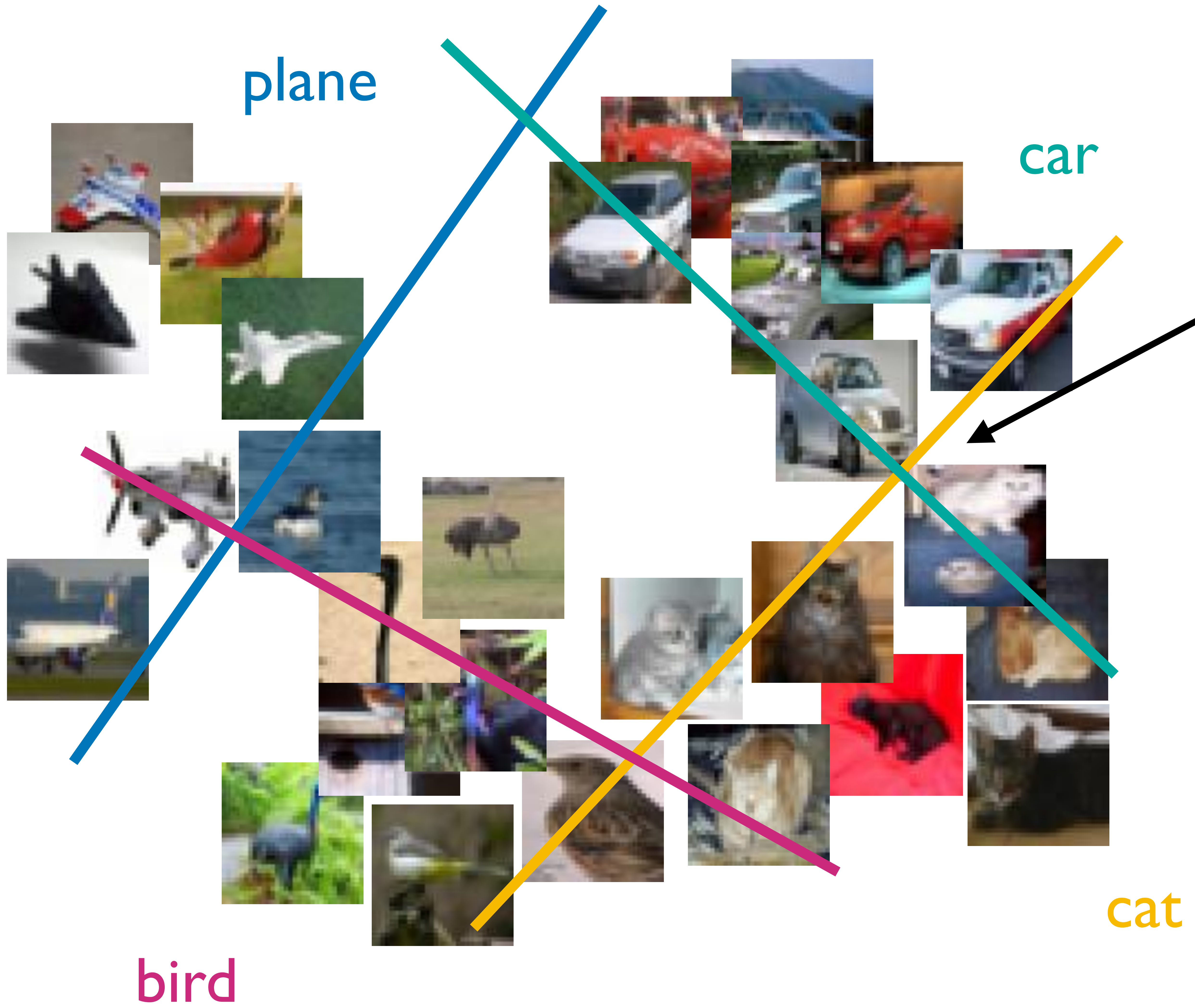
1-vs-All **Linear SVM**



1-vs-All **Linear SVM**



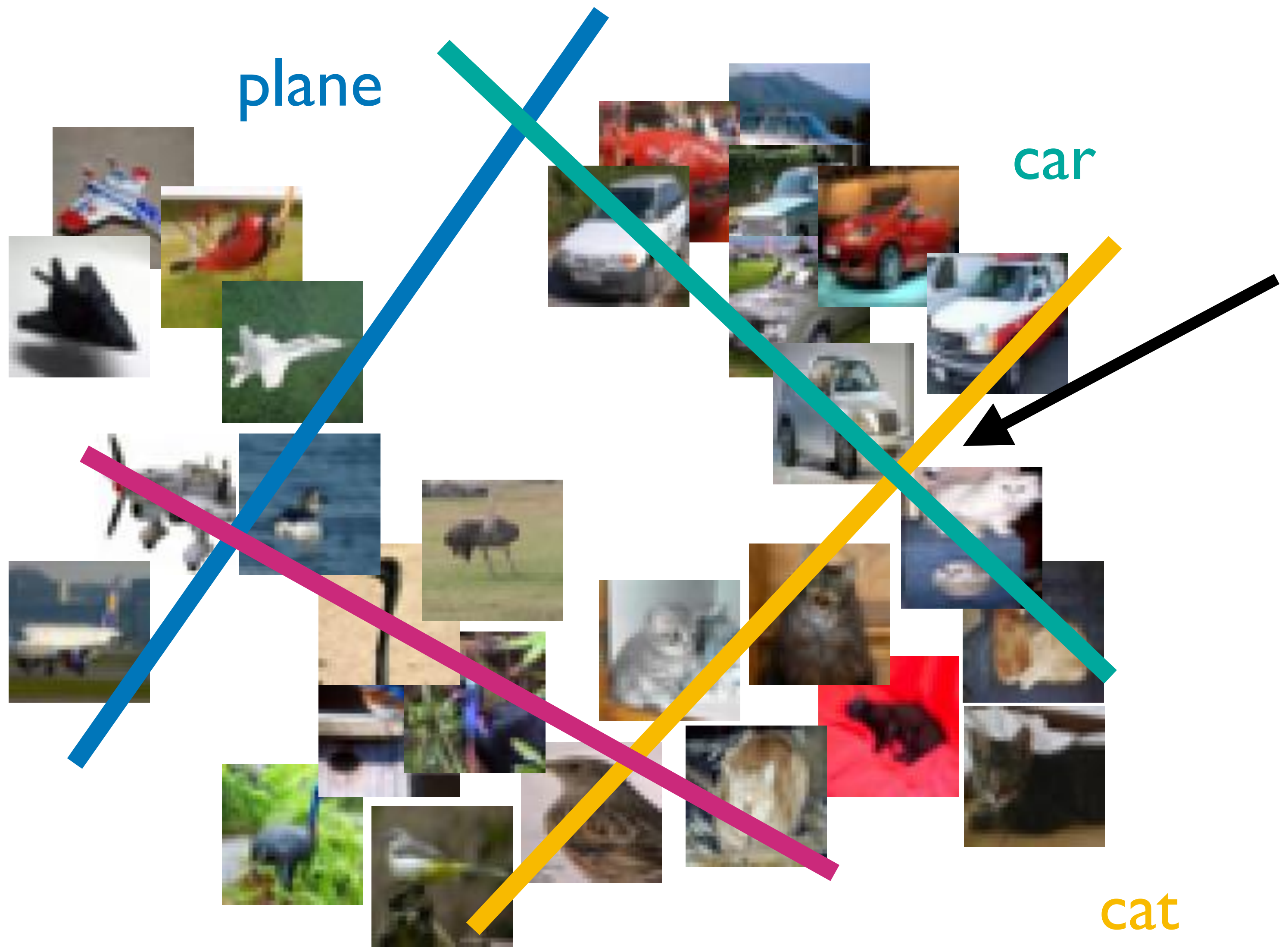
1-vs-All **Linear SVM**



Query:

$$\mathbf{x}_q =$$

1-vs-All **Linear SVM**



Query:

$\mathbf{x}_q =$ 

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

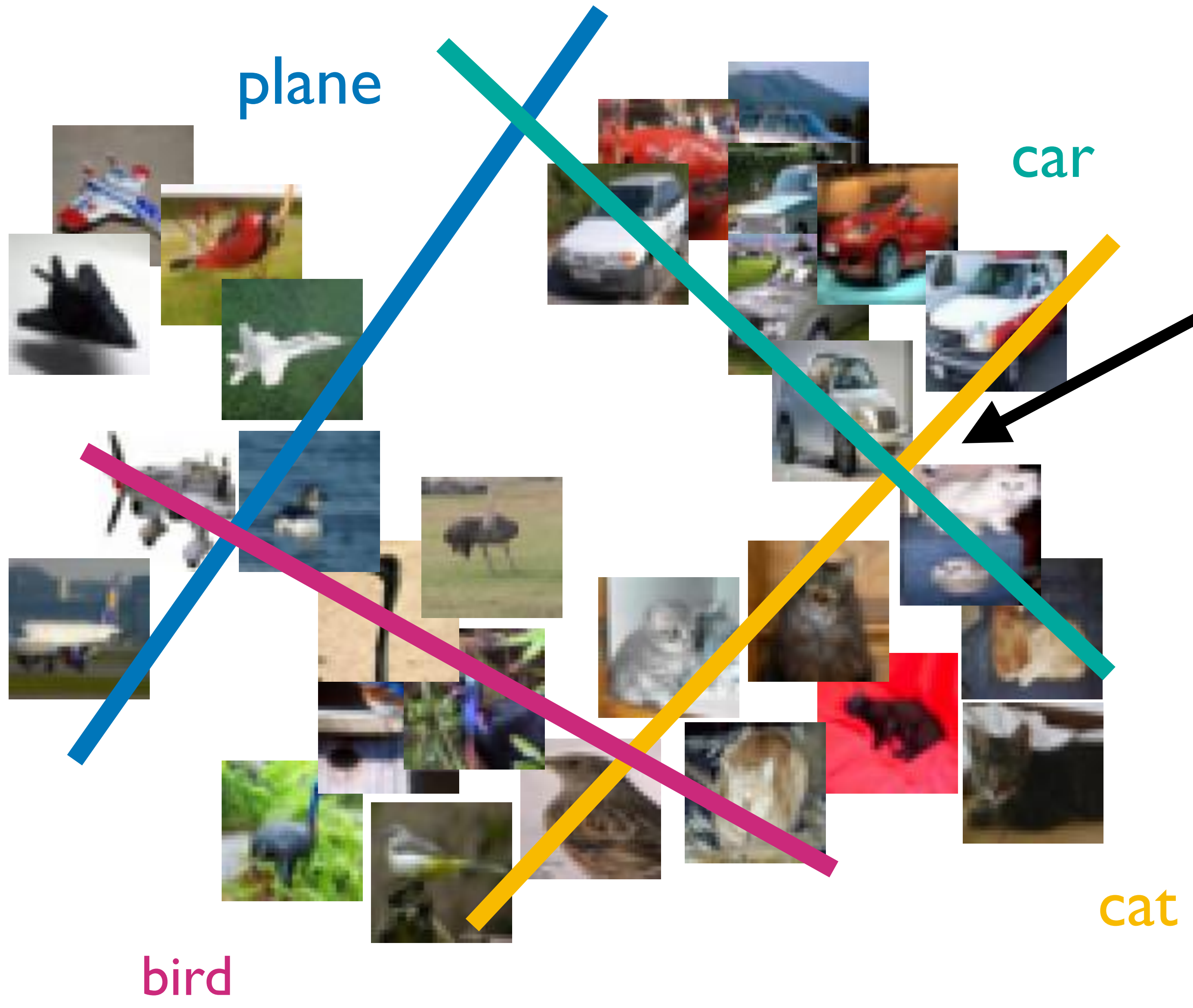
Performance:

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**

Decision Tree

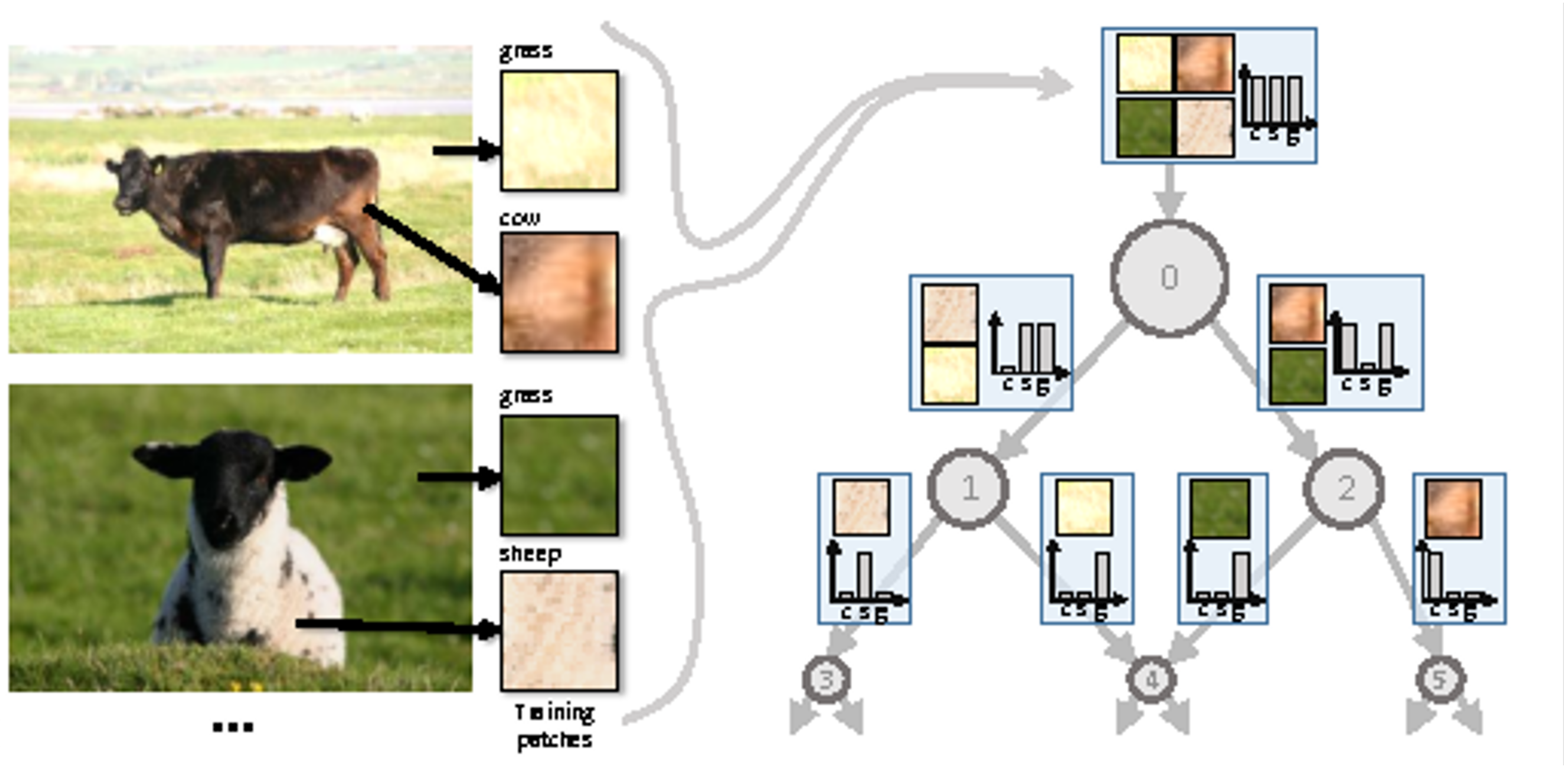
A **decision tree** is a simple non-linear parametric classifier

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

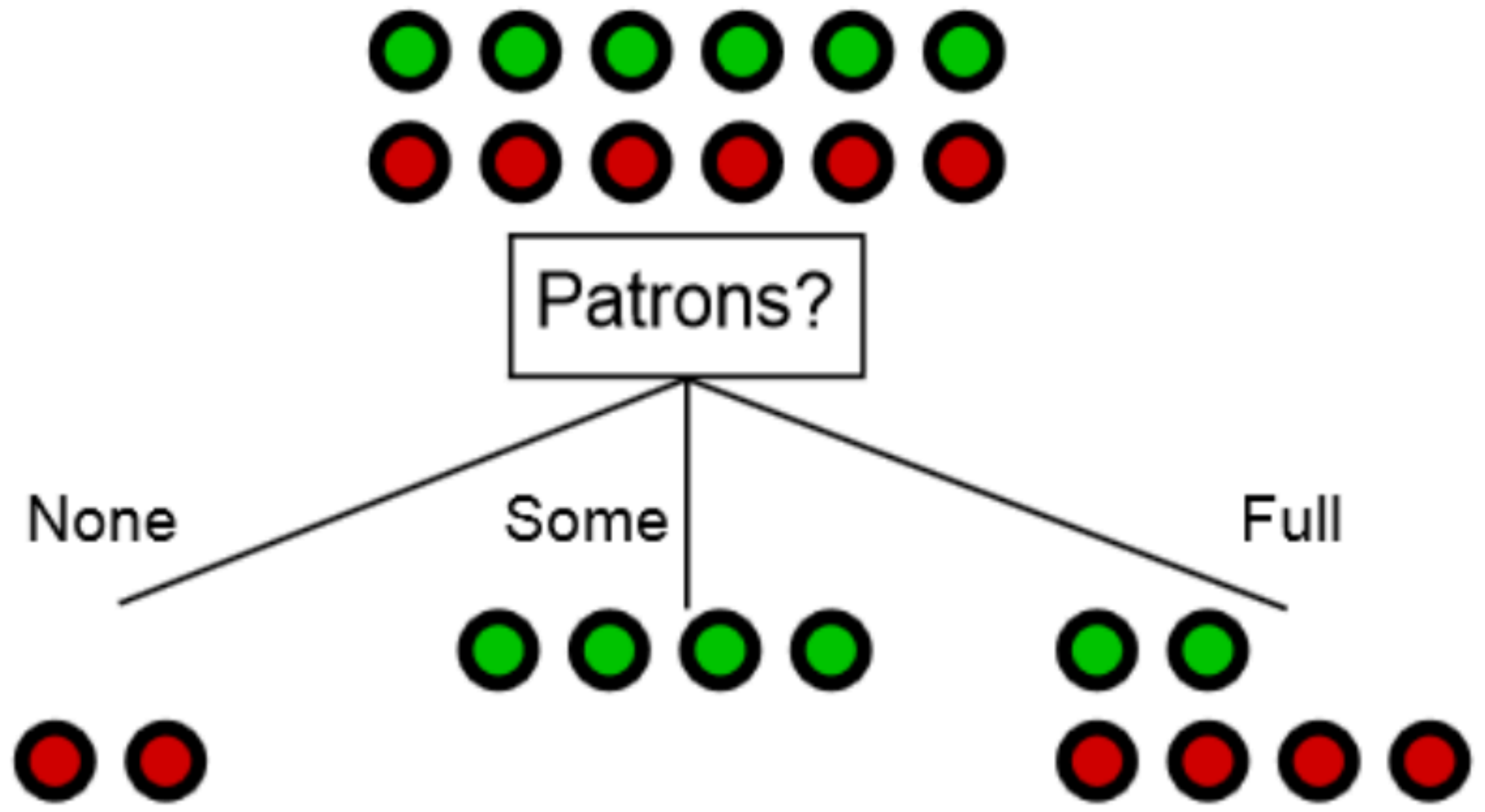
Example: Waiting for a restaurant table

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?



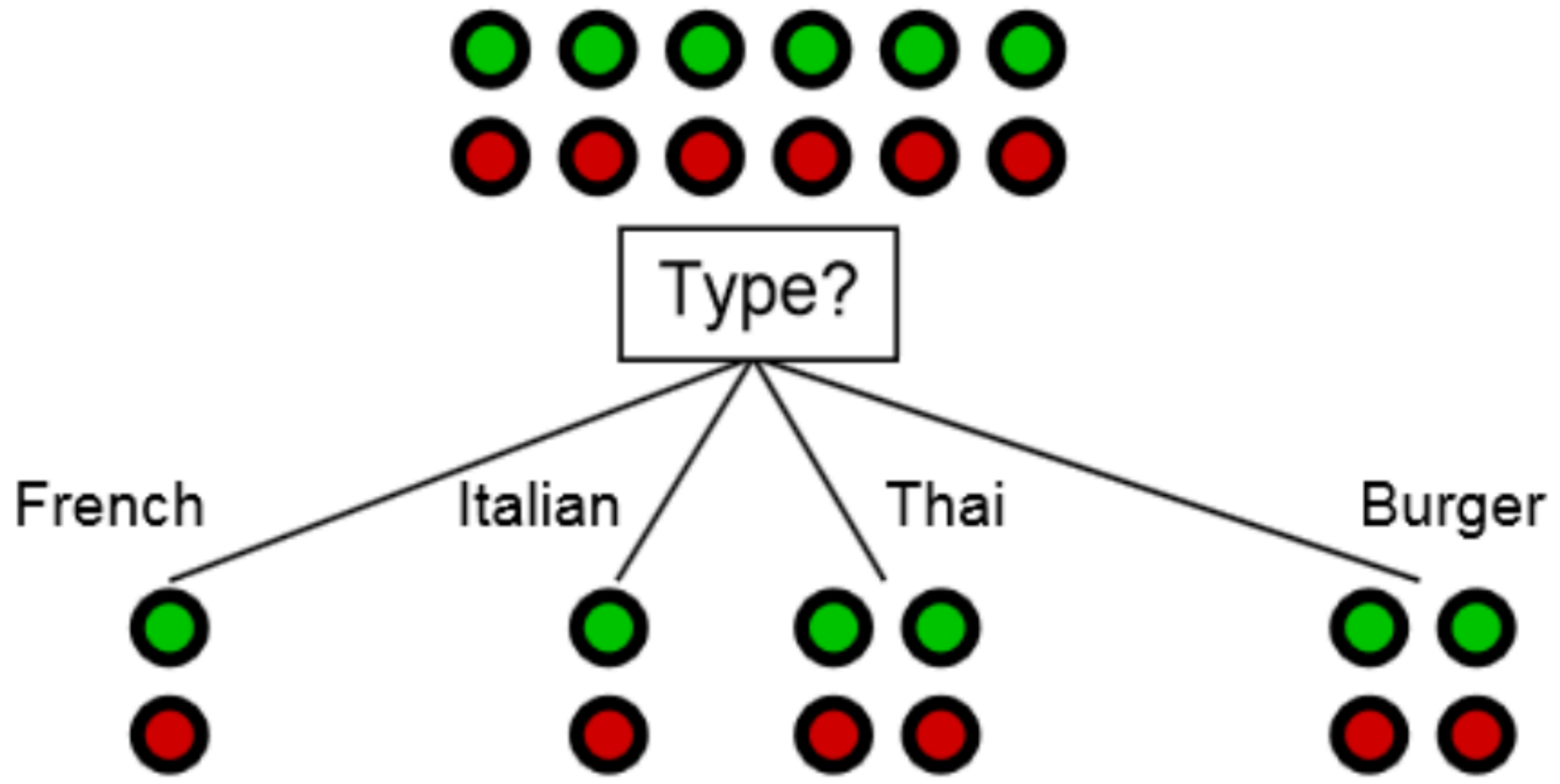


Figure credit: Russell and Norvig (3rd ed.)

Decision Tree

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

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

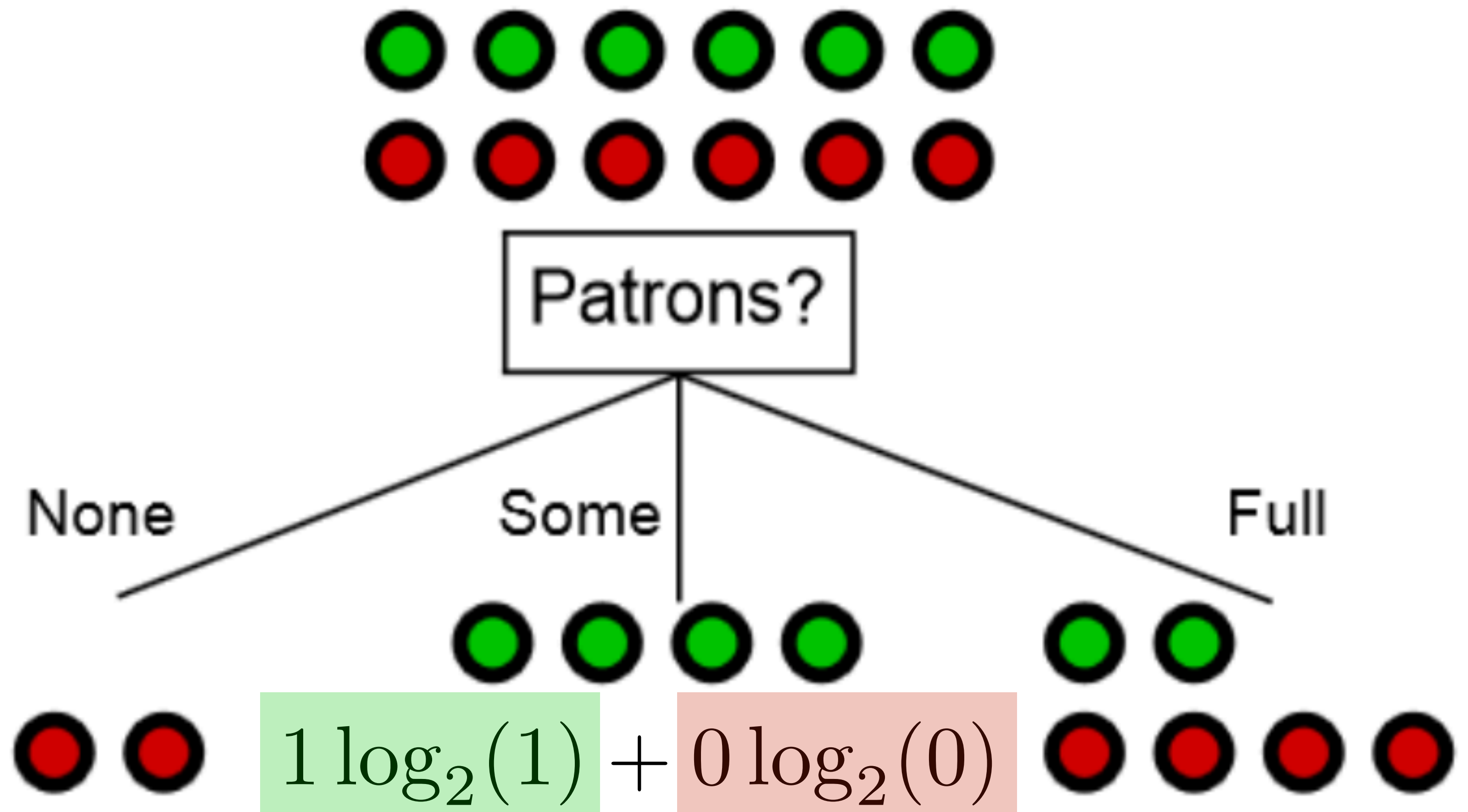
where C is the set of classes represented in S, and $p(c)$ is the empirical distribution of class c in S

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?

$0.5\log_2(0.5) + 0.5\log_2(0.5)$

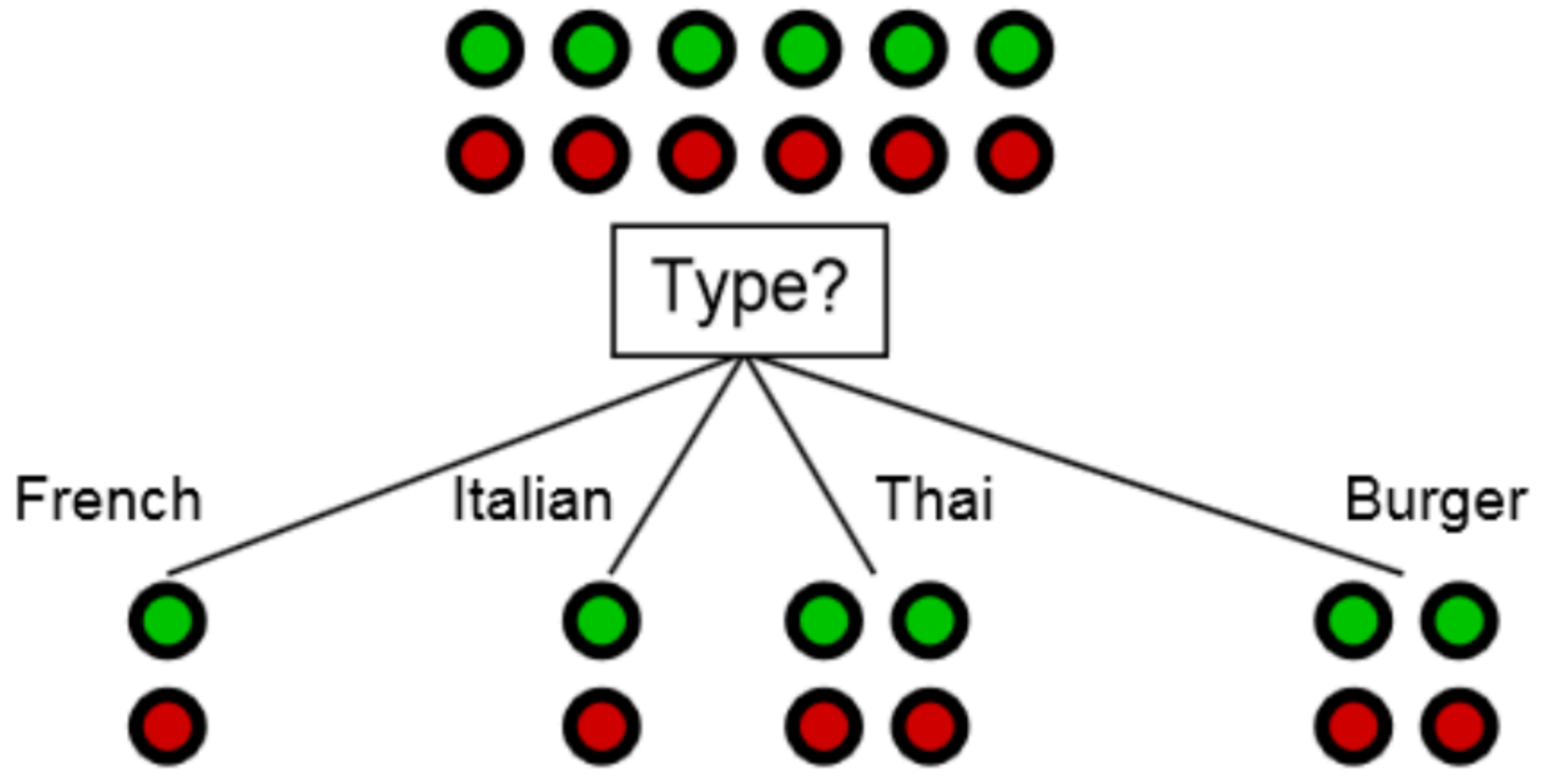


$0\log_2(0) + 1\log_2(1)$

$1\log_2(1) + 0\log_2(0)$

$0.\bar{3}\log_2(0.\bar{3}) + 0.\bar{6}\log_2(0.\bar{6})$

$0.5\log_2(0.5) + 0.5\log_2(0.5)$



$0.5\log_2(0.5) + 0.5\log_2(0.5)$

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:

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

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:

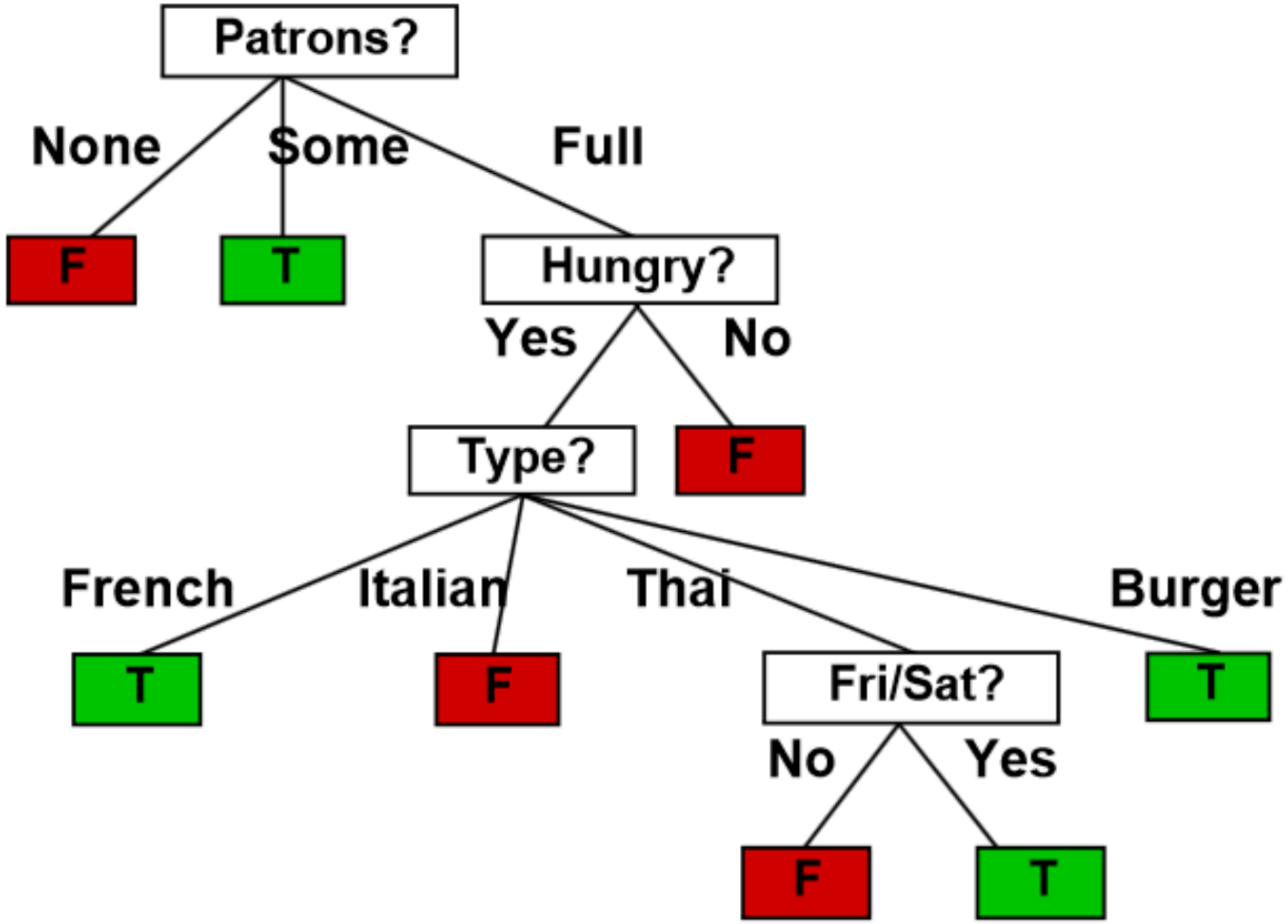


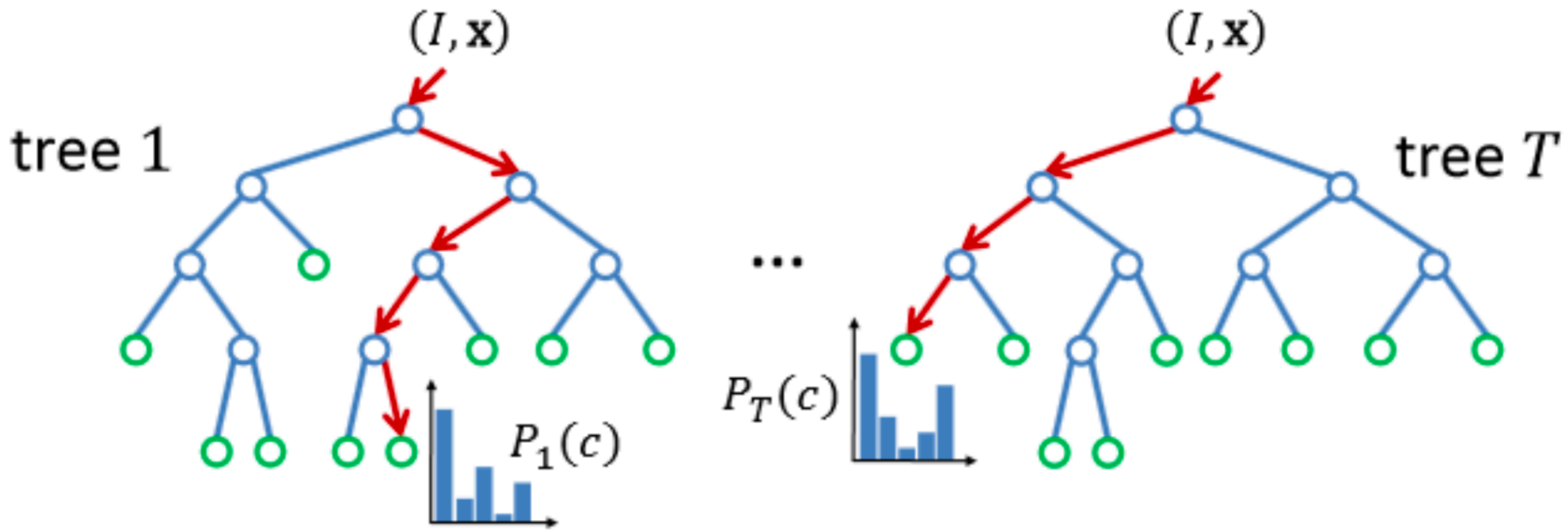
Figure credit: Russell and Norvig (3rd ed.)

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

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.

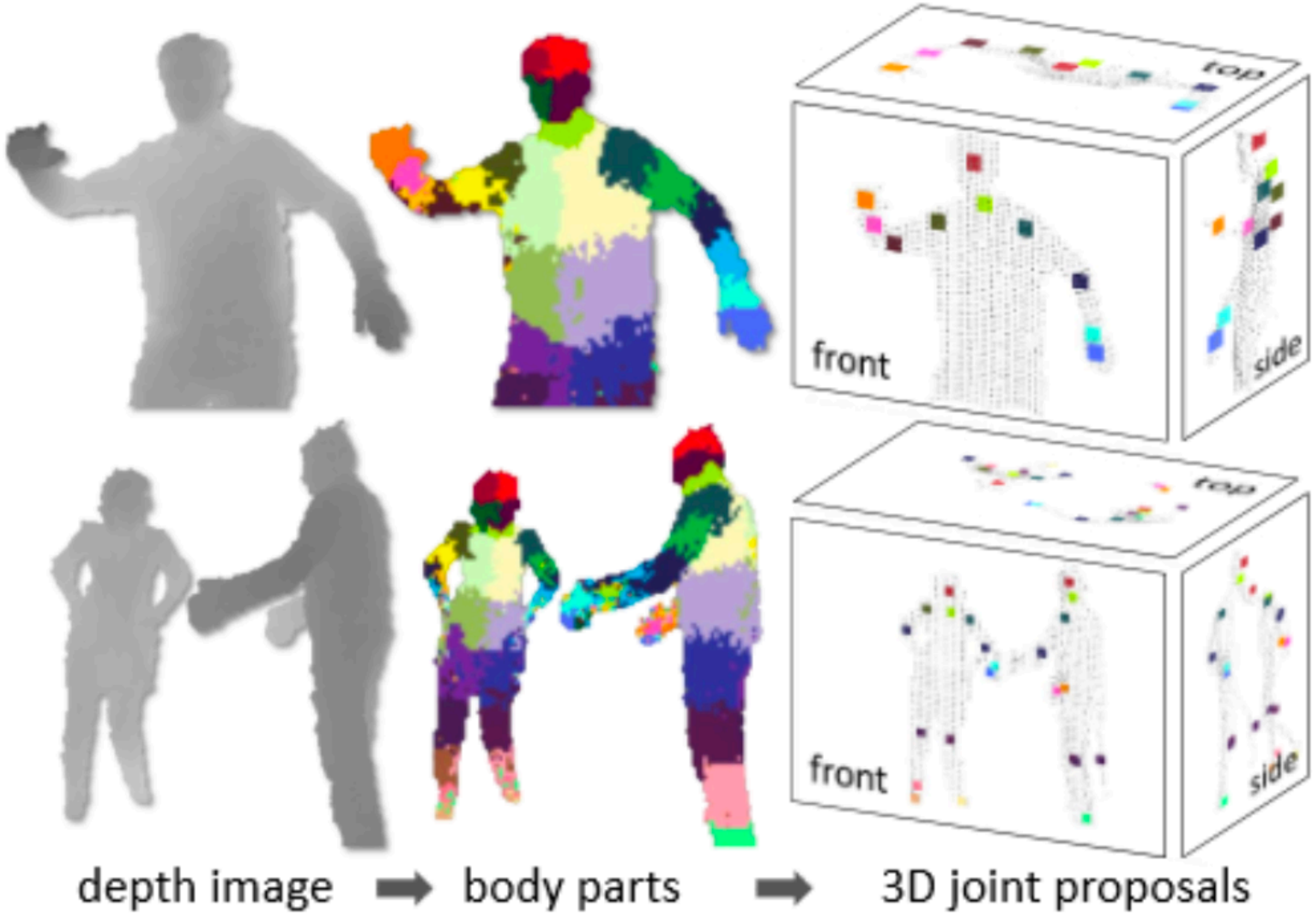
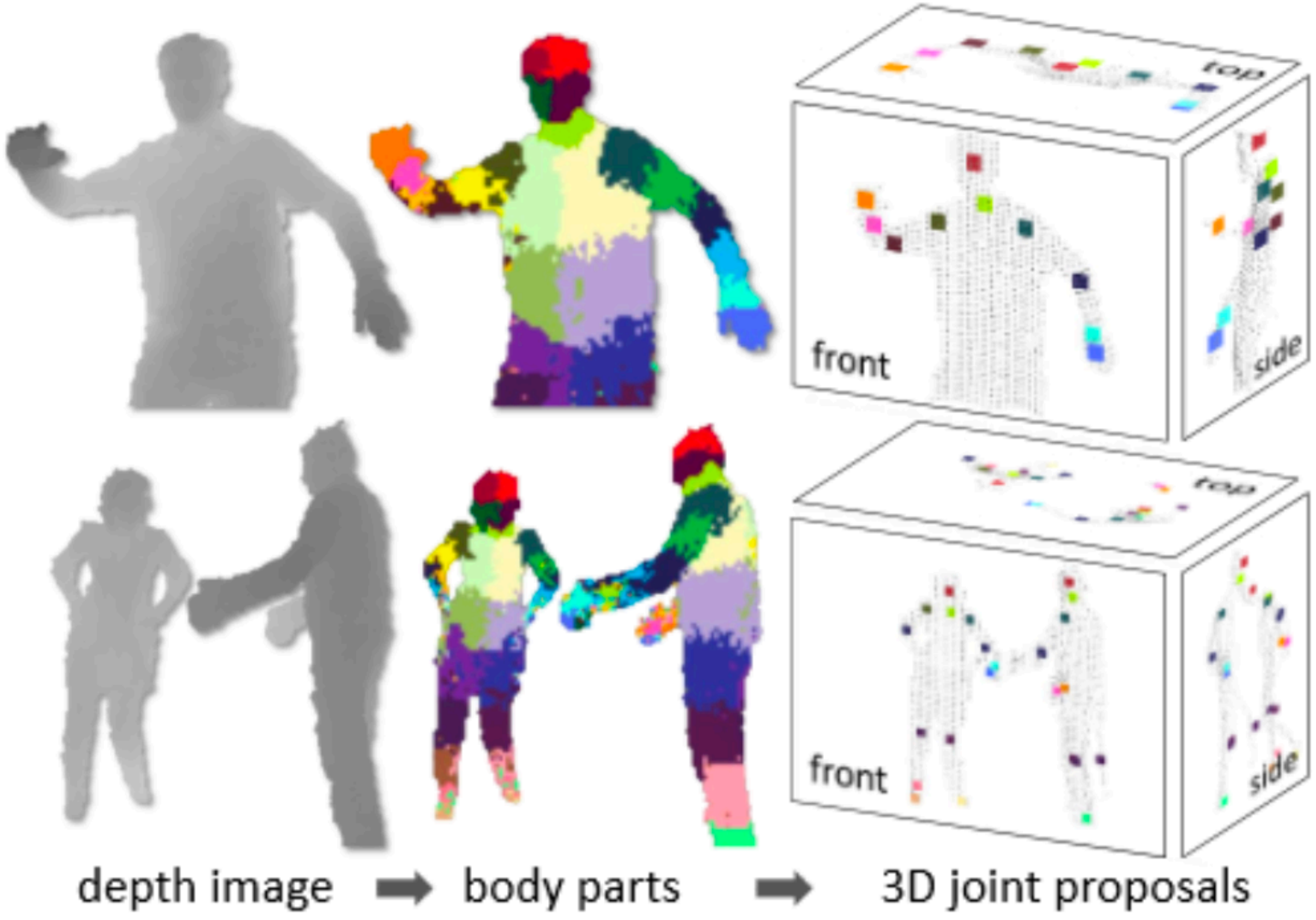


Figure credit: J. Shotton et al., 2011

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.

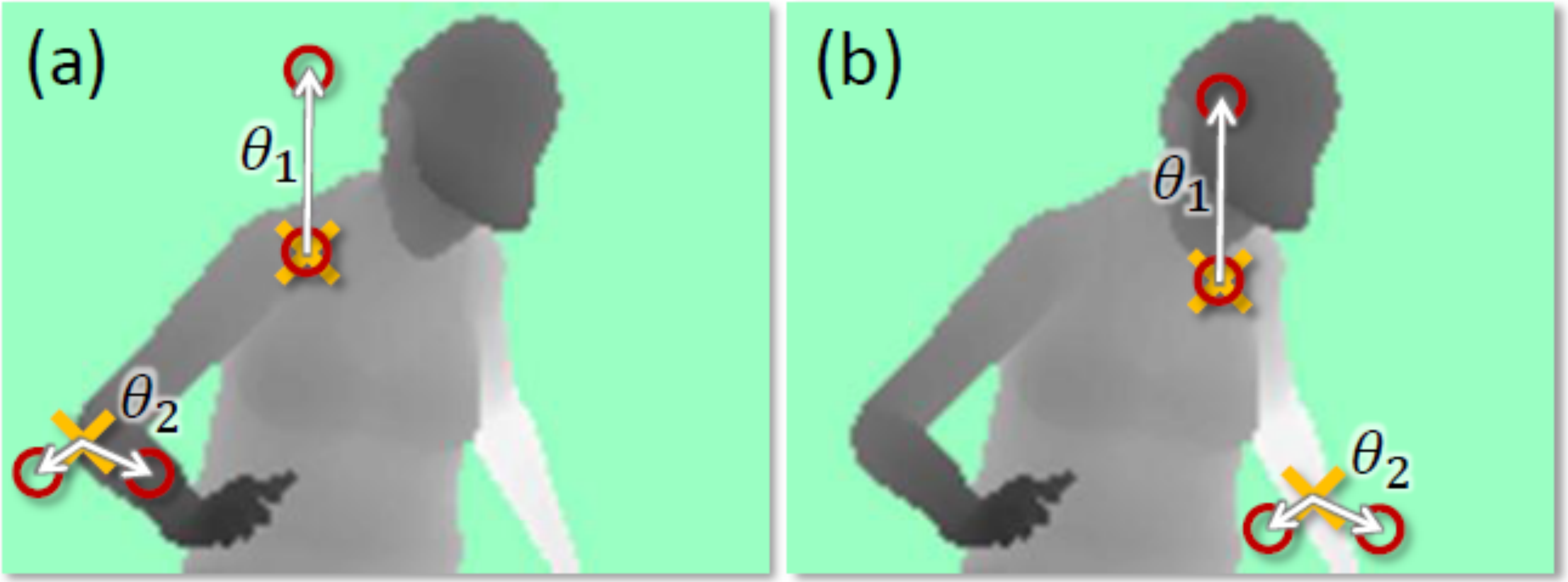


Jamie Shotton

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 …



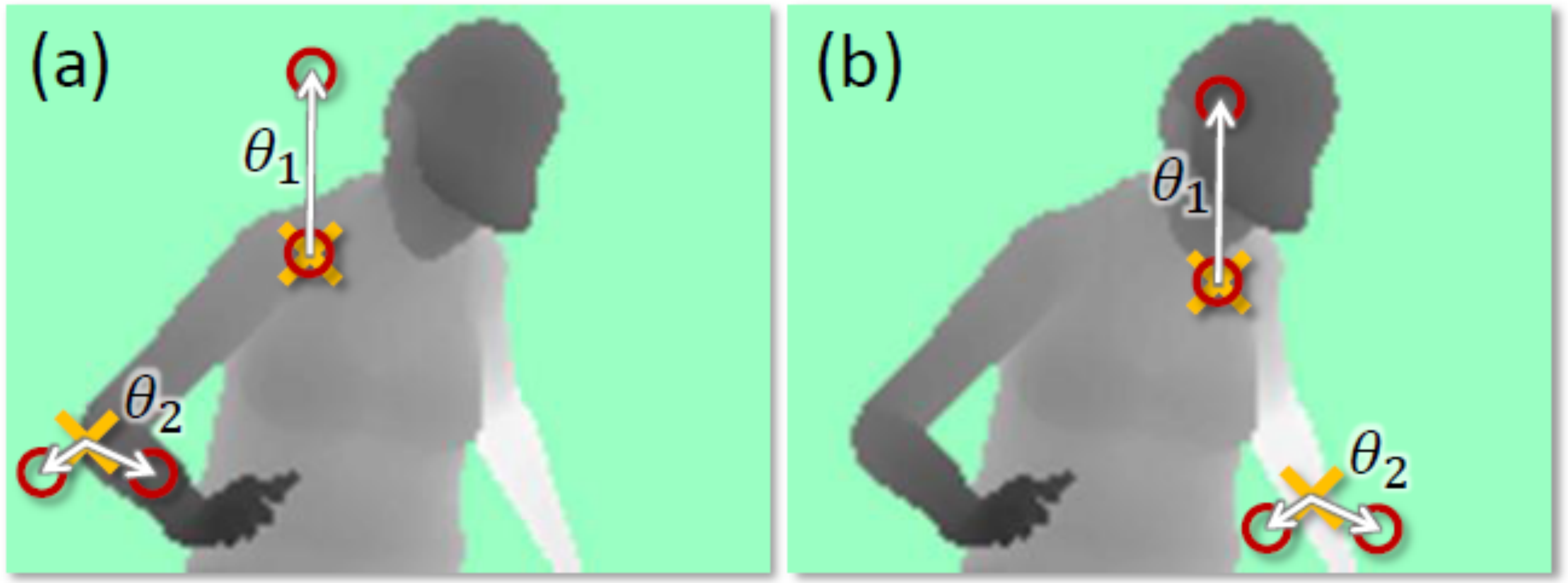
$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Figure credit: J. Shotton et al., 2011

Example 1: Kinect

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

What are the parameters of this test?

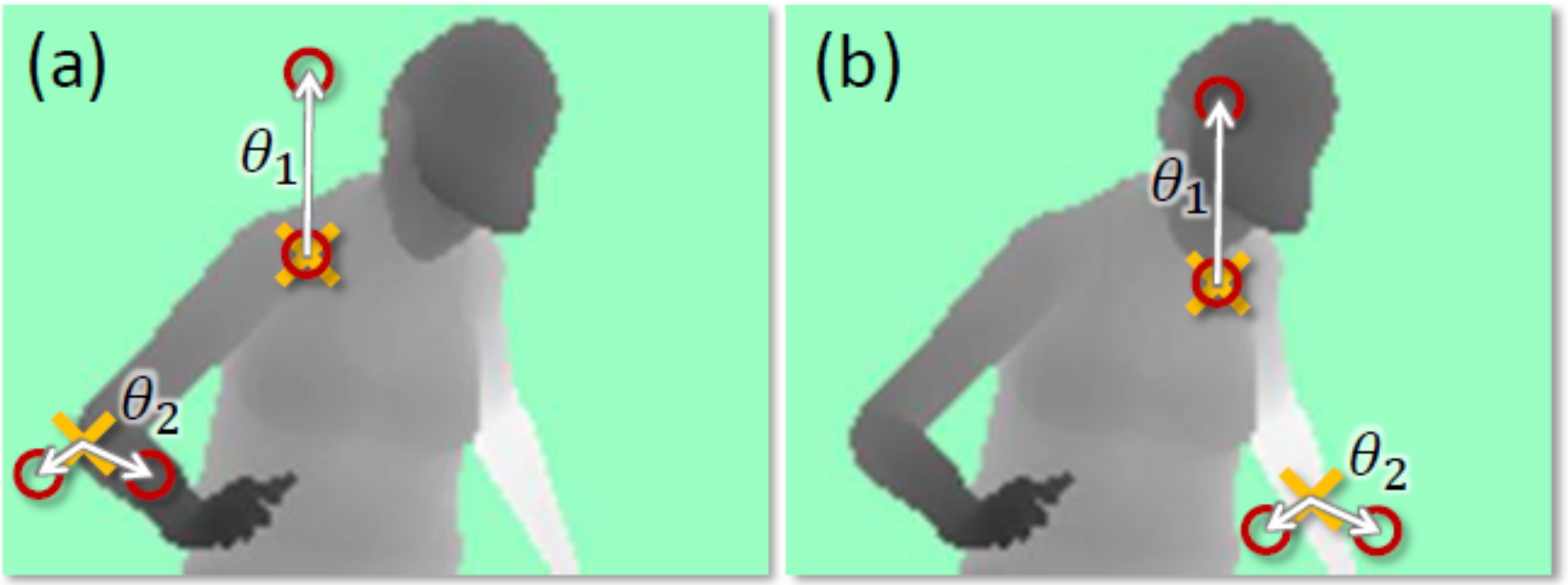


$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Example 1: Kinect

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

What are the parameters of this test?



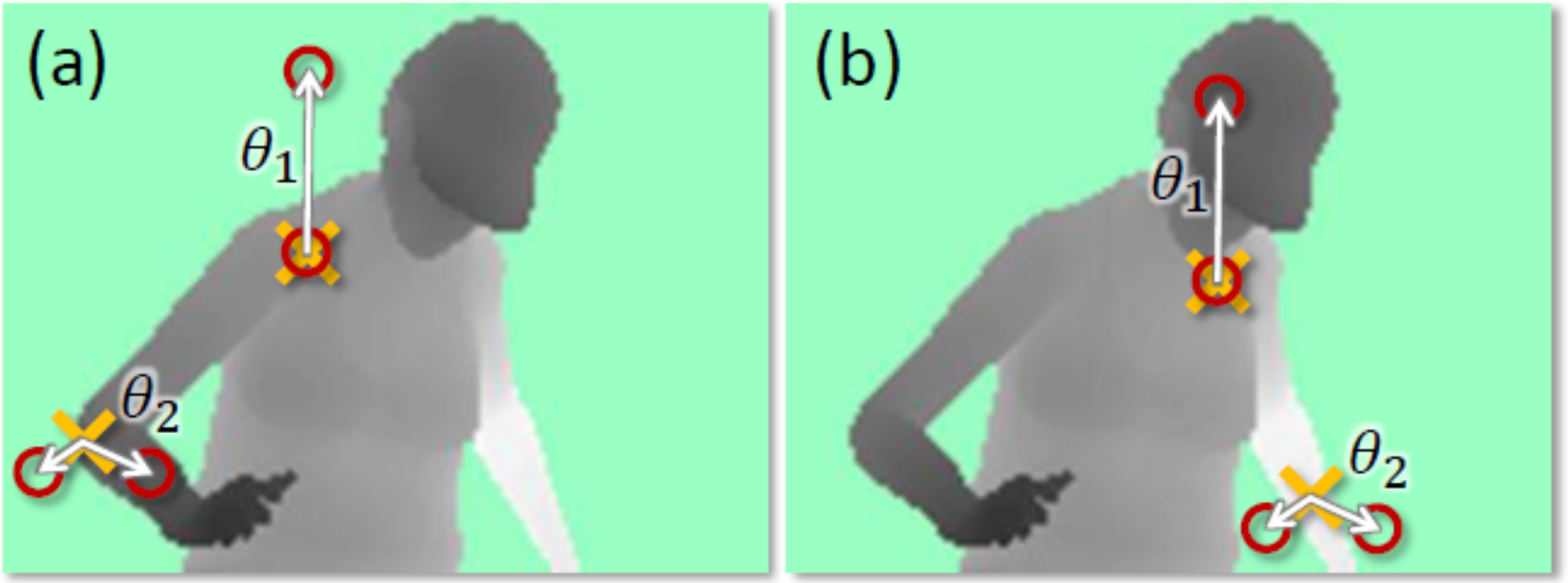
$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

Example 1: Kinect

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

What are the parameters of this test?

How many such tests can we have?



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

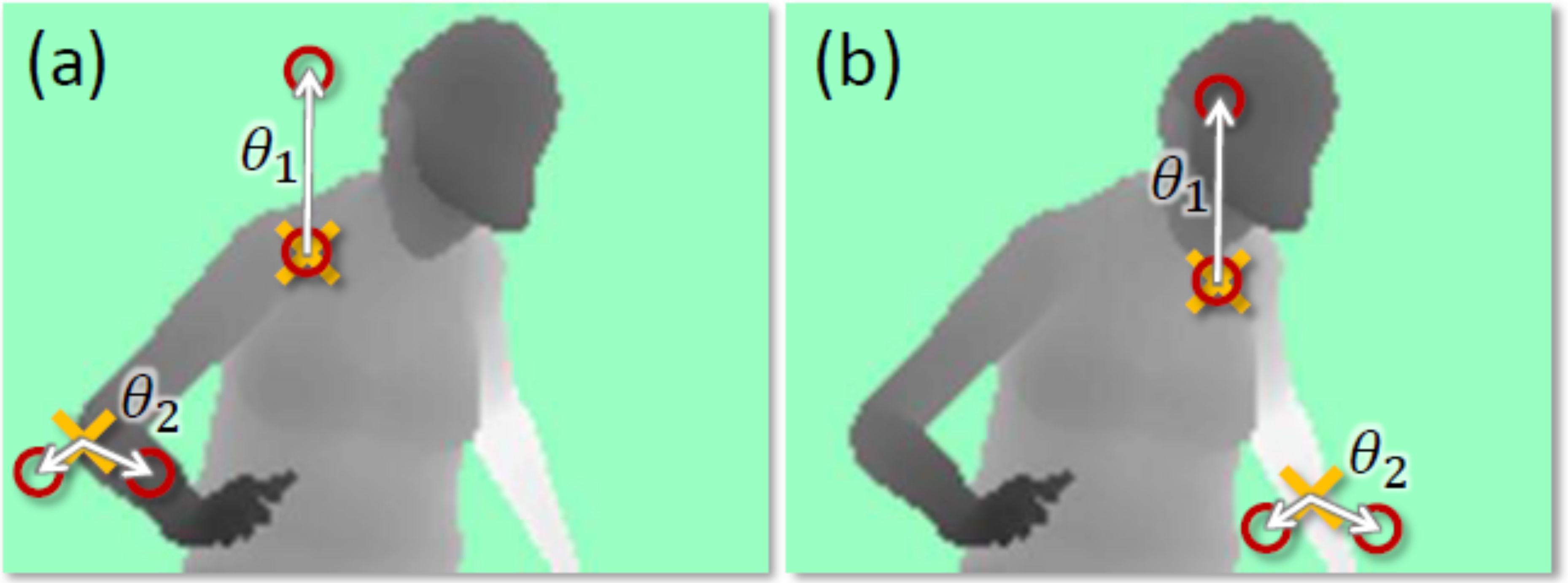
Example 1: Kinect

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

What are the parameters of this test?

How many such tests can we have?

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



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

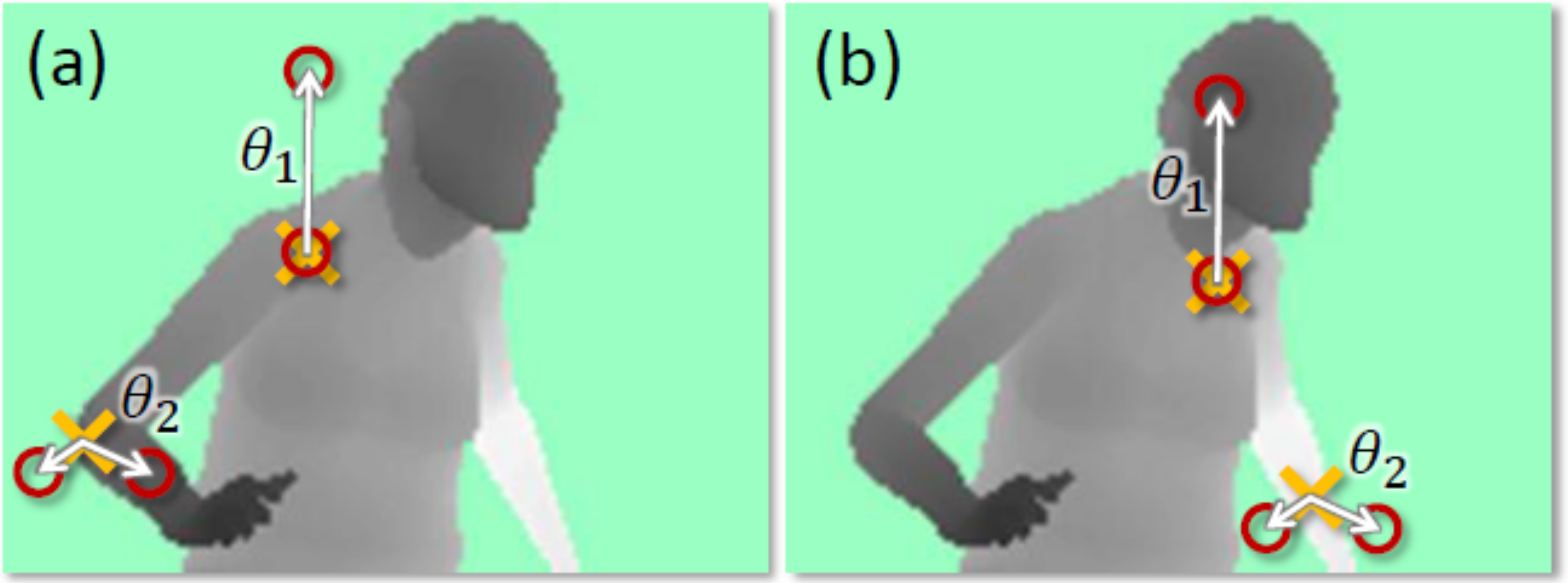
Example 1: Kinect

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

What are the parameters of this test?

How many such tests can we have?

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



Learning is slow (weeks)!

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

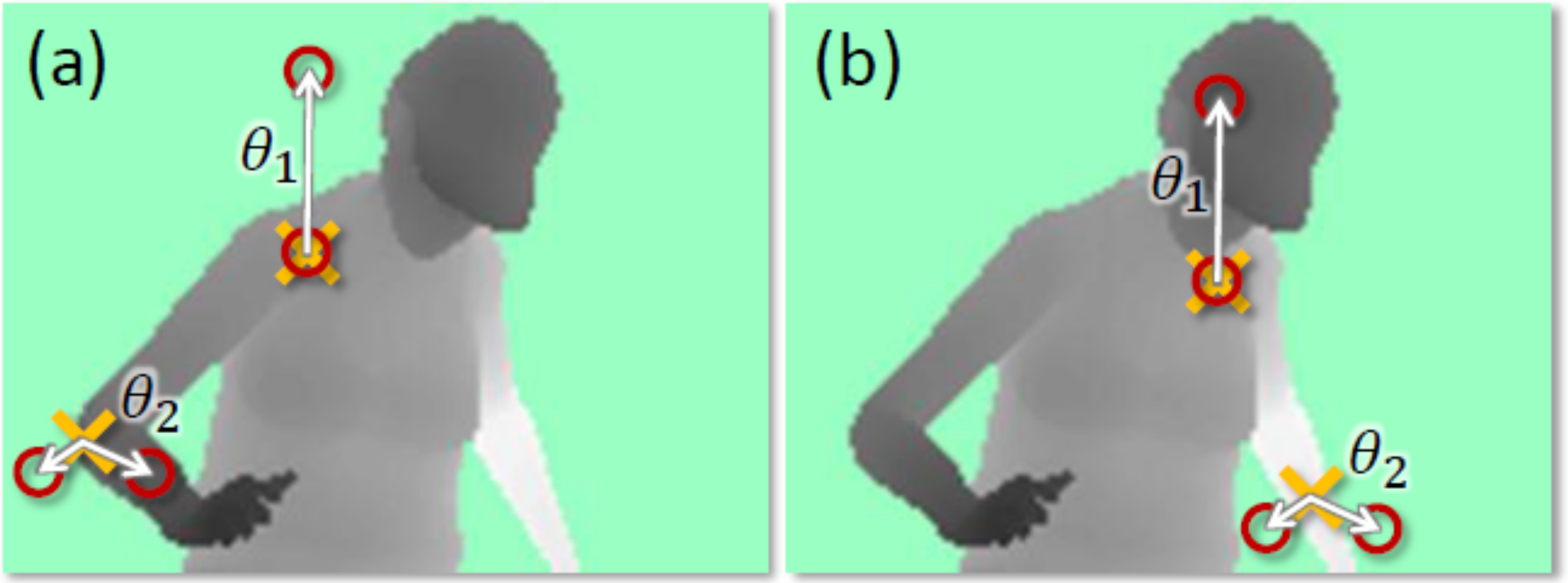
Example 1: Kinect

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

What are the parameters of this test?

How many such tests can we have?

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



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

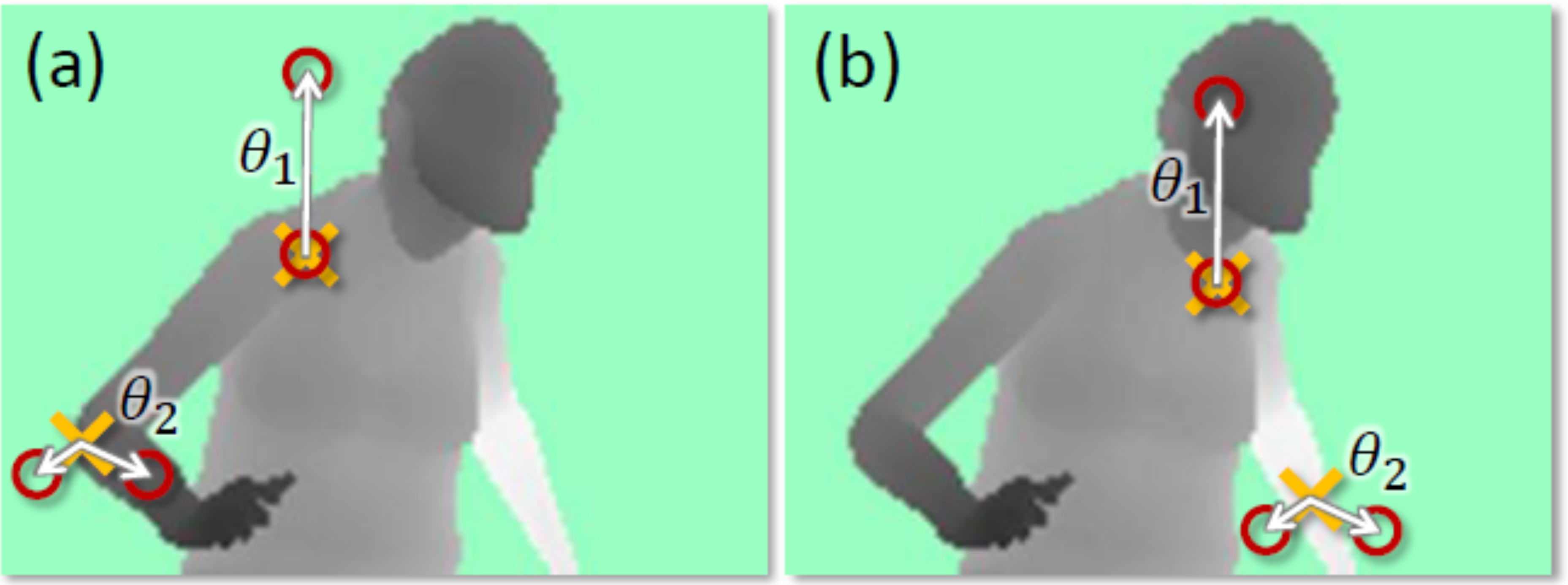
Learning is slow (weeks)!

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
…	…	…

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

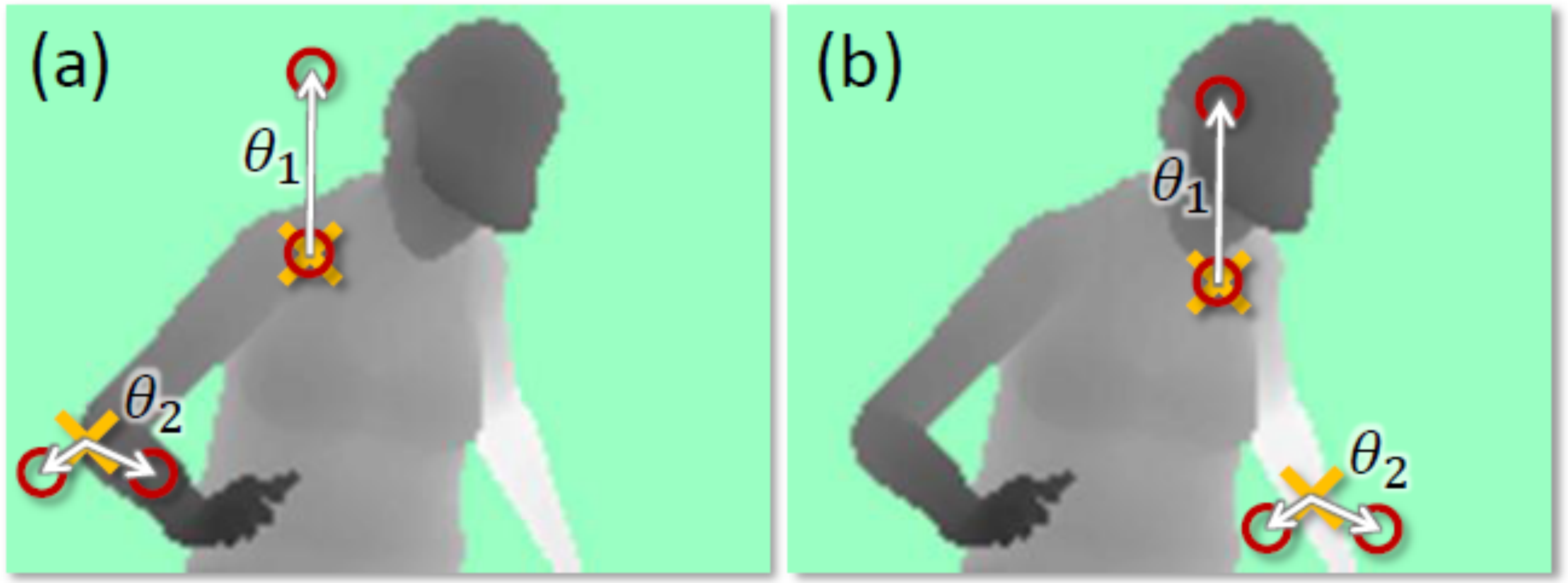


$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

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
...	...	...	...

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

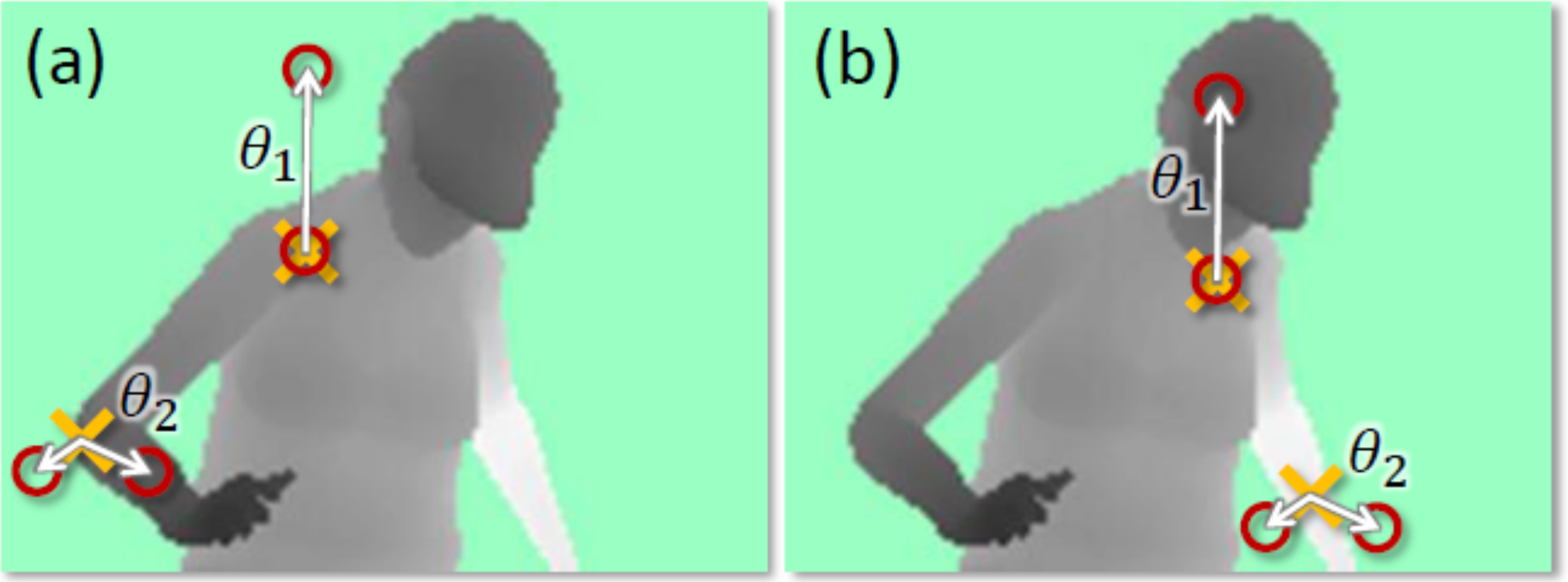


$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

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
…	…	…	…

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



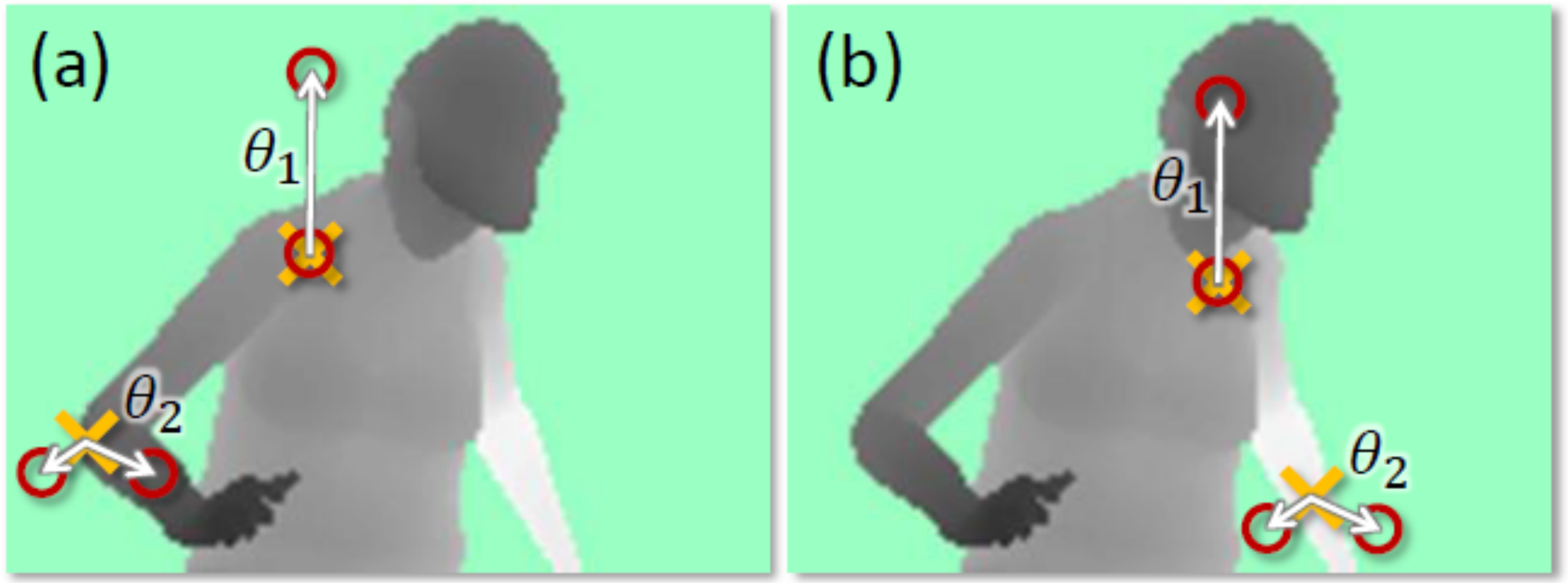
$$f_\theta(I, \mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

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

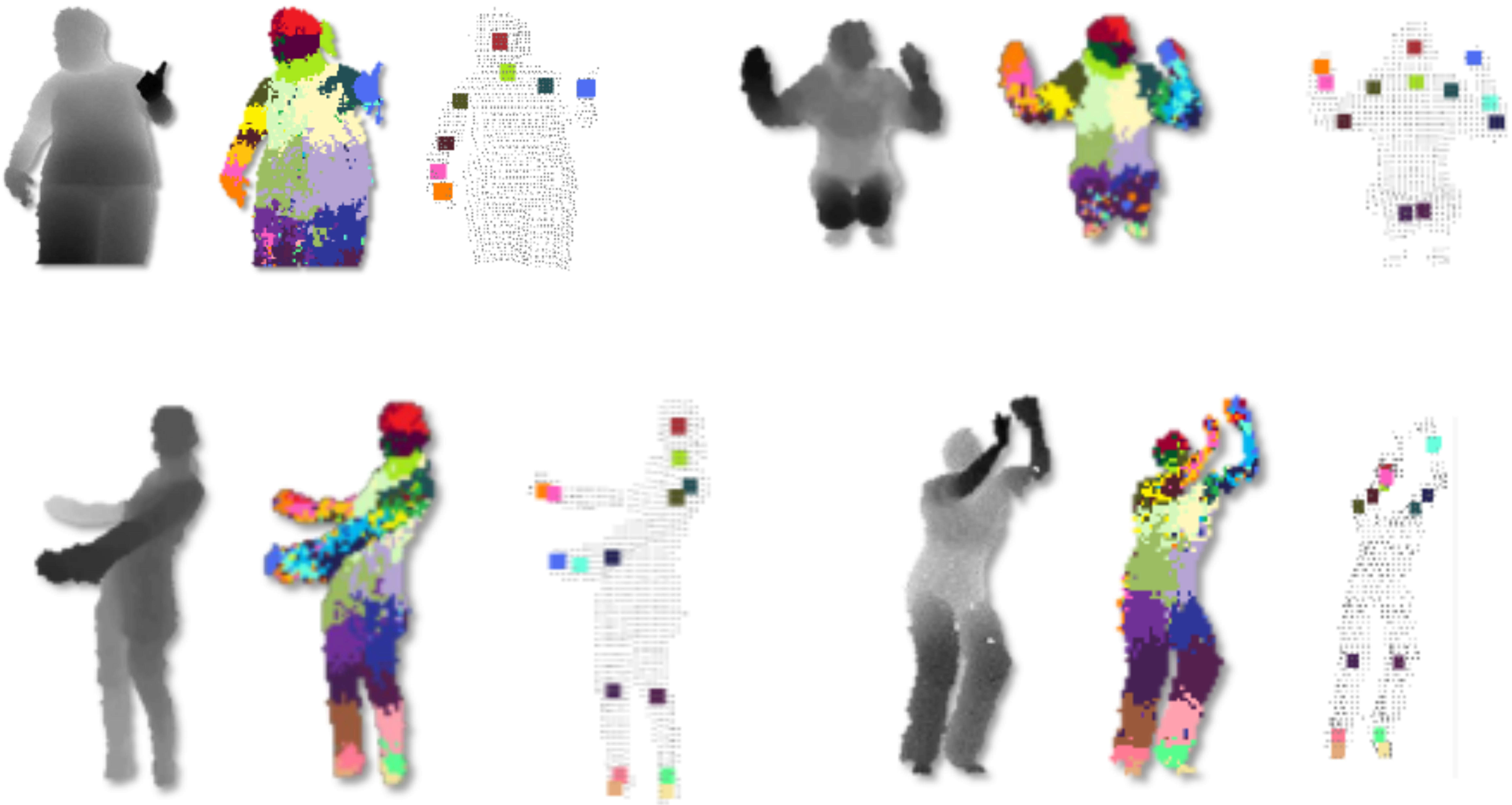


Figure credit: J. Shotton et al., 2011