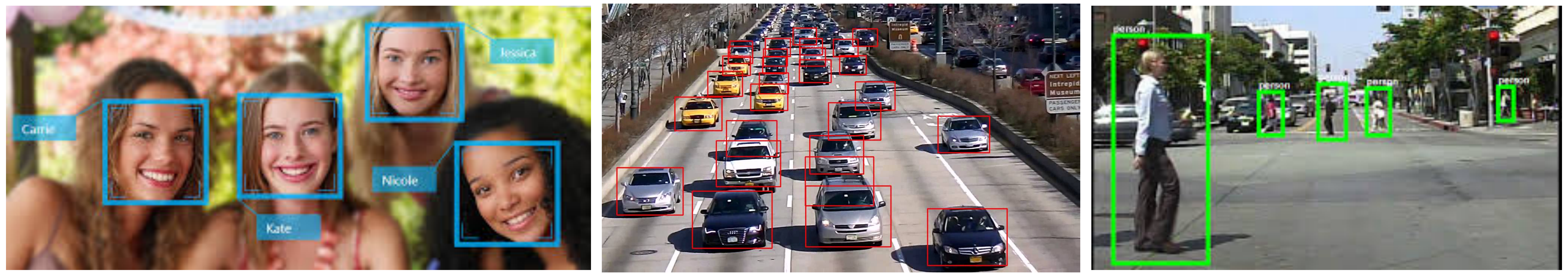
THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 425: Computer Vision



Lecture 21: Object Detection (cont.)

Menu for Today (**March 26, 2020**)

Topics:

- Deformable part models
- Object Proposals
- Grouping
- Image Segmentation

Redings:

- **Today's** Lecture: Forsyth & Ponce (2nd ed.) 15.1, 15.2, 17.2
- **Next** Lecture: Deep Learning (N/A)

Reminders:

- **Assignment 5**: Scene Recognition with Bag of Words due **March 31st**
- **Assignment 6:** Deep Learning will be available **March 31st**

Lecture 20: Re-cap — Boosting

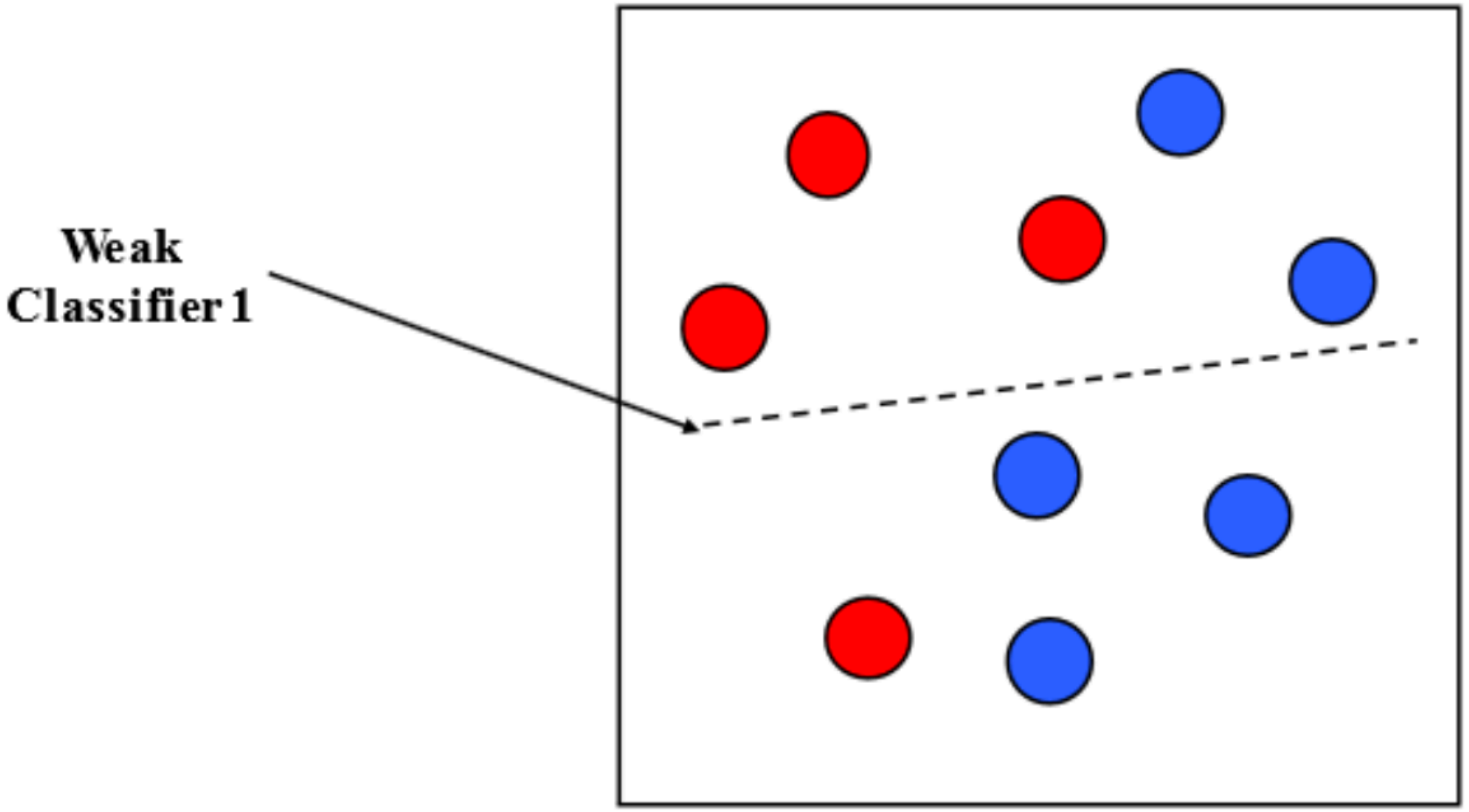


Figure credit: Paul Viola

Lecture 20: Re-cap — Boosting

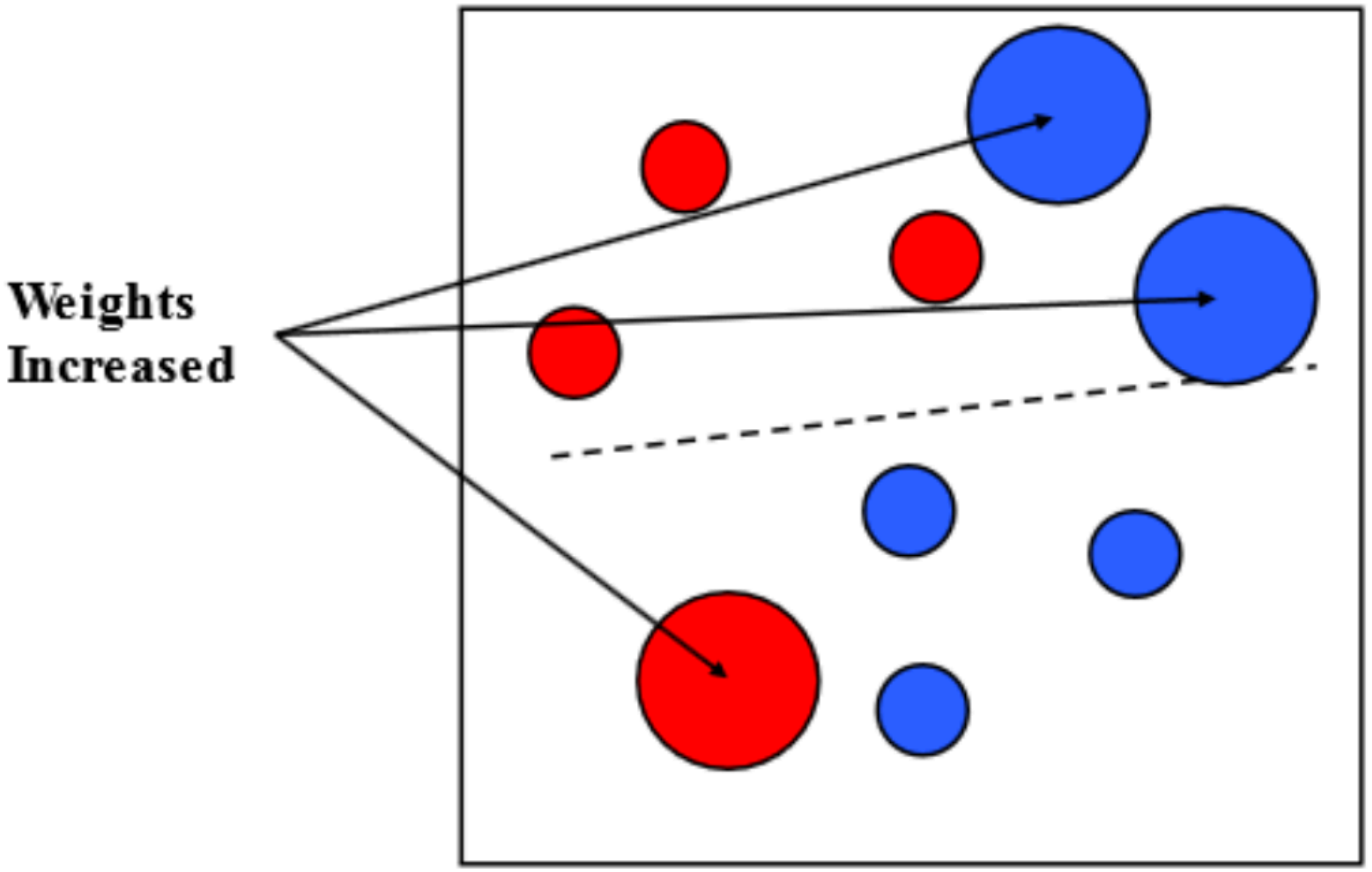


Figure credit: Paul Viola

Lecture 20: Re-cap — Boosting

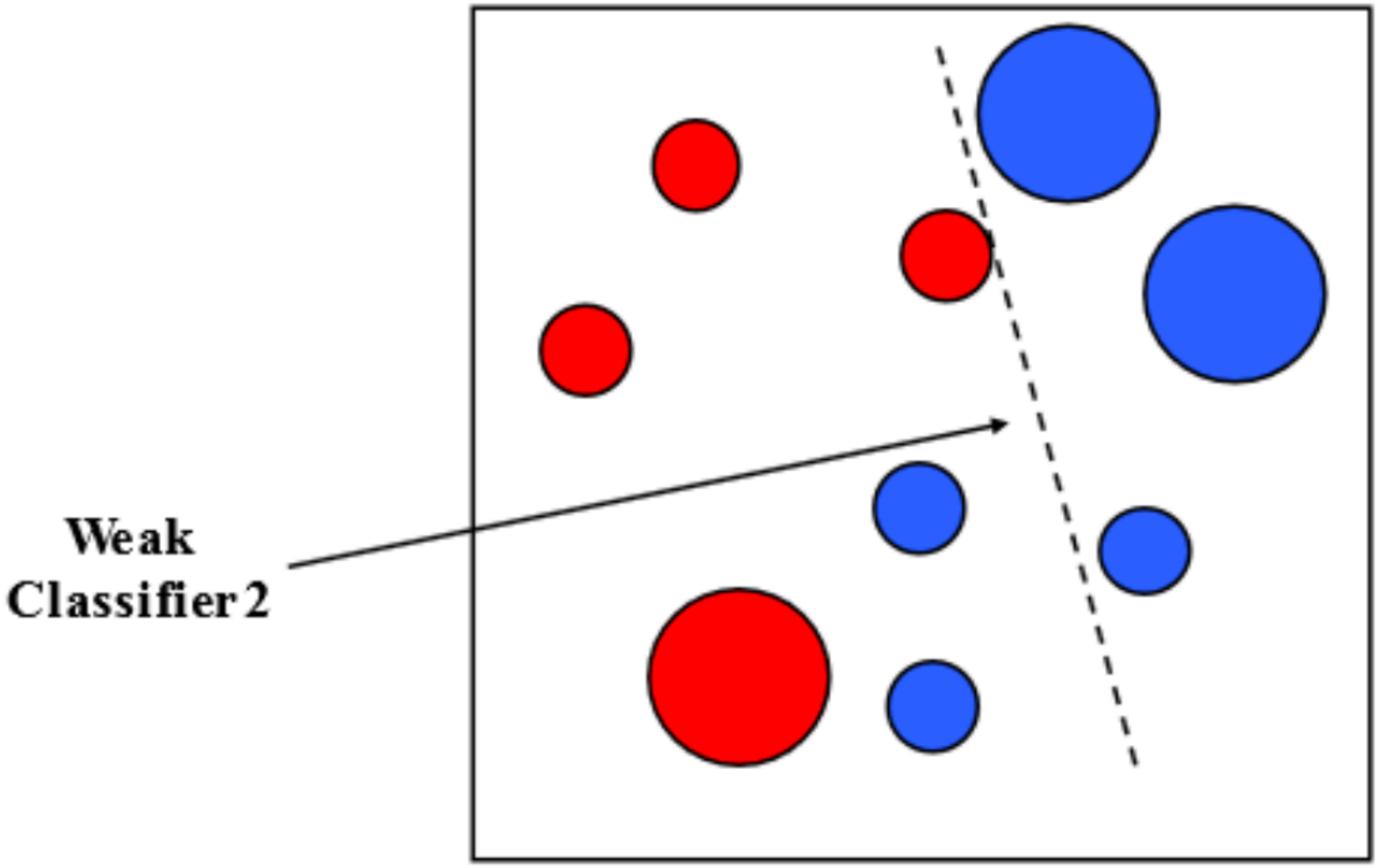


Figure credit: Paul Viola

Lecture 20: Re-cap — Boosting

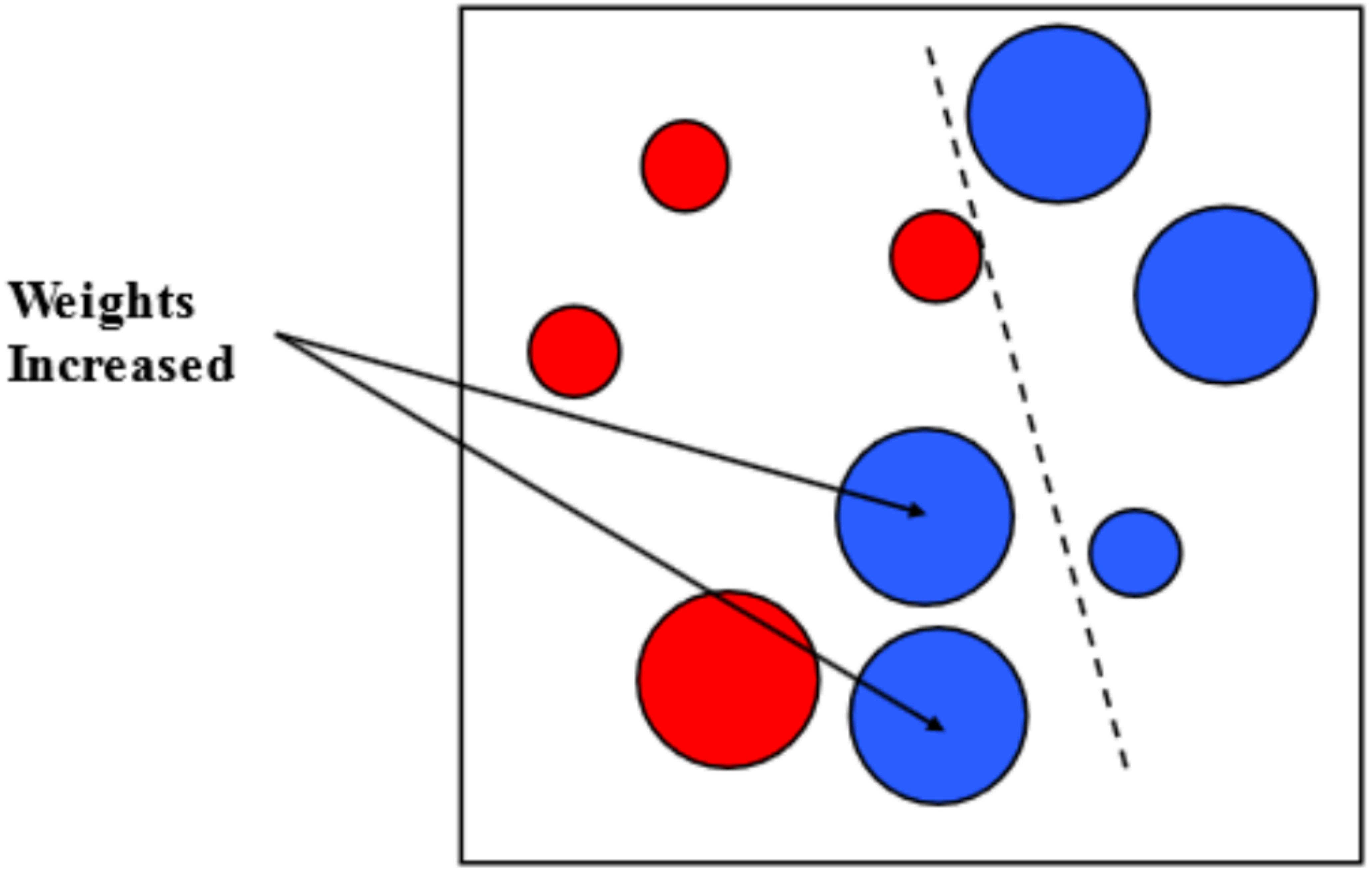


Figure credit: Paul Viola

Lecture 20: Re-cap — Boosting

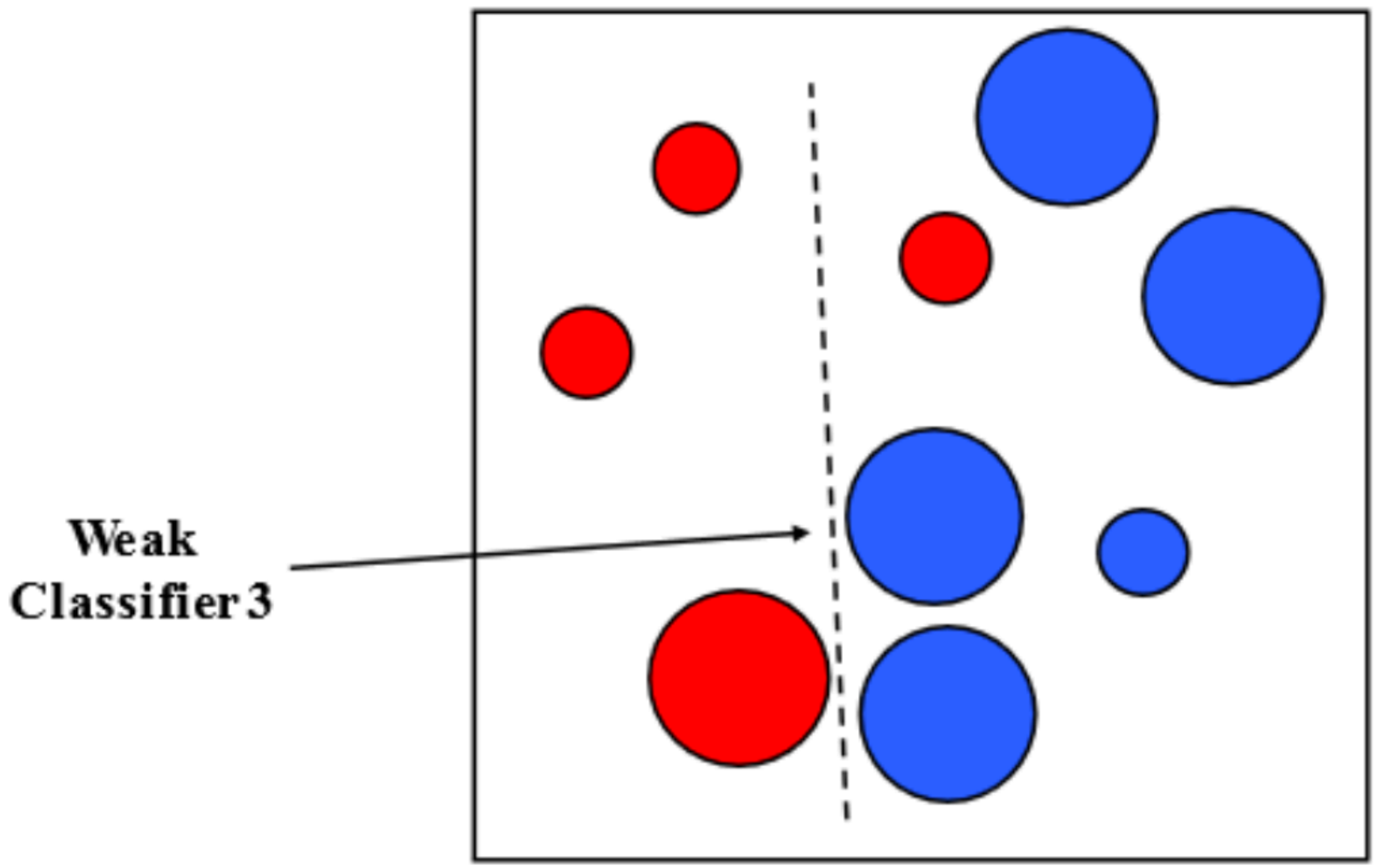


Figure credit: Paul Viola

Lecture 20: Re-cap — Boosting

Final classifier is a combination of weak classifiers

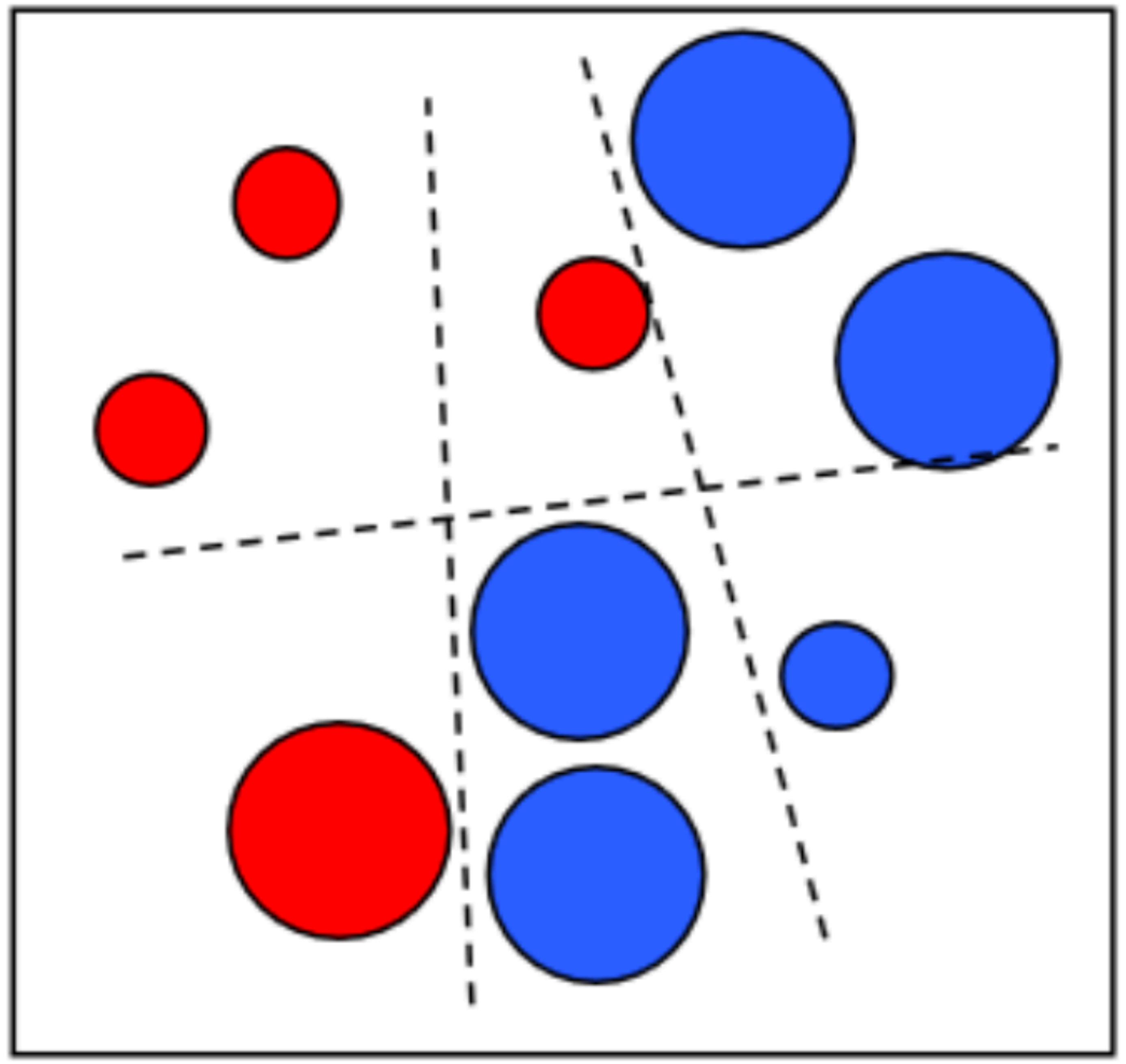


Figure credit: Paul Viola

Lecture 20: Re-cap — 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

Lecture 20: Re-cap — 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

Lecture 20: Re-cap — 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

Lecture 20: Re-cap — 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

Lecture 20: Re-cap — 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

Lecture 20: Re-cap — 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.

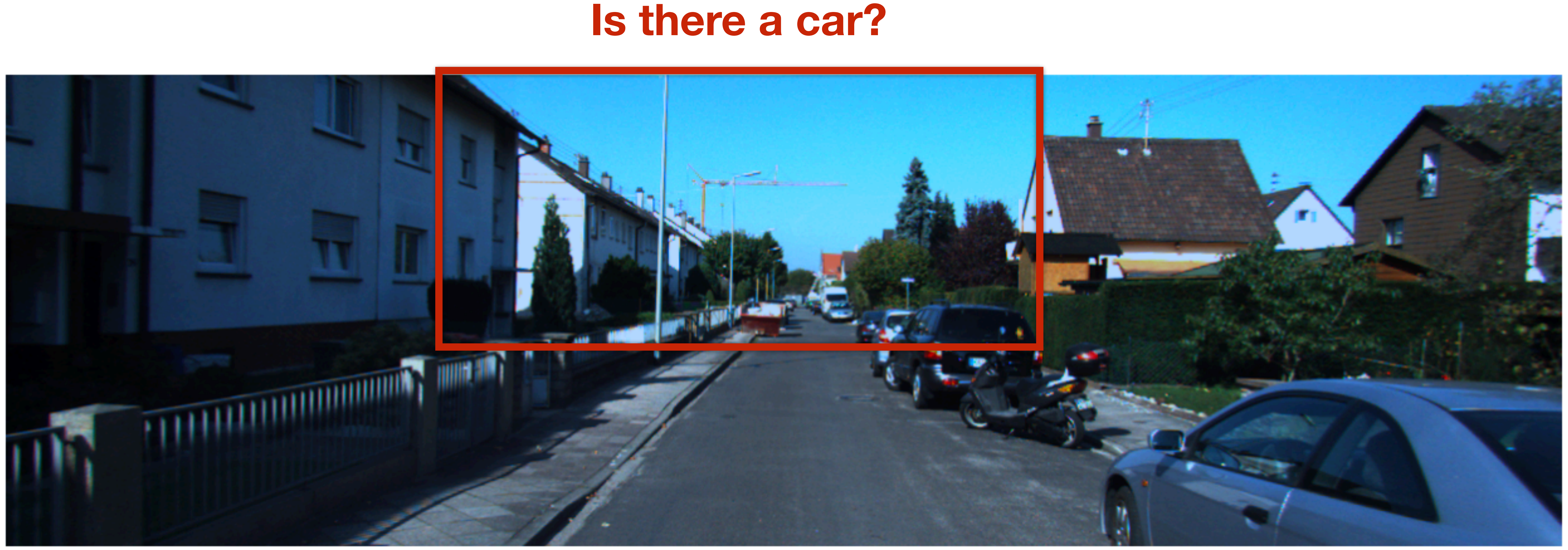


Image credit: KITTI Vision Benchmark

Lecture 20: Re-cap — 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?

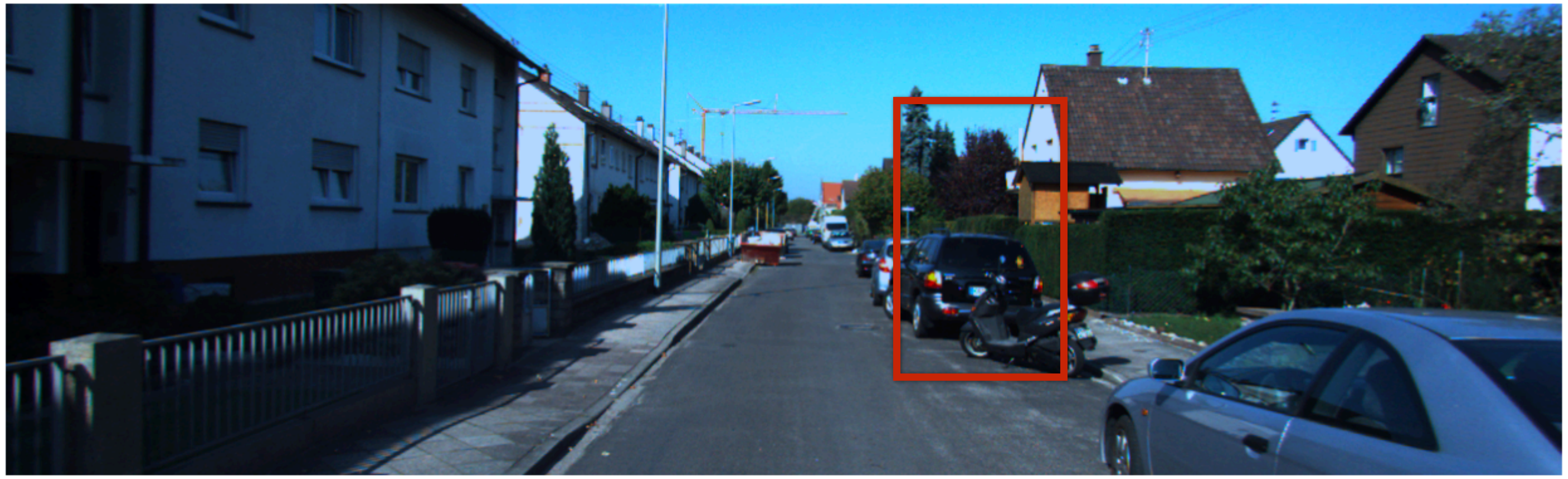


Image credit: KITTI Vision Benchmark

Lecture 20: Re-cap — 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

Lecture 20: Re-cap — Sliding Window

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

Image credit: KITTI Vision Benchmark

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

Example: Face Detection

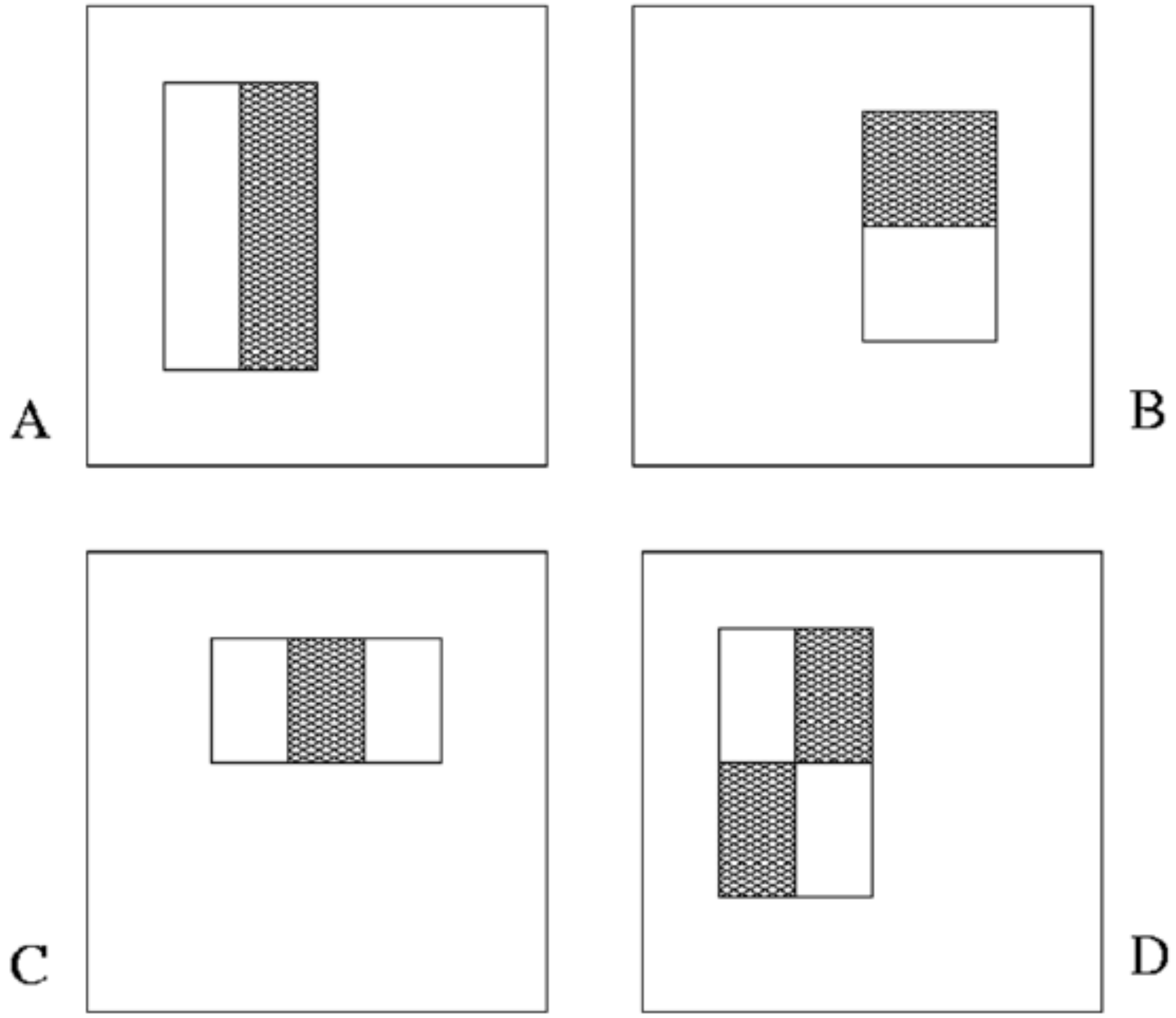
The **Viola-Jones** face detector is a classic sliding window detector that learns both efficient features and a classifier

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

A ‘rectangular’ feature is computed by summing up pixel values within rectangular regions and then differencing those region sums

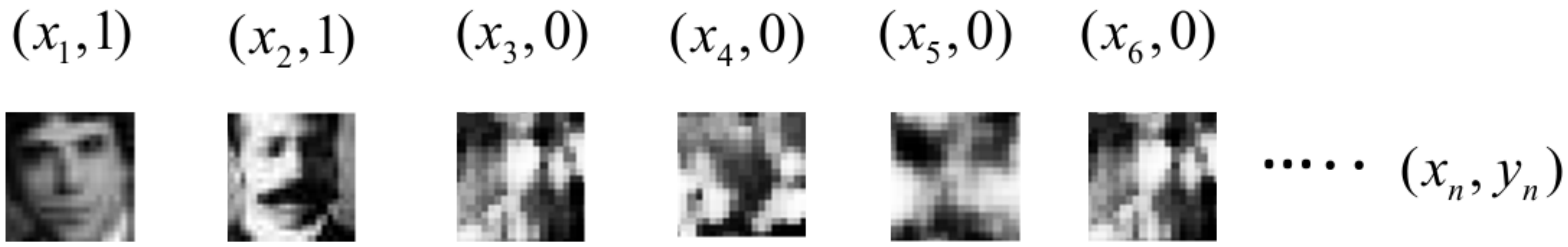


a.k.a. **Harr** Wavelets

Figure credit: P. Viola and M. Jones, 2001

Example: Face Detection

Training Dataset:



Faces

Not-faces

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

Evaluate a Harr Wavelet filter on each training example

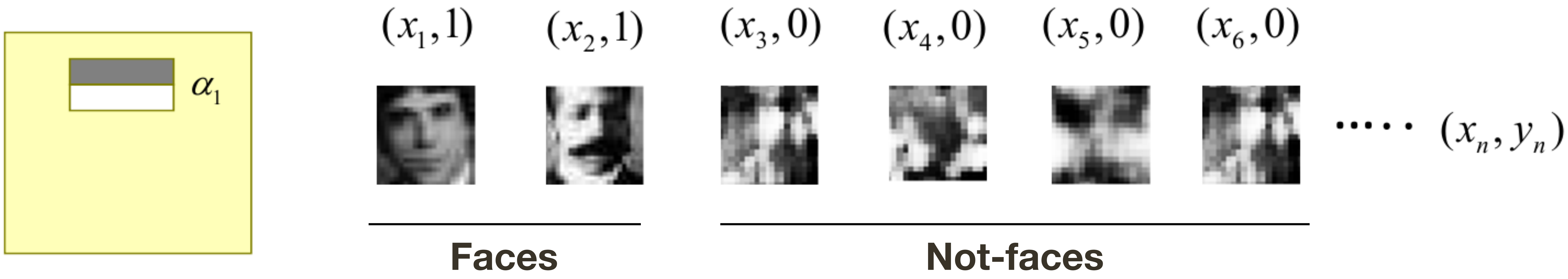
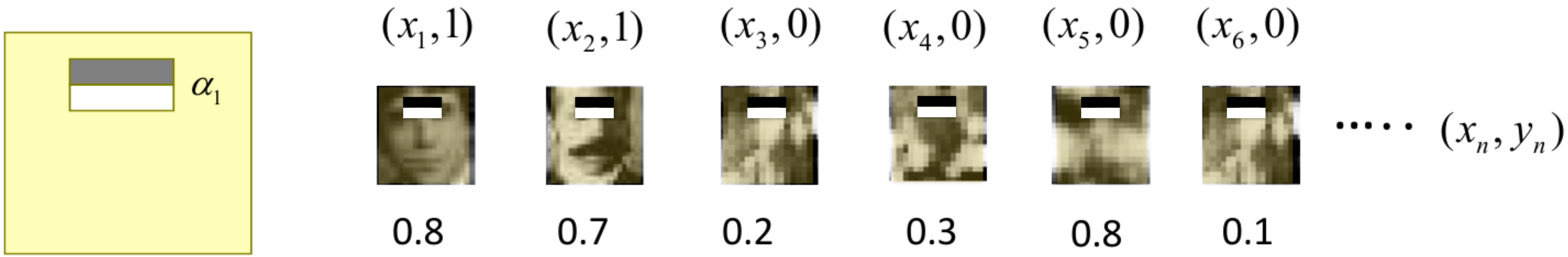


Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

Evaluate a Harr Wavelet filter on each training example



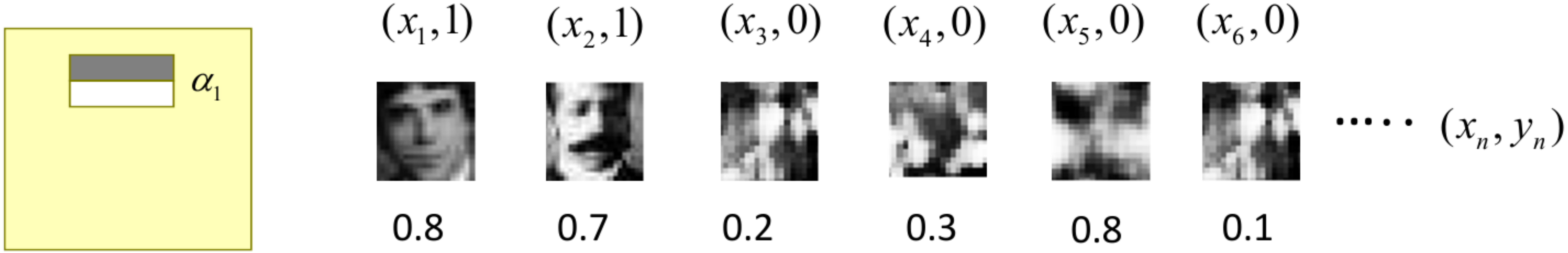
We can build a simple classifier by just selecting a threshold on the filter response (e.g. Harr filter response > 0.6 = face; Harr filter response <= 0.6 = not face)

Note: it is easy to find an **optimal** threshold. Just requires linear search over training example responses.

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

Evaluate a Harr Wavelet filter on each training example



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

threshold

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

Evaluate a Harr Wavelet filter on each training example

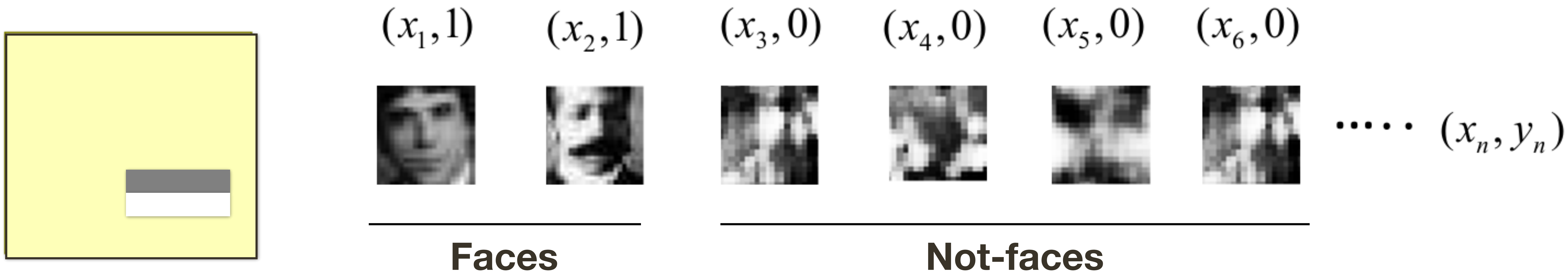
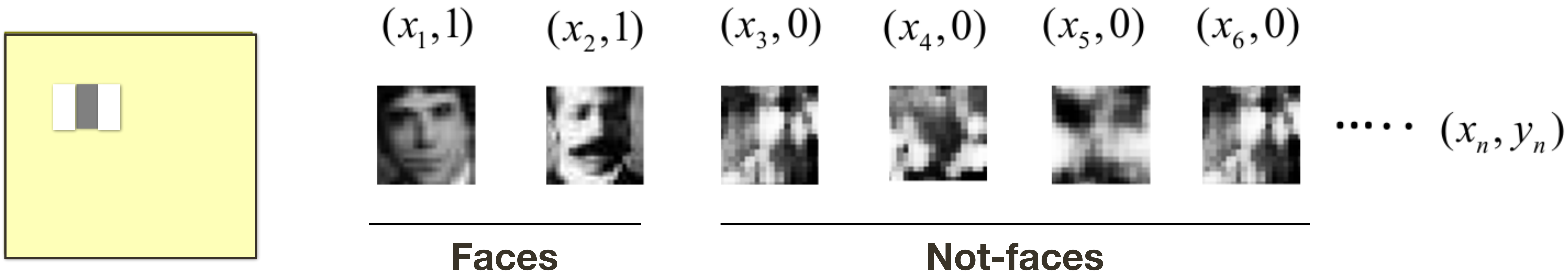


Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

Evaluate a Harr Wavelet filter on each training example



Note: we can easily compare different Harr Wavelet features under their individual best thresholds to see is the <u>most</u> informative (has highest classification)

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

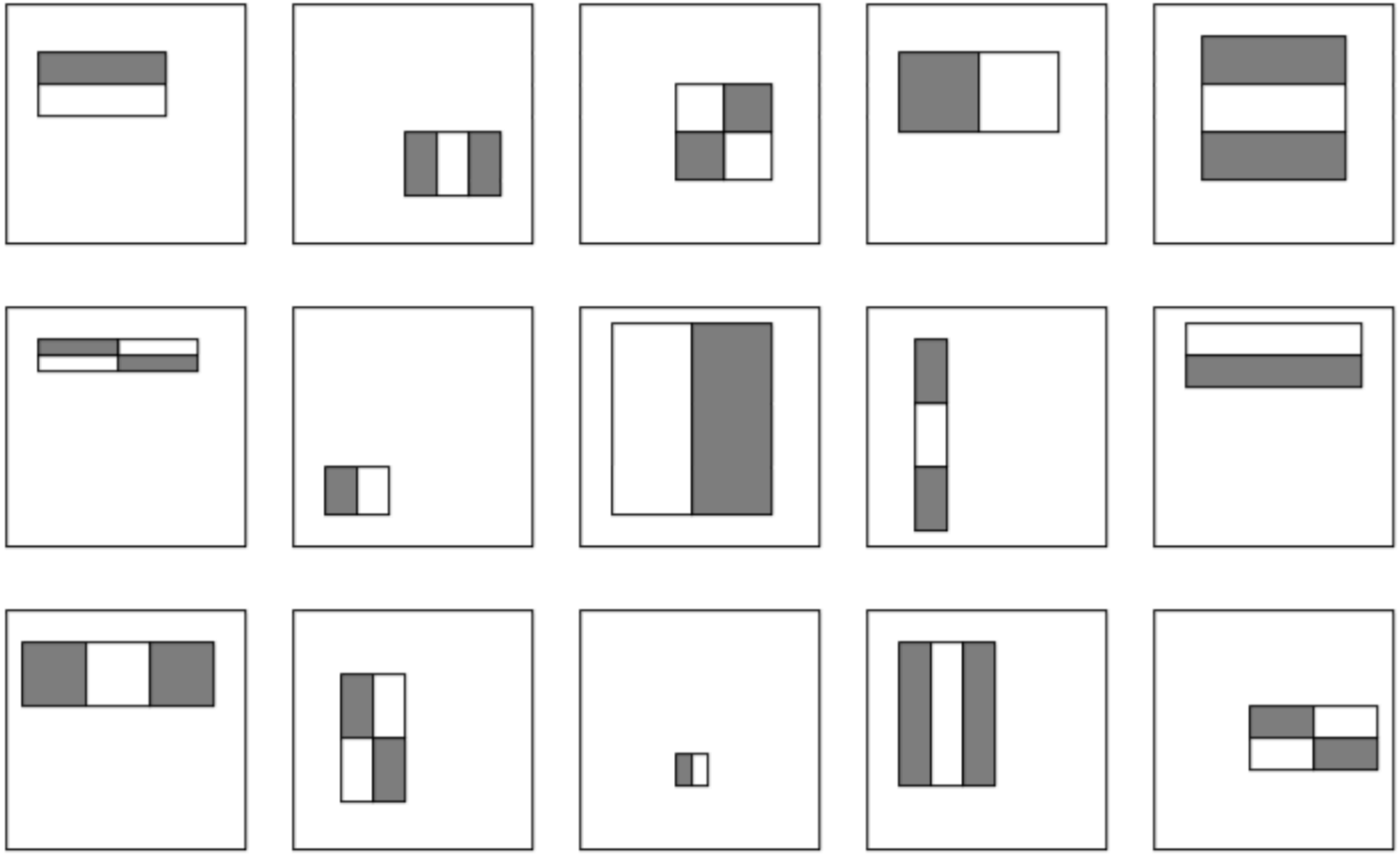
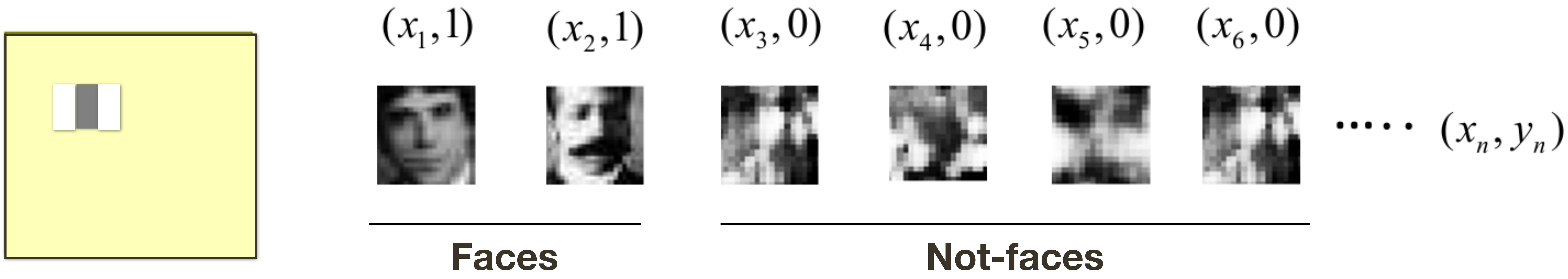


Figure credit: B. Freeman

Many possible rectangular features (180,000+ were used in the original paper)

Example: Face Detection

Evaluate a Harr Wavelet filter on each training example



Note: we can easily compare different Harr Wavelet features under their individual best thresholds to see is the <u>most</u> informative (has highest classification)

However, no one feature is likely to be good enough

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

Use **boosting** to both select the informative features and form the classifier. Each round chooses a weak classifier that simply compares a single rectangular feature against a threshold

Figure credit: P. Viola and M. Jones, 2001

Example: Face Detection

1. Select best filter/threshold combination

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

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

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

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

2. Re-weight examples

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

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$$

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

1. Select best filter/threshold combination

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

We start with all sample weights = 1

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

1. Select best filter/threshold combination

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

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

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

weighed sum of miss-classified training examples

Note: the second term is 0/1

— 0 predicted label and true label are same

— 1 predicted label and true label are different (error)

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

1. Select best filter/threshold combination

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

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

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

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

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

1. Select best filter/threshold combination

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

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

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

   c. Choose the classifier, ***h***$_t$ with the lowest error ε_t

2. Re-weight examples

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

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

Case 1: Classification for the sample i is **correct**

$$\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i}\ \beta_t$$

Case 2: Classification for the sample i is **incorrect**

$$\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i}$$

2. Re-weight examples

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

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$$

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

Case 1: Classification for the sample i is **correct**

$$\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i}\ \beta_t$$

Case 2: Classification for the sample i is **incorrect**

$$\mathbf{w}_{t+1,i} = \mathbf{w}_{t,i}$$

Note: the Beta is < 1

2. Re-weight examples

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

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$$

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

1. Select best filter/threshold combination

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

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

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

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

2. Re-weight examples

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

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$$

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection

Viola & Jones algorithm

3. The final strong classifier is

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

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

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

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection Summary

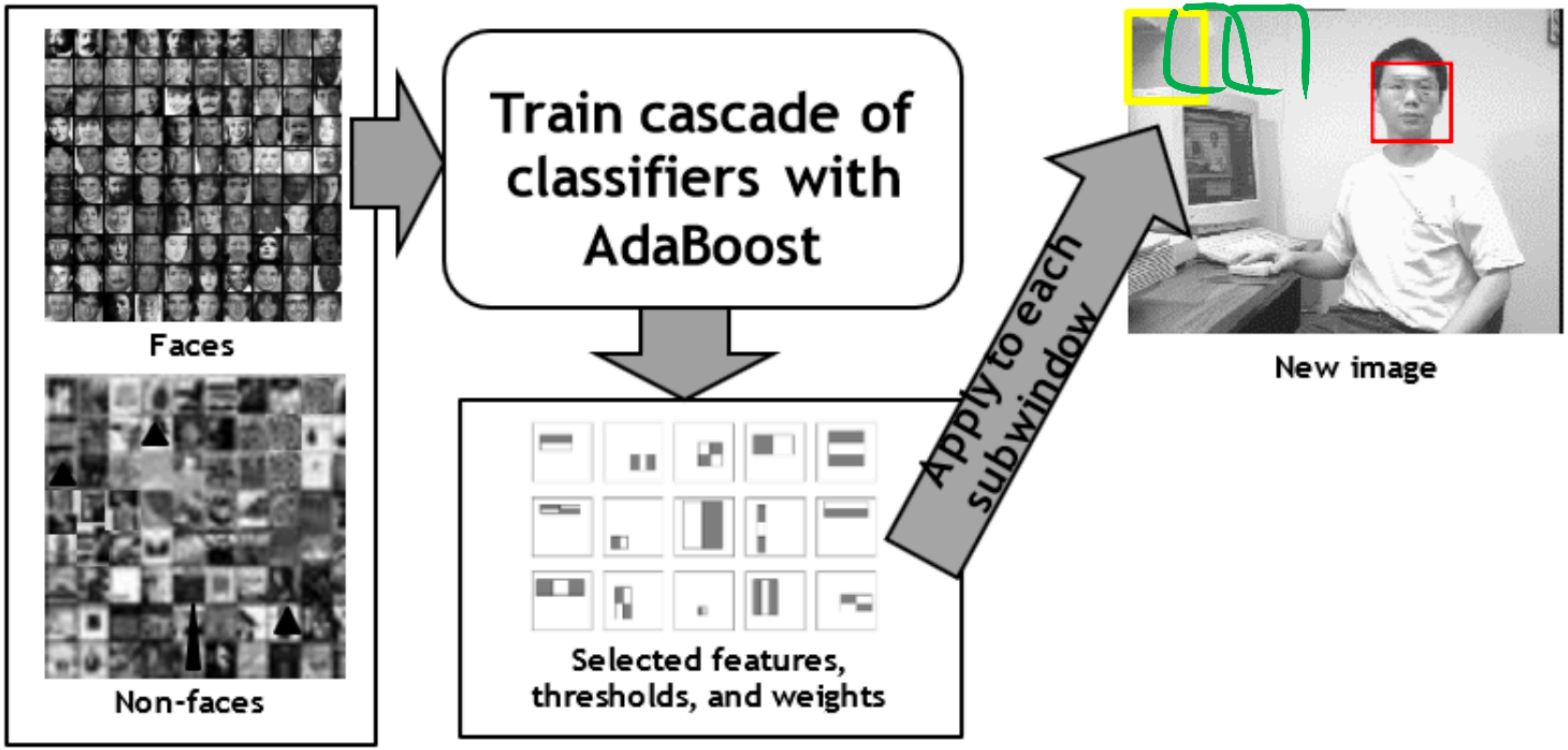
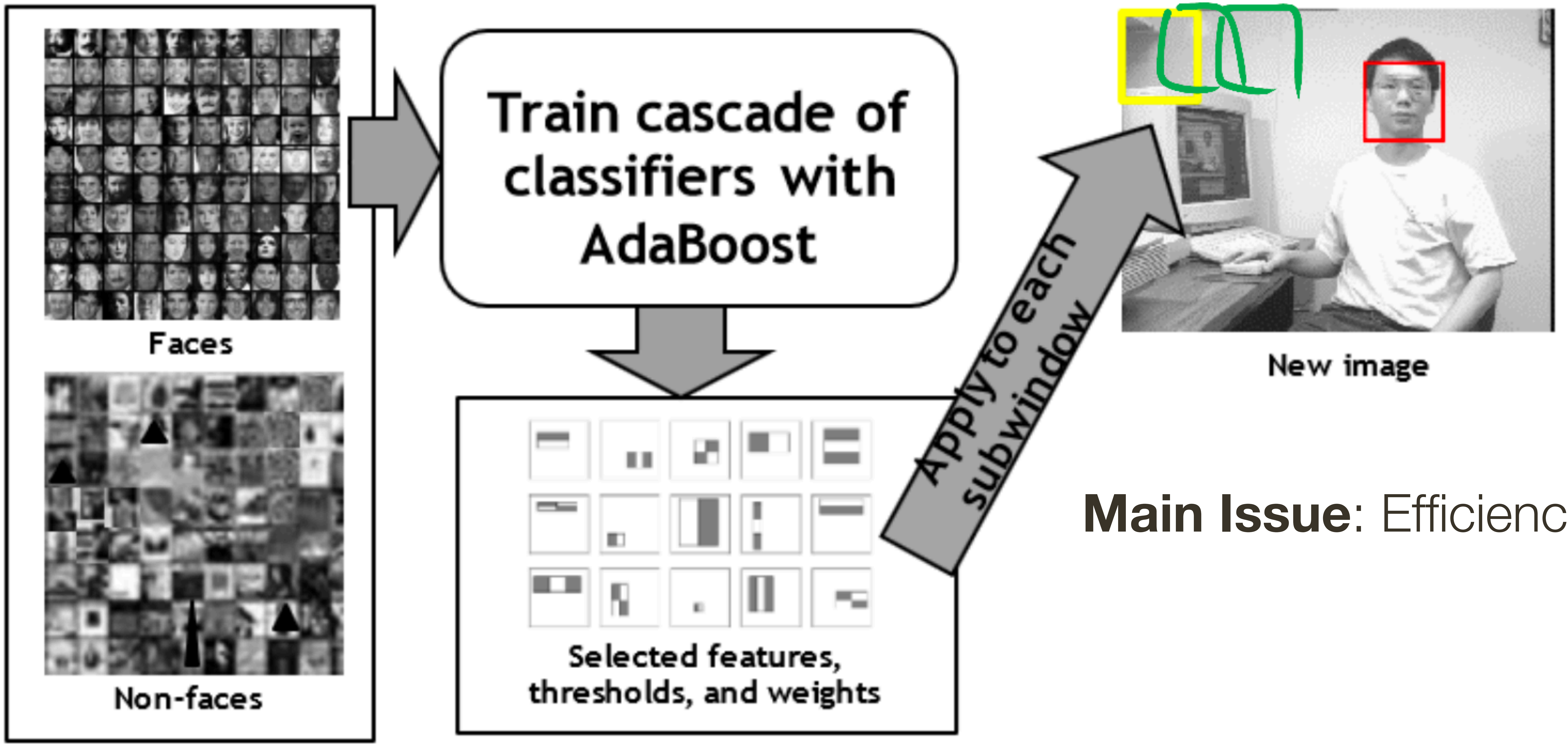


Figure credit: K. Grauman

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

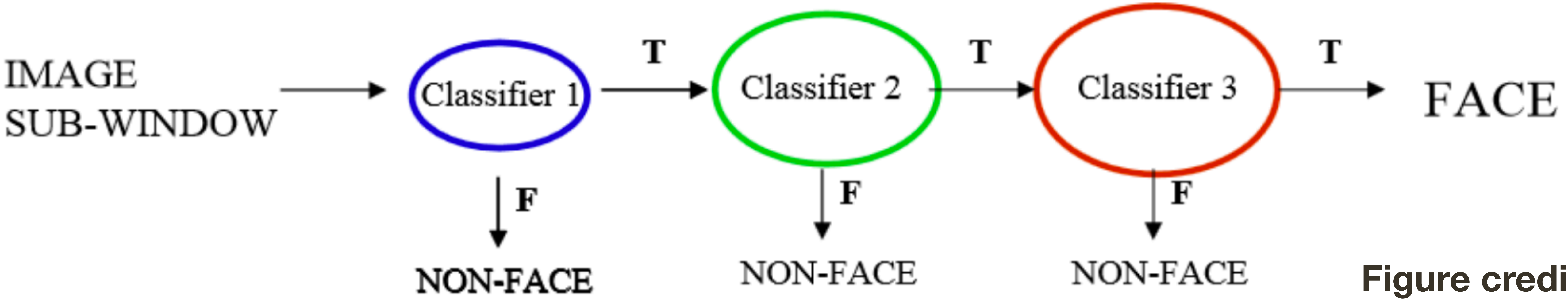


Figure credit: P. Viola

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

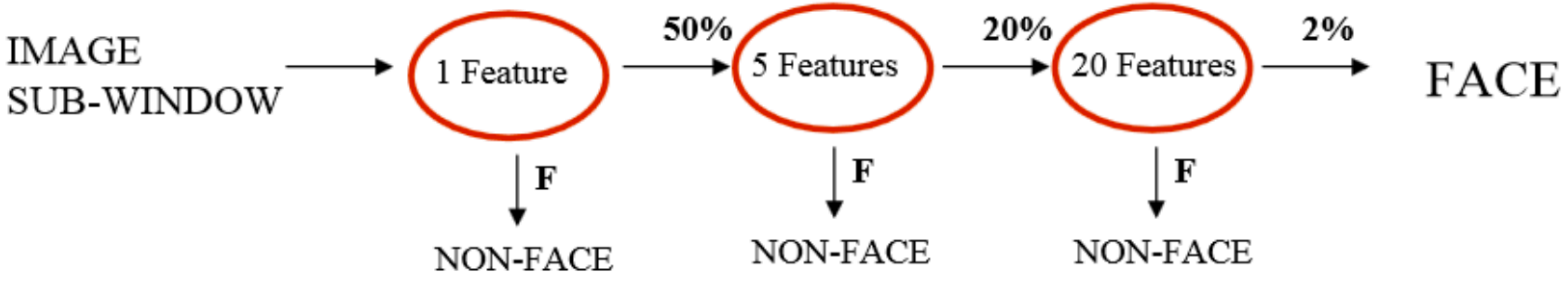


Figure credit: P. Viola

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

Example: Face Detection

Viola & Jones algorithm

3. The final strong classifier is

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

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

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

Image Credit: Ioannis (Yannis) Gkioulekas (CMU)

Example: Face Detection Summary

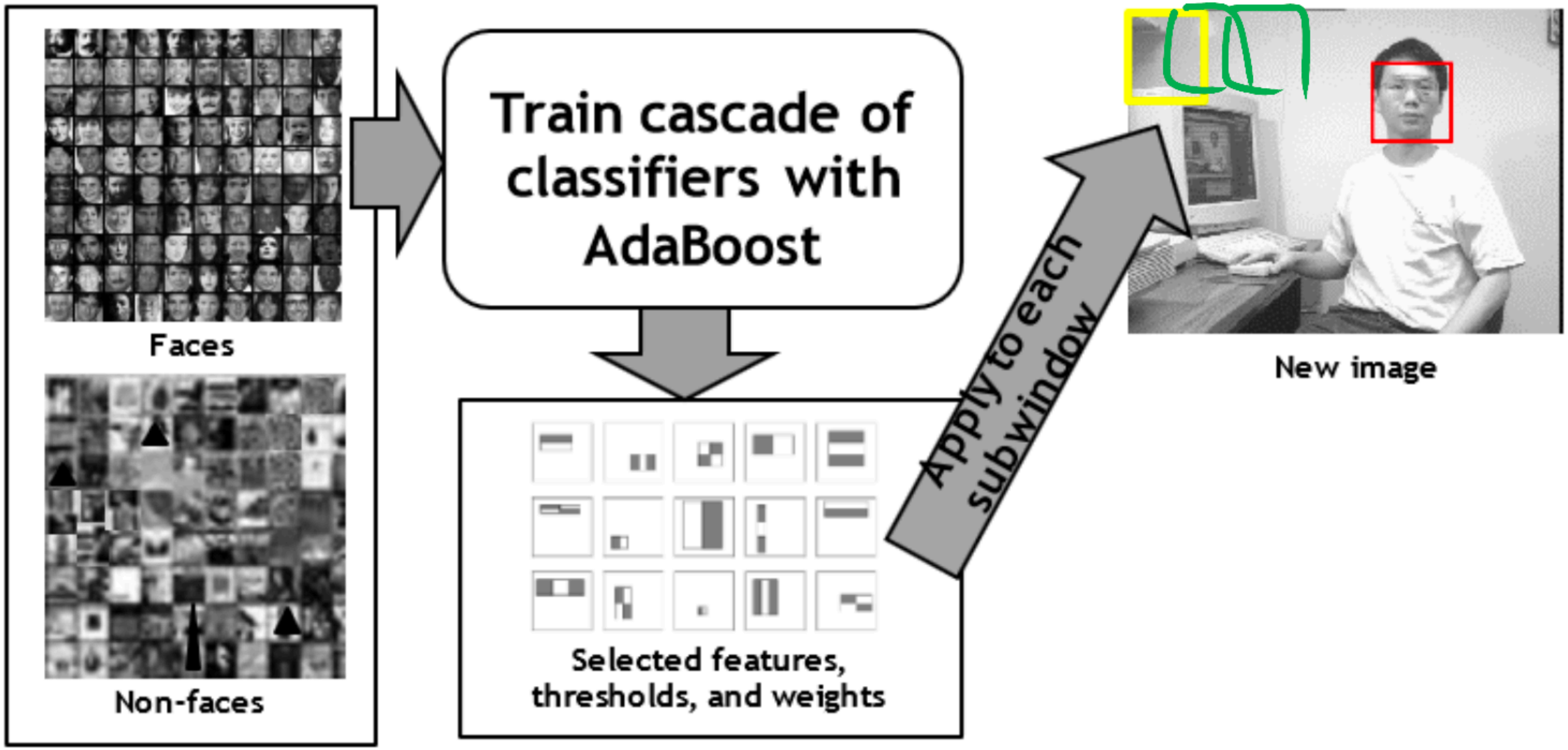
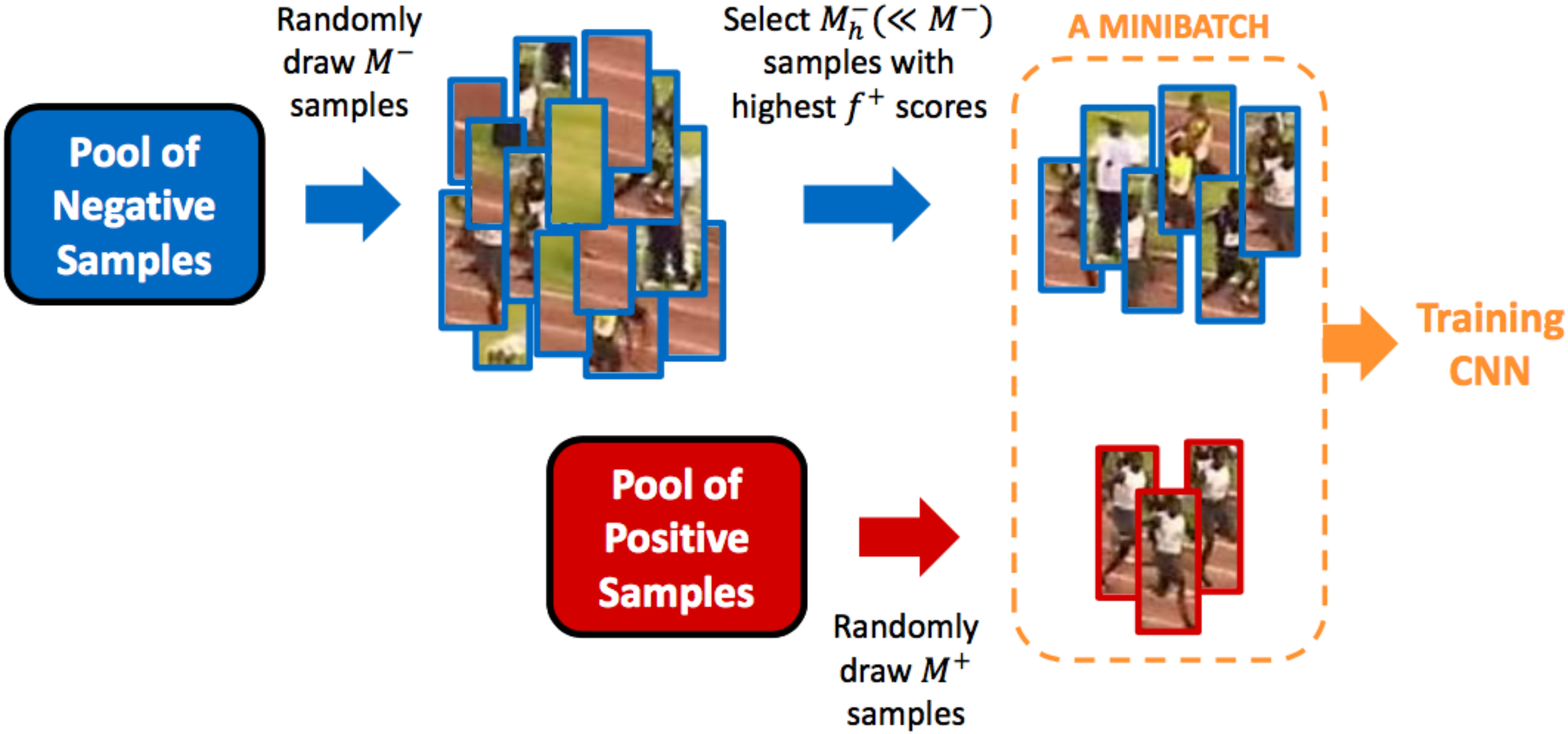


Figure credit: K. Grauman

Hard Negative Mining



Pool of Negative Samples

Randomly draw M^- samples

Select $M_h^- (\ll M^-)$ samples with highest f^+ scores

A MINIBATCH

Training CNN

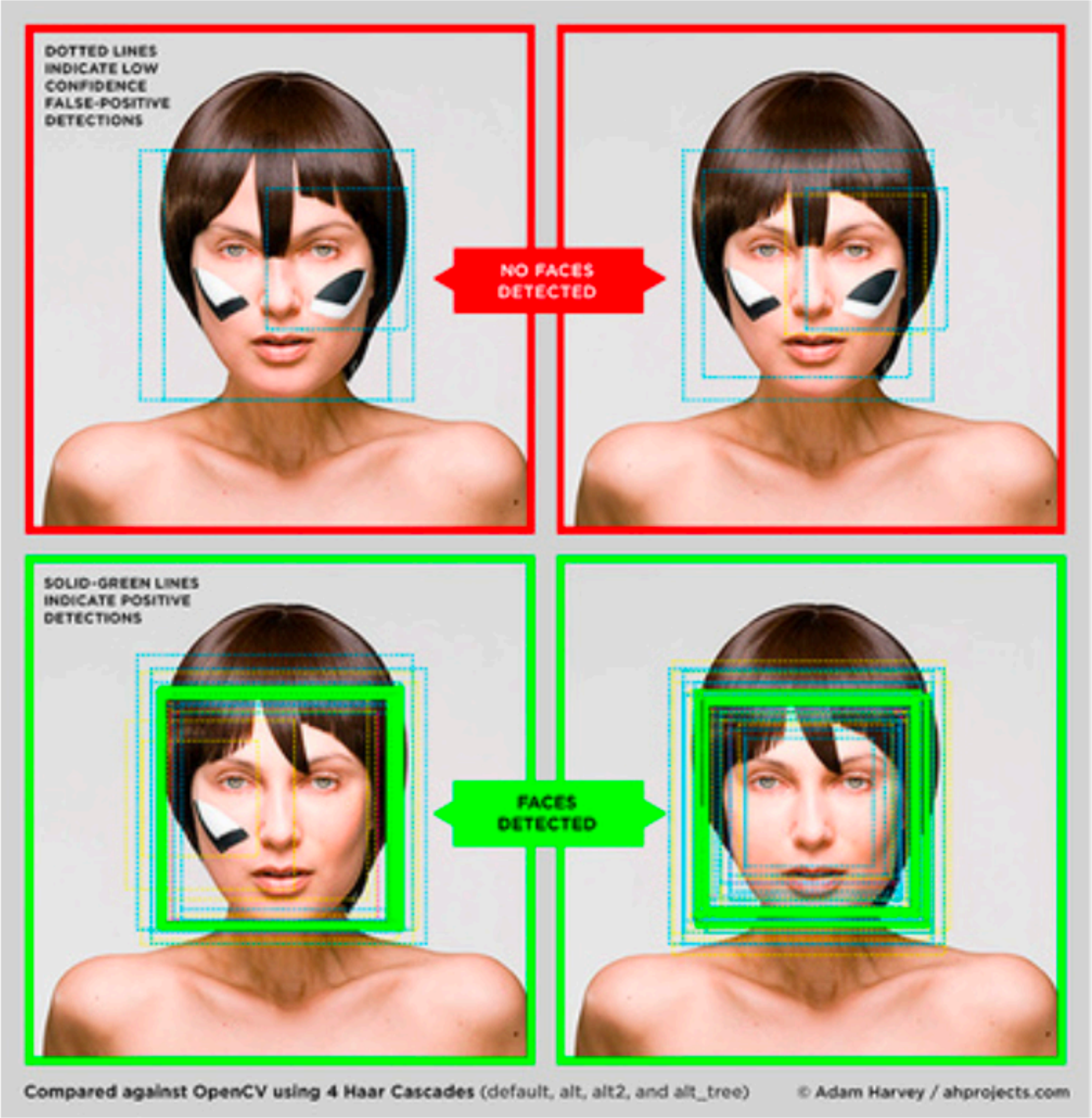
Pool of Positive Samples

Randomly draw M^+ samples

Image From: Jamie Kang

Example: Face Detection

Just for fun:



Compared against OpenCV using 4 Haar Cascades (default, alt, alt2, and alt_tree) © Adam Harvey / ahprojects.com

"CV Dazzle, a project focused on finding fashionable ways to thwart facial-recognition technology"

Figure source: Wired, 2015

Pedestrian Detection

The sliding window approach applies naturally to pedestrian detection because pedestrians tend to take characteristic poses, (e.g. standing, walking)

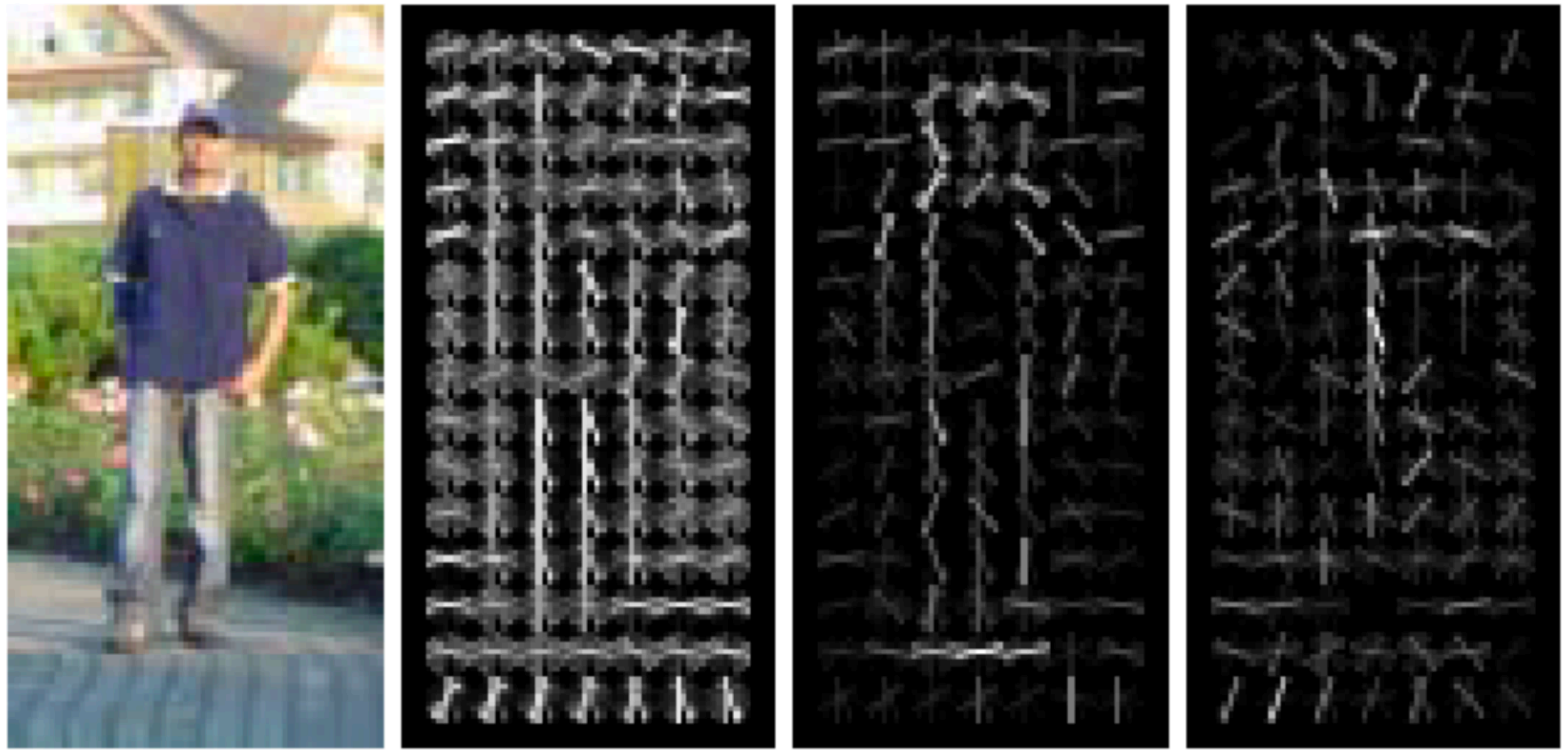
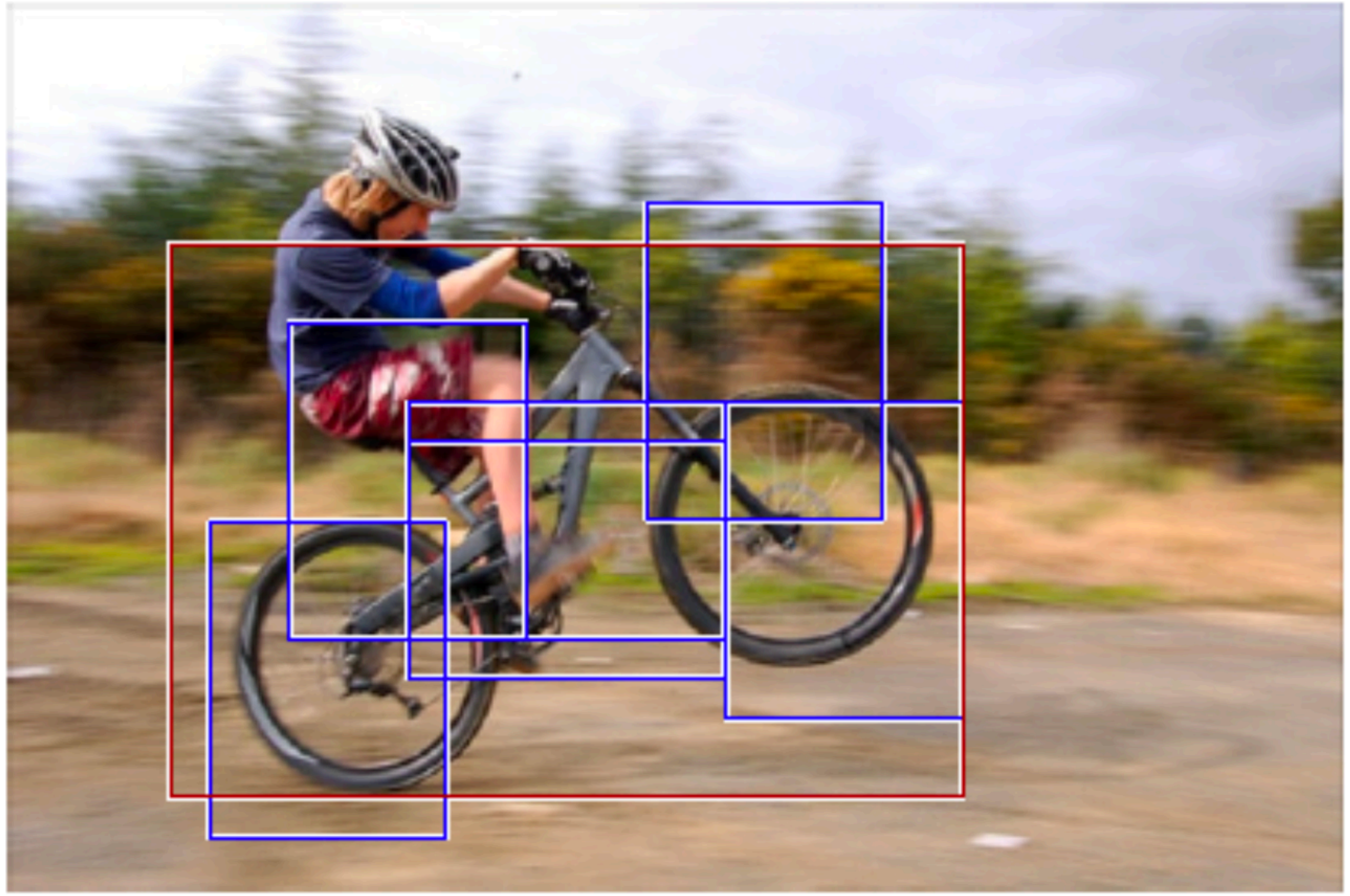


Image window; Visualisation of HOG features; HOG features weighted by positive weights; HOG features weighted by negative weights

Fig. 17.7 in Forsyth & Ponce (2nd ed). Original source: Dalal and Triggs, 2005.

Deformable Part Model

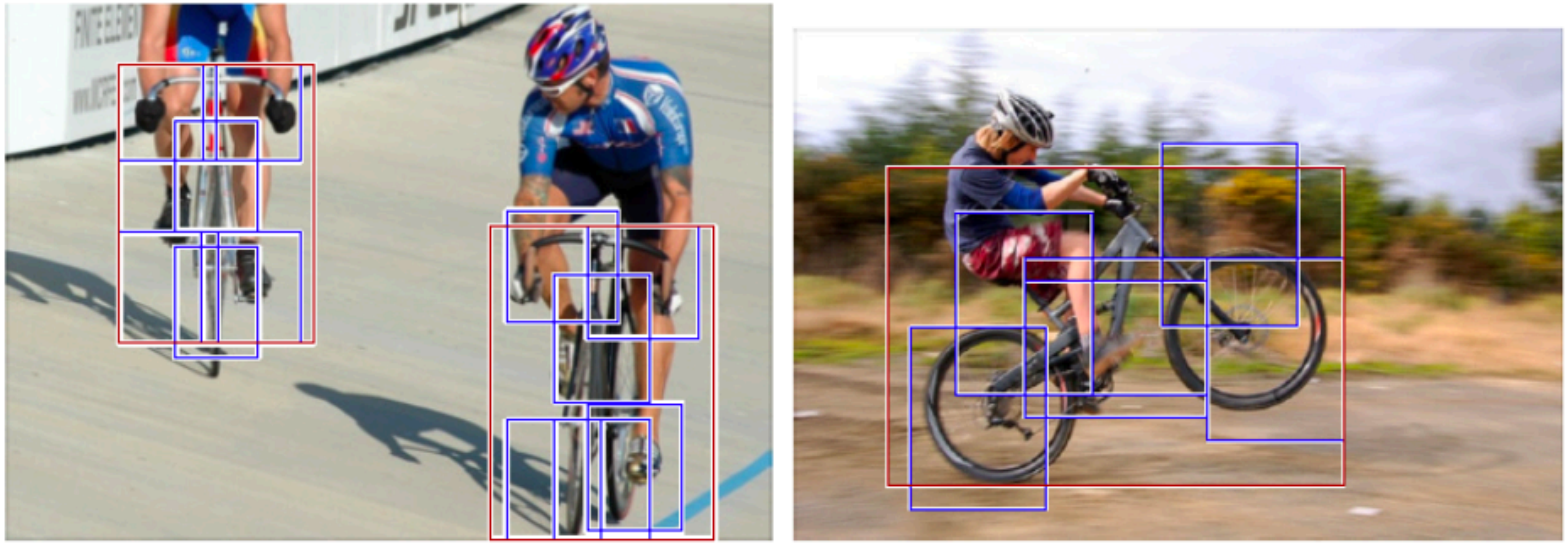
Sliding window detectors tend to fail when the object is not well described by a rigid template



Felzenszwalb et al., 2010

Many complex objects are better represented using a parts model

Deformable Part Model



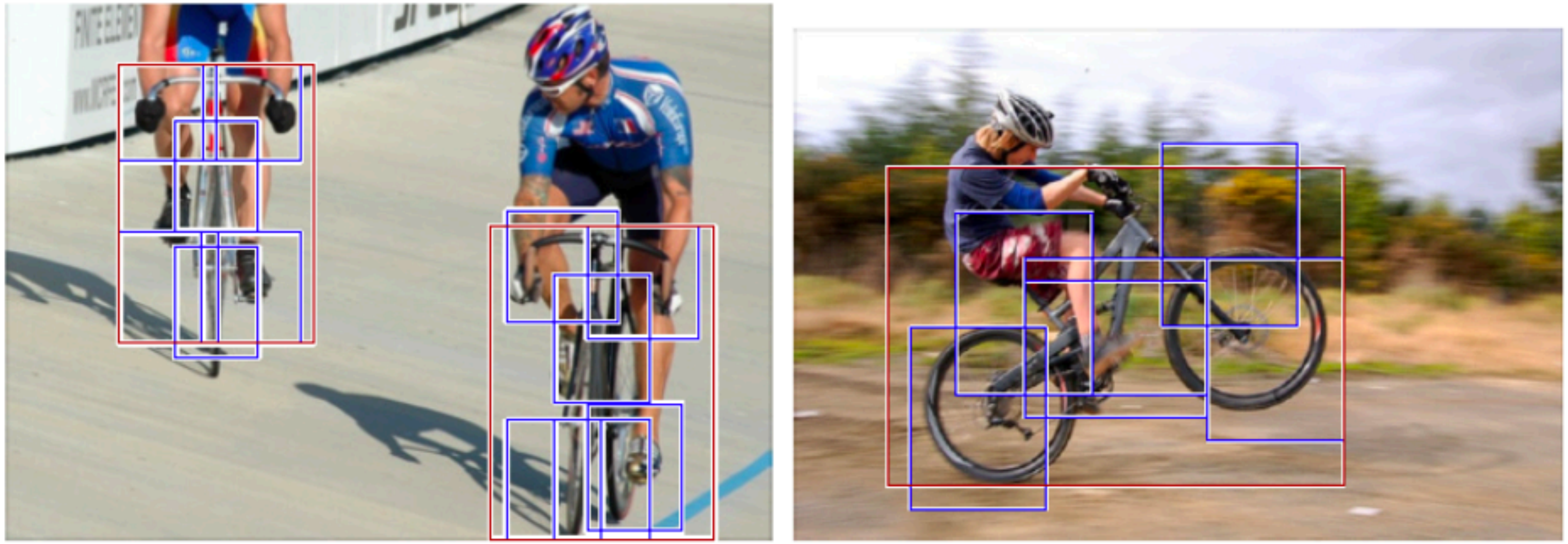
Felzenszwalb et al., 2010

A **deformable part model** consists of a root and a set of parts

— **Root**: an approximate model that gives the overall location of the object

— **Parts**: object components that have reliable appearance but might appear at somewhat different locations on the root for different instances

Deformable Part Model

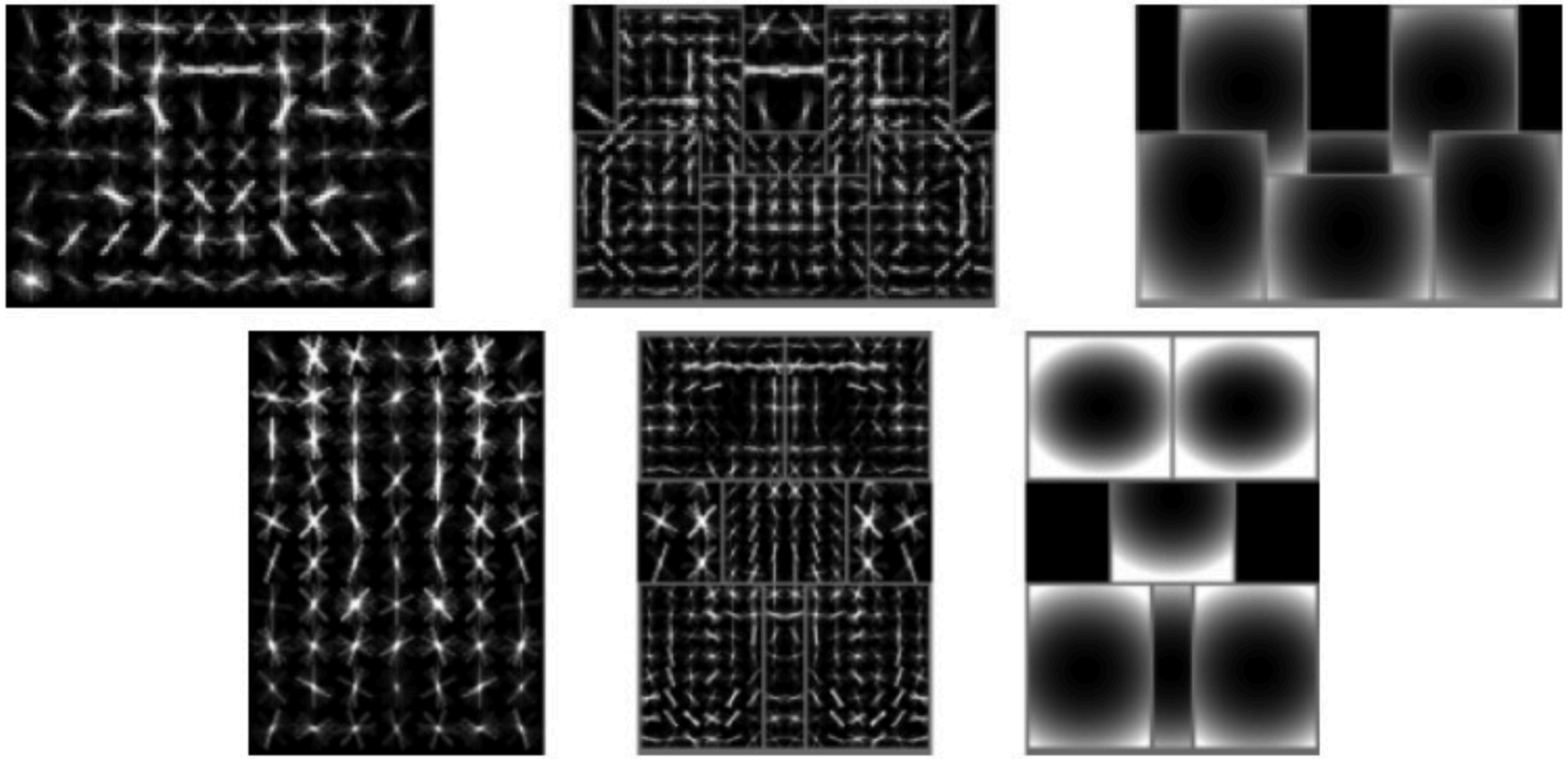


Felzenszwalb et al., 2010

Each part has an appearance model and a natural location relative to the root

Finding a window that looks a lot like the part close to that part's natural location relative to the root yields evidence that the object is present

Deformable Part Model



A parts model for a bicycle, containing a root and 6 parts

Figure source: Felzenszwalb et al., 2010

Deformable Part Model

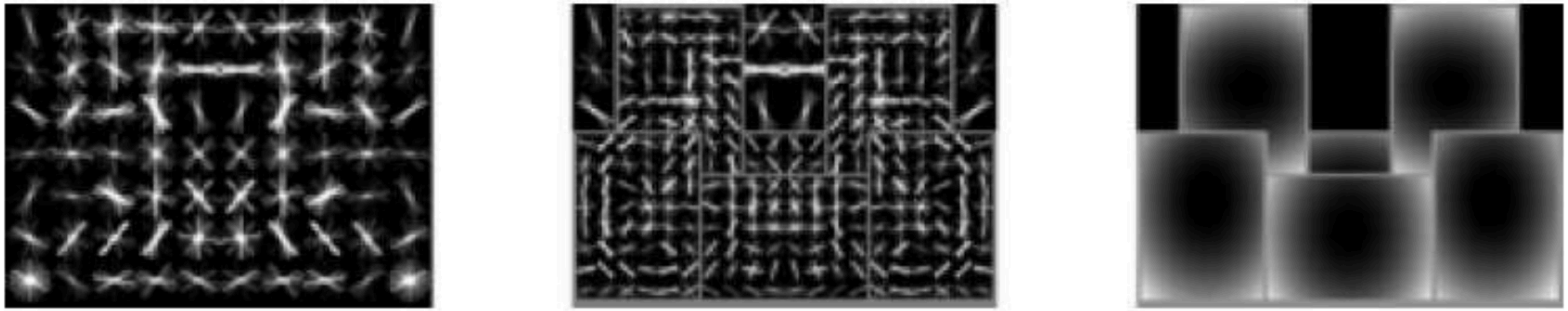


Figure source: Felzenszwalb et al., 2010

The learned root model is a set of linear weights $\beta^{(r)}$ applied to the feature descriptor of the root window

The i-th learned part model consists of

— a set of linear weights $\beta^{(p_i)}$ applied to the feature descriptor of the part window

— a natural location (offset) relative to the root $\mathbf{v}^{(p_i)} = (u^{(p_i)}, v^{(p_i)})$

— a set of distance weights $\mathbf{d}^{(p_i)} = (d_1^{(p_i)}, d_2^{(p_i)}, d_3^{(p_i)}, d_4^{(p_i)})$

Sliding Window with Deformable Part Model

The overall score of the deformable parts model at a particular window will be the sum of several scores

— A root score compares the root to the window

— Each part has its own score, consisting of an appearance score and a location score

$$\text{Model score} = \text{Root score} + \sum_i \text{Part i score}$$

Sliding Window with Deformable Part Model

Denote by $\phi(x, y)$ the feature descriptor of a part window at offset (x, y) relative to the root.

Denote by $(dx, dy) = (u^{(p_i)}, v^{(p_i)}) - (x, y)$ the difference from the part's natural offset relative to the root.

The score for part i at offset (x, y) is given by:

$$S^{(p_i)}(x, y; \beta^{(p_i)}, \mathbf{d}^{(p_i)}, \mathbf{v}^{(p_i)}) = \beta^{(p_i)}\phi(x, y) \\ - \left(d_1^{(p_i)} dx + d_2^{(p_i)} dy + d_3^{(p_i)} (dx)^2 + d_4^{(p_i)} (dy)^2\right)$$

Sliding Window with Deformable Part Model

Denote by $\phi(x, y)$ the feature descriptor of a part window at offset (x, y) relative to the root.

Denote by $(dx, dy) = (u^{(p_i)}, v^{(p_i)}) - (x, y)$ the difference from the part's natural offset relative to the root.

The score for part i at offset (x, y) is given by:

$$S^{(p_i)}(x, y; \beta^{(p_i)}, \mathbf{d}^{(p_i)}, \mathbf{v}^{(p_i)}) = \beta^{(p_i)}\phi(x, y) - \left(d_1^{(p_i)}dx + d_2^{(p_i)}dy + d_3^{(p_i)}(dx)^2 + d_4^{(p_i)}(dy)^2\right)$$

The final part i score is the best score found over all possible offsets (x, y)

Part *i* score = $max_{(x,y)}S^{(p_i)}(x, y; \beta^{(p_i)}, \mathbf{d}^{(p_i)}, \mathbf{v}^{(p_i)})$

Learning a Deformable Part Model

Learning the model can be tricky. Why?

Learning a Deformable Part Model

Learning the model can be tricky. Why?

A class model can consist of multiple component models representing different canonical views

— e.g. a front and lateral model of a bicycle

We do not know which component model should respond to which training example

Learning a Deformable Part Model

Learning the model can be tricky. Why?

A class model can consist of multiple component models representing different canonical views

— e.g. a front and lateral model of a bicycle

We do not know which component model should respond to which training example

We also do not know the locations of the parts in the training examples

Learning a Deformable Part Model

However, notice that if the component and the part locations for each training example are given (fixed), we can simply train a **linear SVM** as usual

Learning a Deformable Part Model

However, notice that if the component and the part locations for each training example are given (fixed), we can simply train a **linear SVM** as usual

This observation leads to the following iterative strategy:

— Assume components and part locations are given (fixed). Compute appearance and offset models.

— Assume appearance and offset models are given (fixed). Re-estimate components and part locations.

Deformable Part Models: **Hard Negative Mining**

Sliding window detectors must search over an immense number of windows
— Even a small false positive rate becomes noticeable

As a result, we want to train on as many negative examples as possible, but remain computationally feasible

Hard negative mining: As we train the classifier, apply it to the negative examples (e.g. ‘not a bicycle’) and keep track of ones that get a strong response (e.g. are mistakenly detected as bicycles). Include these in the next round of training.

Deformable Part Model: **Examples**

A learned car model

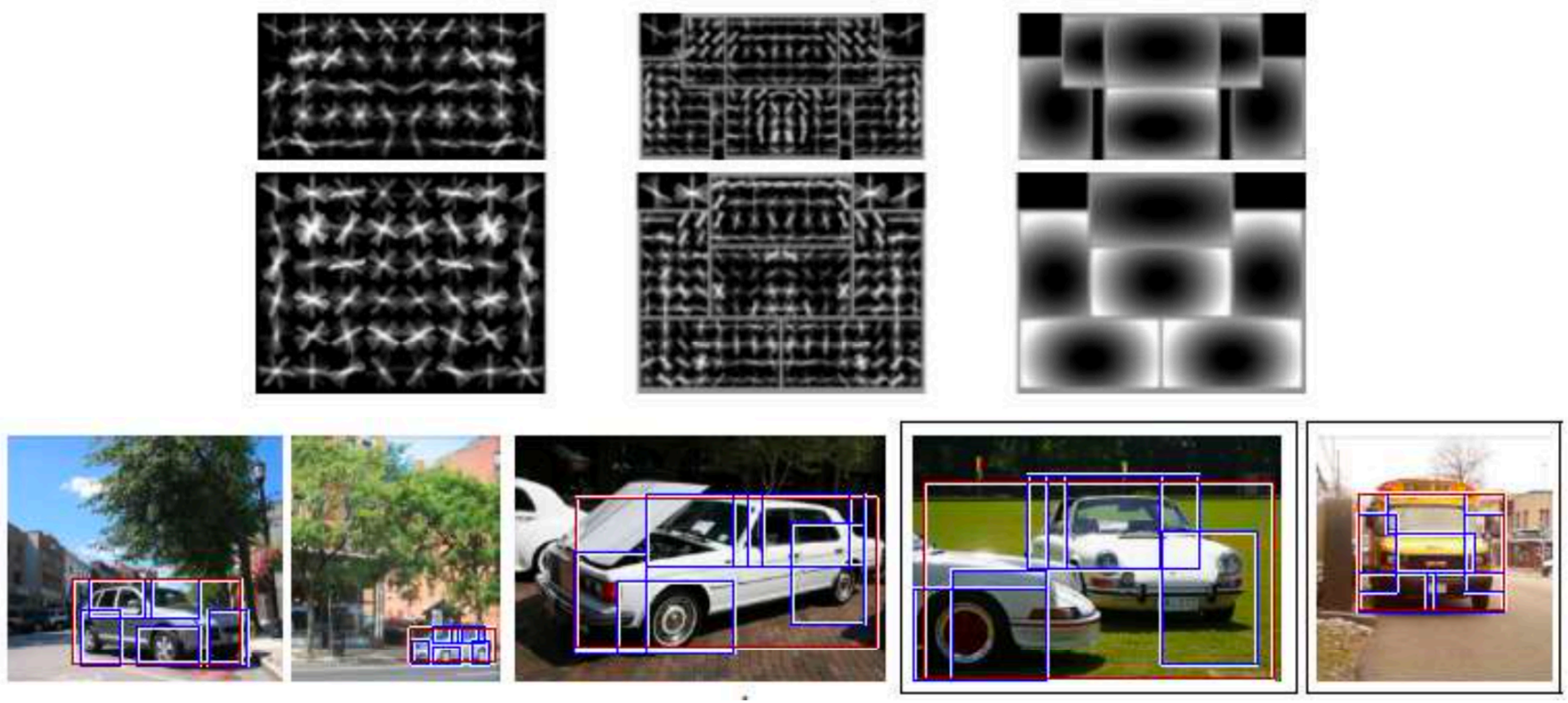
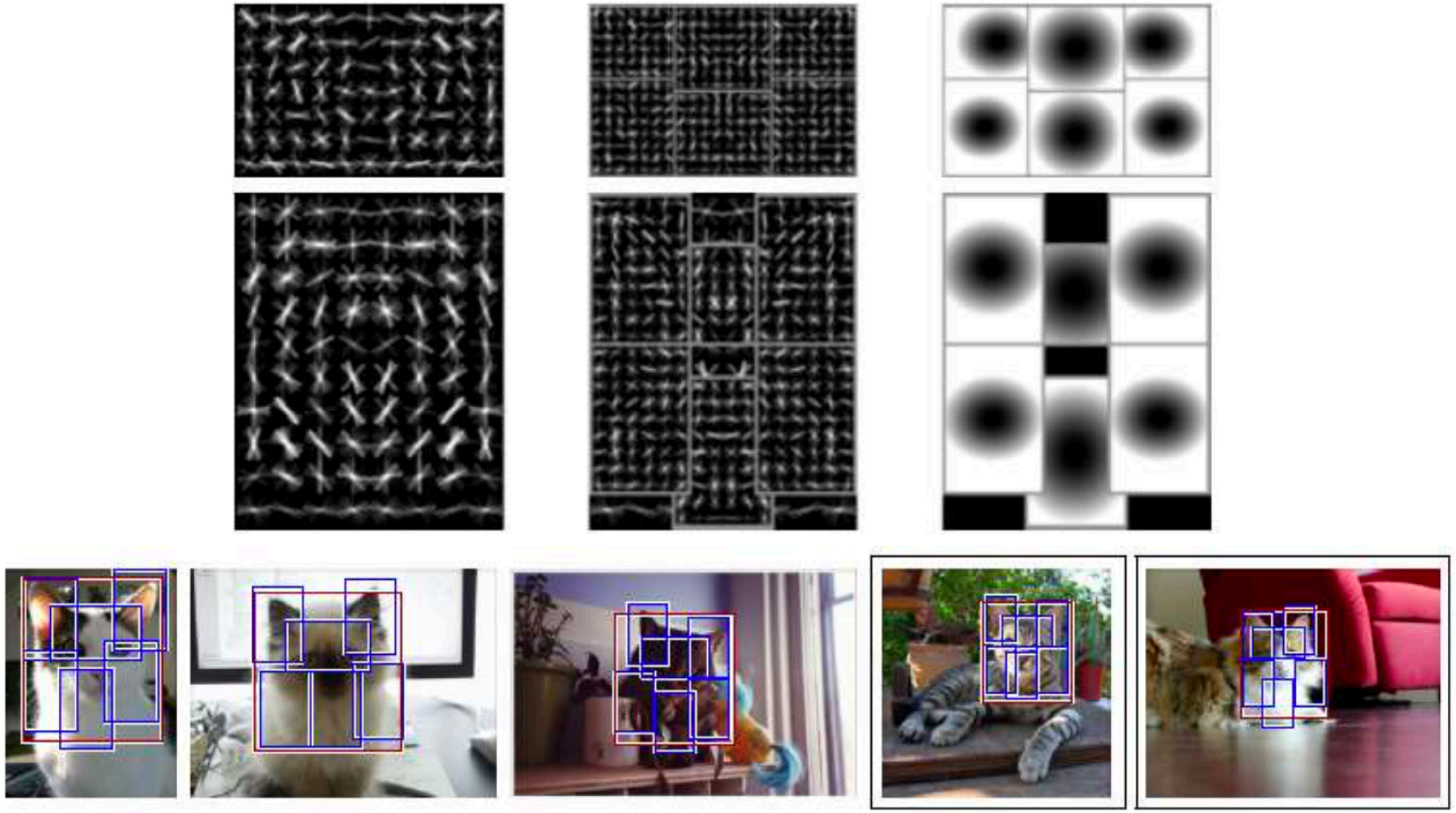


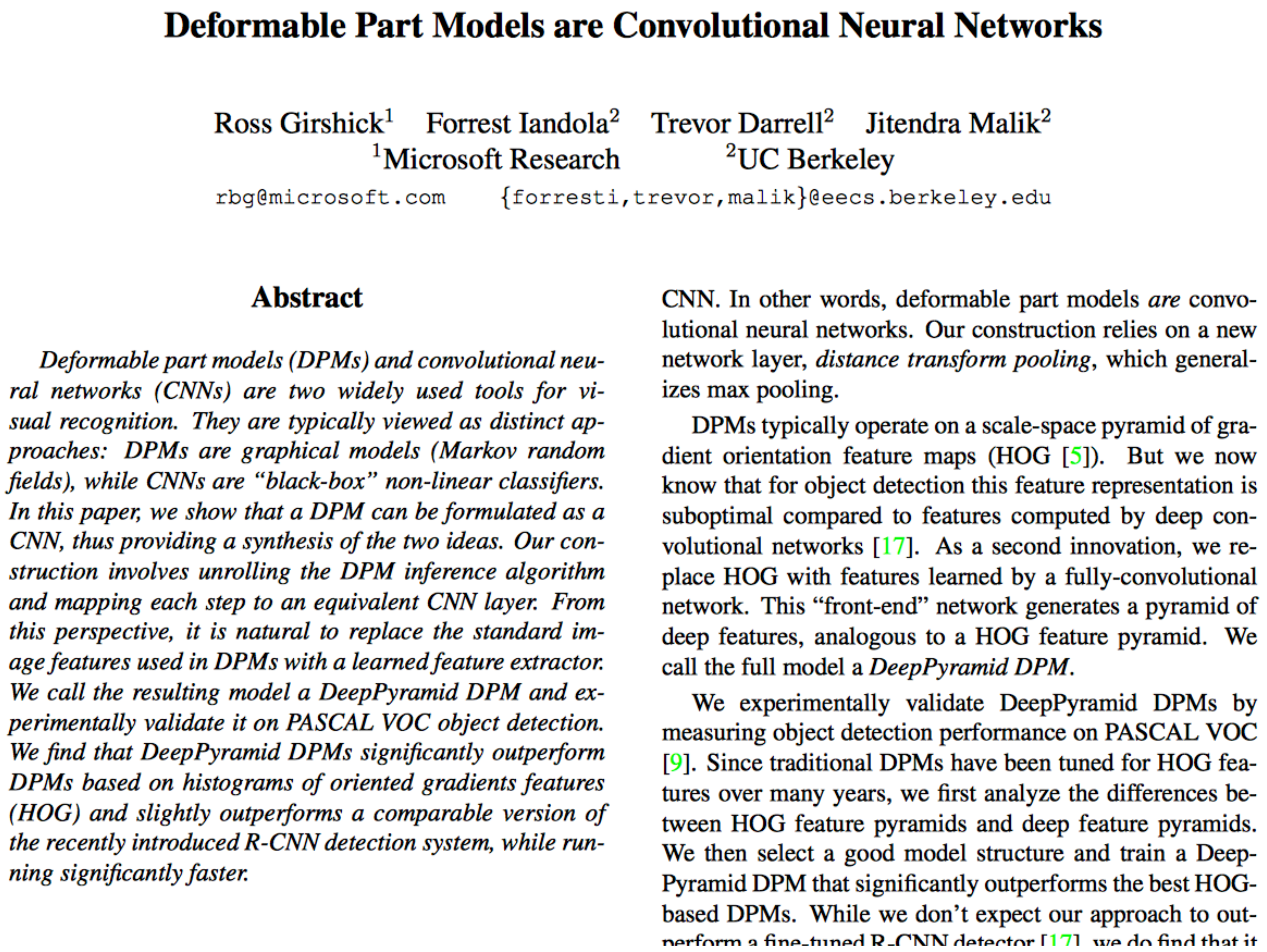
Figure source: Felzenszwalb et al., 2010

Deformable Part Model: **Examples**



A learned cat model

Figure source: Felzenszwalb et al., 2010



Deformable Part Models are Convolutional Neural Networks

Ross Girshick[1] Forrest Iandola[2] Trevor Darrell[2] Jitendra Malik[2]
[1]Microsoft Research [2]UC Berkeley
rbg@microsoft.com {forresti,trevor,malik}@eecs.berkeley.edu

Abstract

Deformable part models (DPMs) and convolutional neural networks (CNNs) are two widely used tools for visual recognition. They are typically viewed as distinct approaches: DPMs are graphical models (Markov random fields), while CNNs are "black-box" non-linear classifiers. In this paper, we show that a DPM can be formulated as a CNN, thus providing a synthesis of the two ideas. Our construction involves unrolling the DPM inference algorithm and mapping each step to an equivalent CNN layer. From this perspective, it is natural to replace the standard image features used in DPMs with a learned feature extractor. We call the resulting model a DeepPyramid DPM and experimentally validate it on PASCAL VOC object detection. We find that DeepPyramid DPMs significantly outperform DPMs based on histograms of oriented gradients features (HOG) and slightly outperforms a comparable version of the recently introduced R-CNN detection system, while running significantly faster.

CNN. In other words, deformable part models *are* convolutional neural networks. Our construction relies on a new network layer, *distance transform pooling*, which generalizes max pooling.

DPMs typically operate on a scale-space pyramid of gradient orientation feature maps (HOG [5]). But we now know that for object detection this feature representation is suboptimal compared to features computed by deep convolutional networks [17]. As a second innovation, we replace HOG with features learned by a fully-convolutional network. This "front-end" network generates a pyramid of deep features, analogous to a HOG feature pyramid. We call the full model a *DeepPyramid DPM*.

We experimentally validate DeepPyramid DPMs by measuring object detection performance on PASCAL VOC [9]. Since traditional DPMs have been tuned for HOG features over many years, we first analyze the differences between HOG feature pyramids and deep feature pyramids. We then select a good model structure and train a DeepPyramid DPM that significantly outperforms the best HOG-based DPMs. While we don't expect our approach to outperform a fine-tuned R-CNN detector [17], we do find that it

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

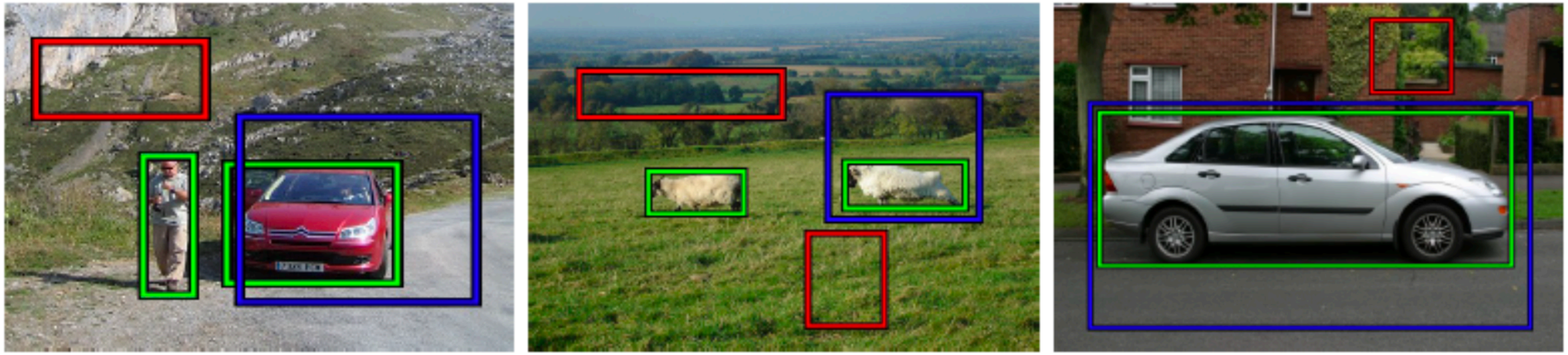


Figure credit: Alexe et al., 2012

Object Proposals

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

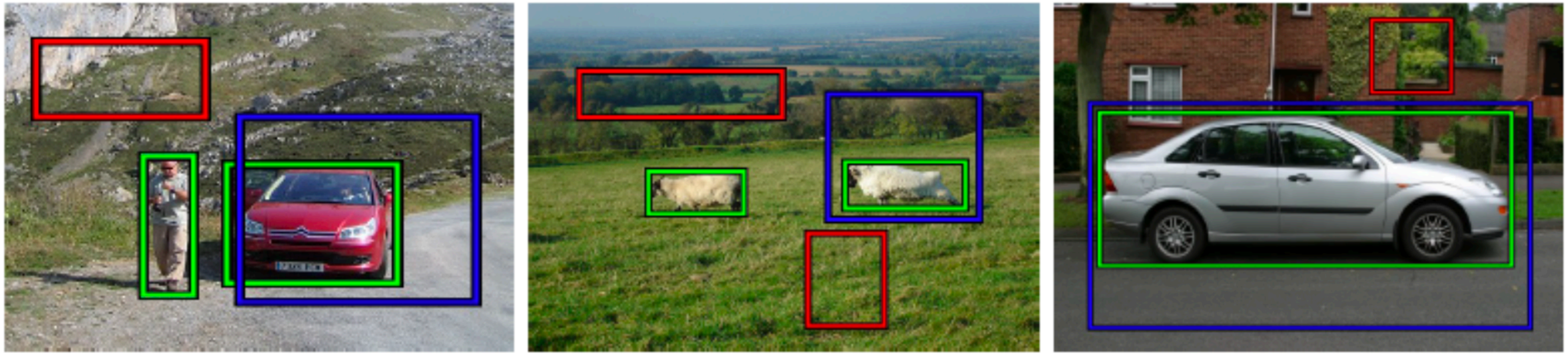


Figure credit: Alexe et al., 2012

This work argued that objects typically

— are unique within the image and stand out as salient

— have a contrasting appearance from surroundings and/or

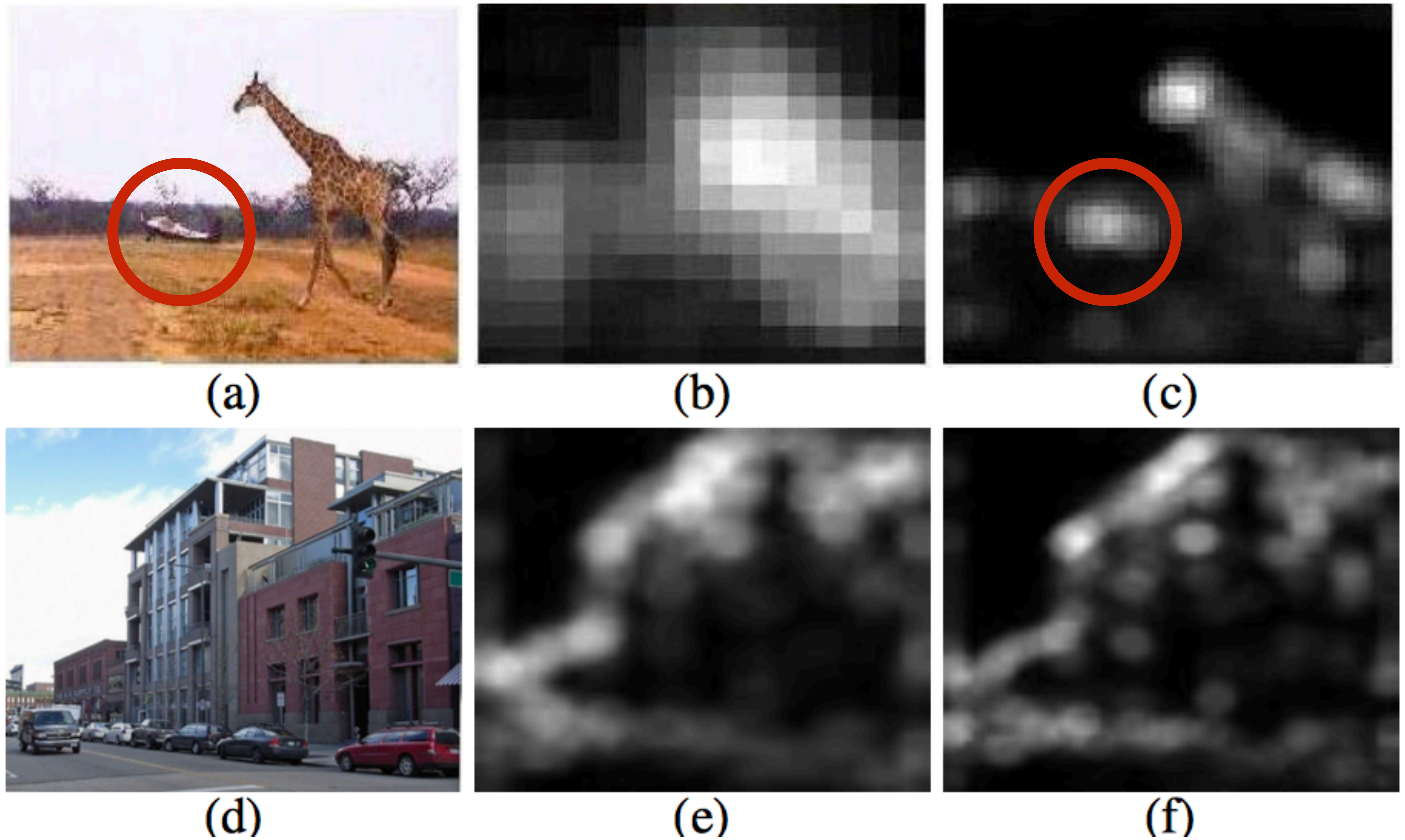
— have a well-defined closed boundary in space

Object Proposals

Multiscale **Saliency**

— Favors regions with a unique appearance within the image

High scale **Low** scale



Successful Case

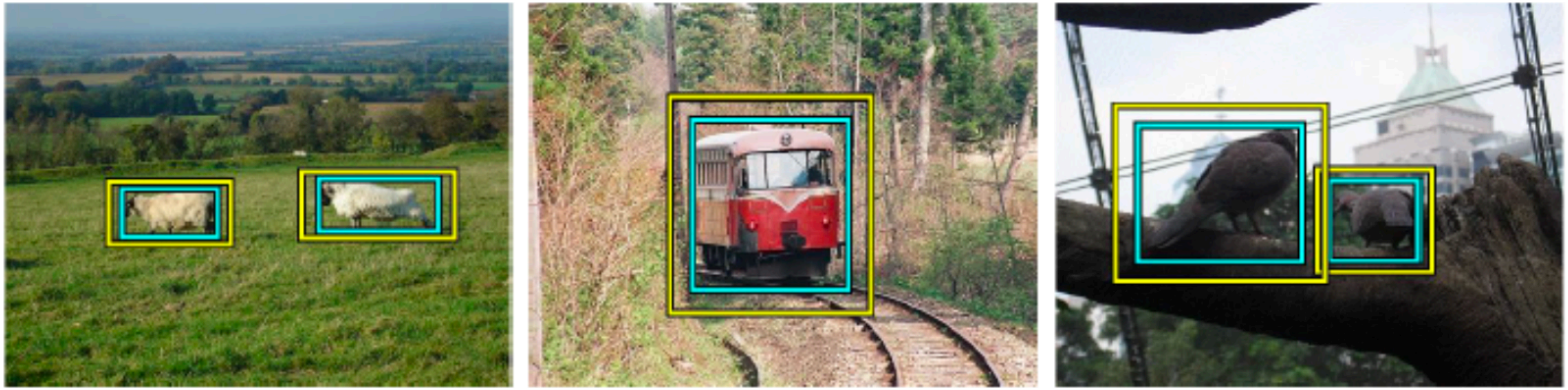
Failure Case

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

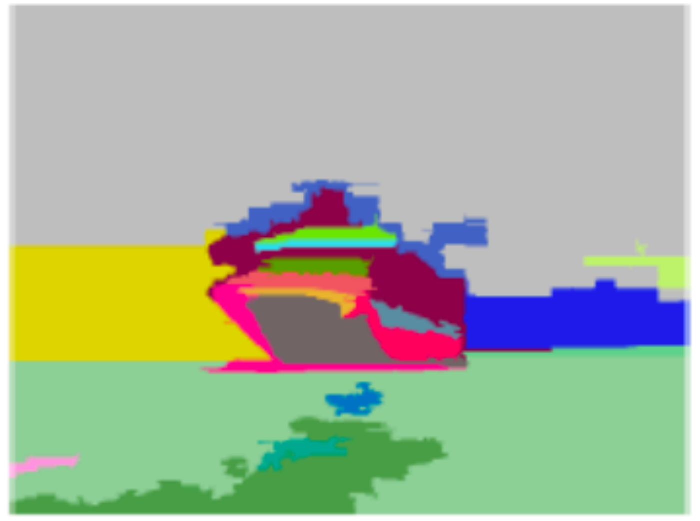
Object Proposals

Superpixels Straddling

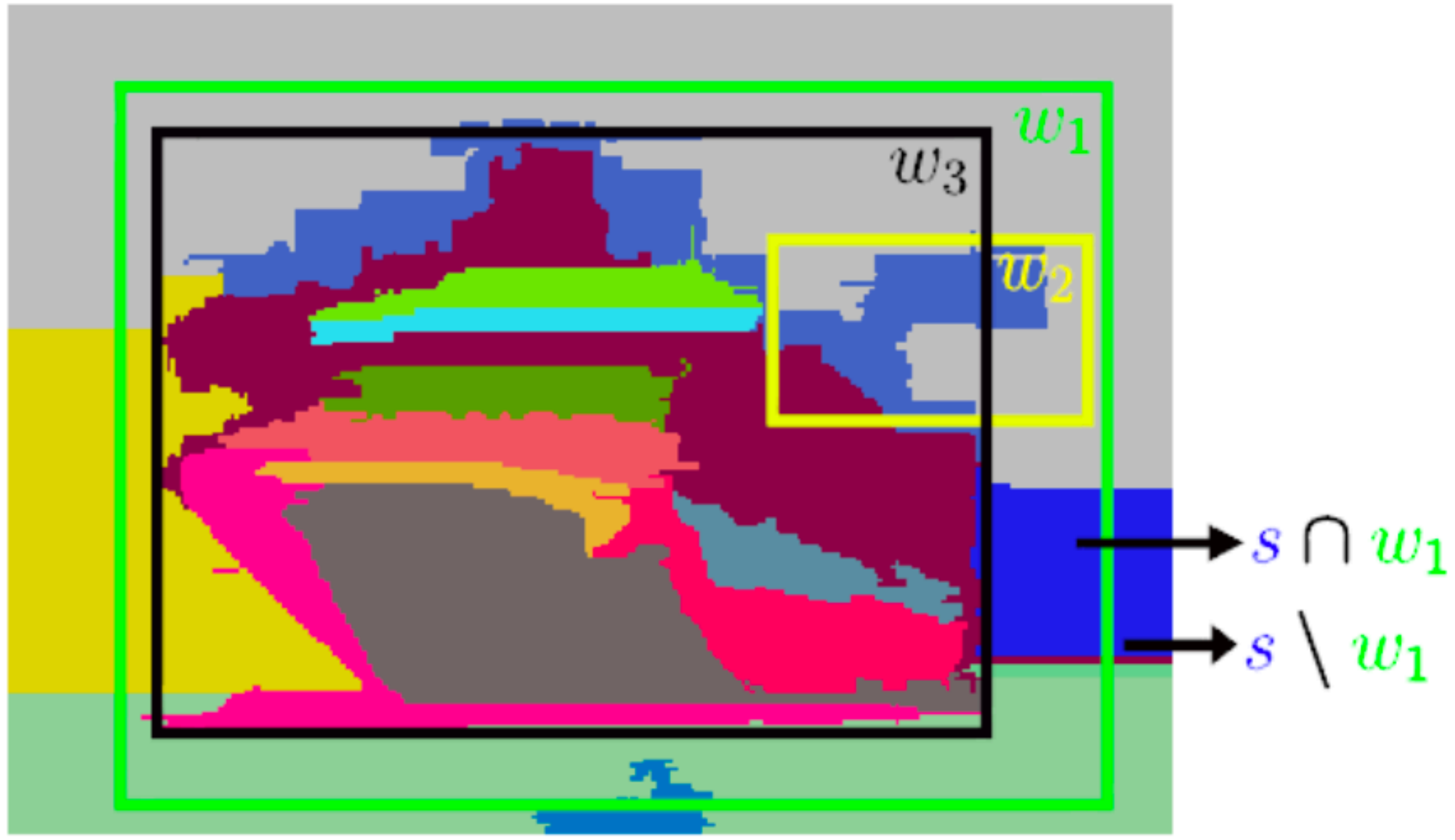
— Favors regions with a well-defined closed boundary

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

(a)



(b)



(c)

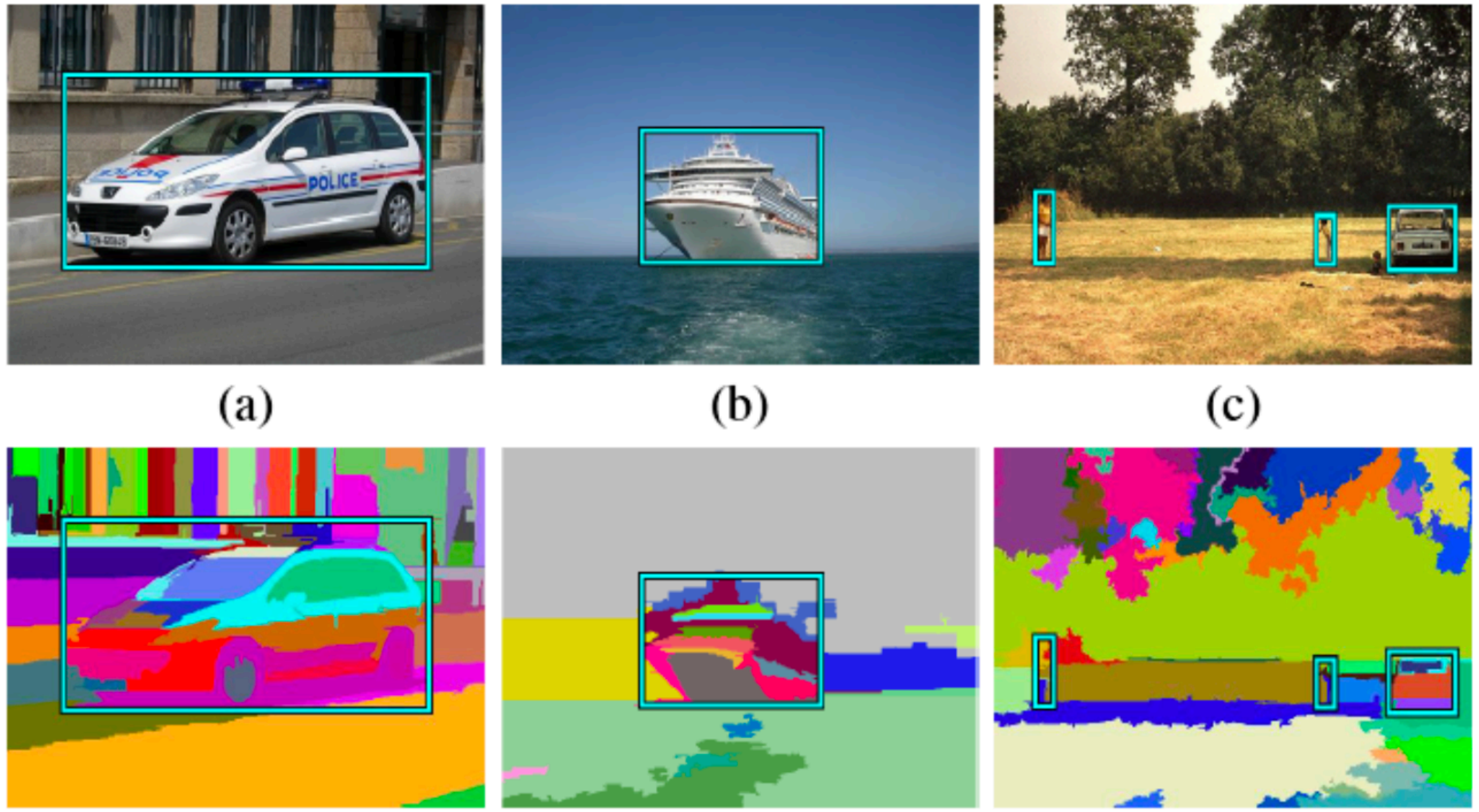
Figure credit: Alexe et al., 2012

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

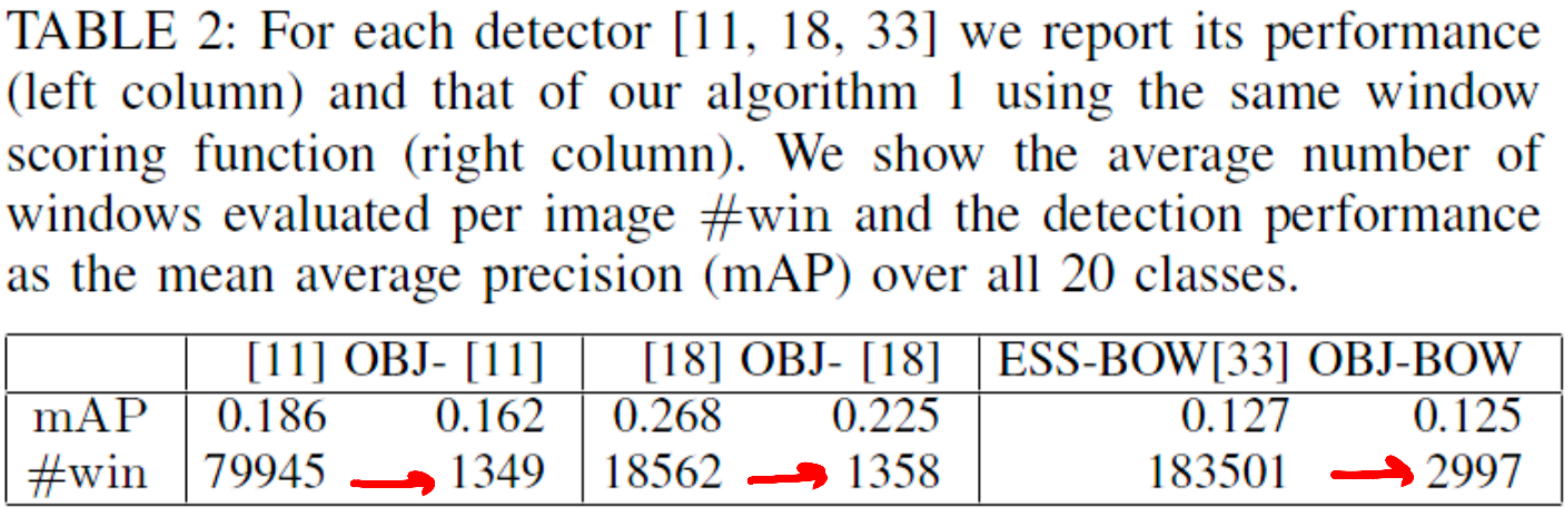


TABLE 2: For each detector [11, 18, 33] we report its performance (left column) and that of our algorithm 1 using the same window scoring function (right column). We show the average number of windows evaluated per image #win and the detection performance as the mean average precision (mAP) over all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