Exploiting and Introducing Parallelism for Efficient Object Detection

Mario Fritz
Rodrigo Benenson

Today

- Overview of techniques that make detection fast
- Exercise will be about optimizing already working pedestrian detector (1Hz -> 50Hz)

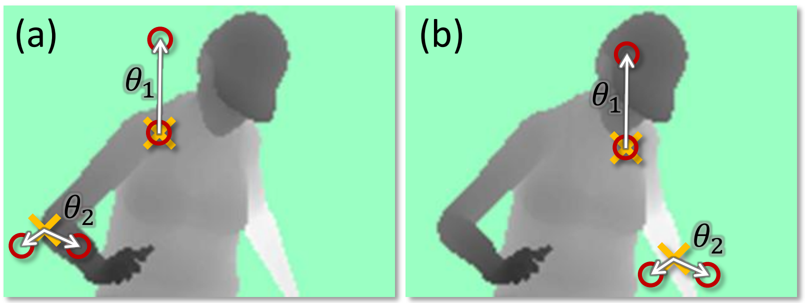
- First overview of recent success stories

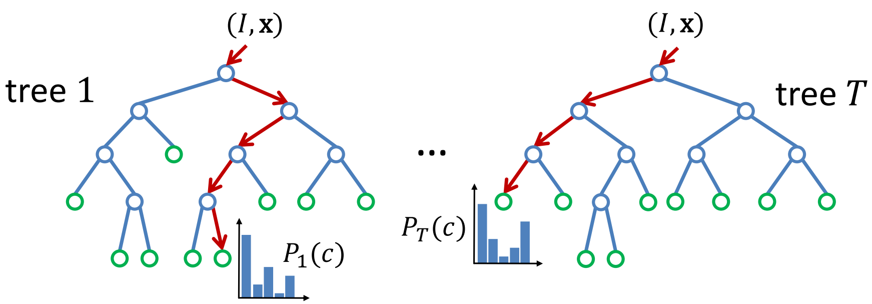
Main tricks you should know of

- integral images -> fast feature computation
- sliding window detection -> parallelism
- boosted classifier -> cascade
- sparselets -> sharing of computation for multi class
- efficient search over scale -> Rodrigo

Recent Success Stories

Real-Time Human Pose Recognition in Parts from Single Depth Images (XBox Kinect Post Estimation) [Shotton] (best paper)





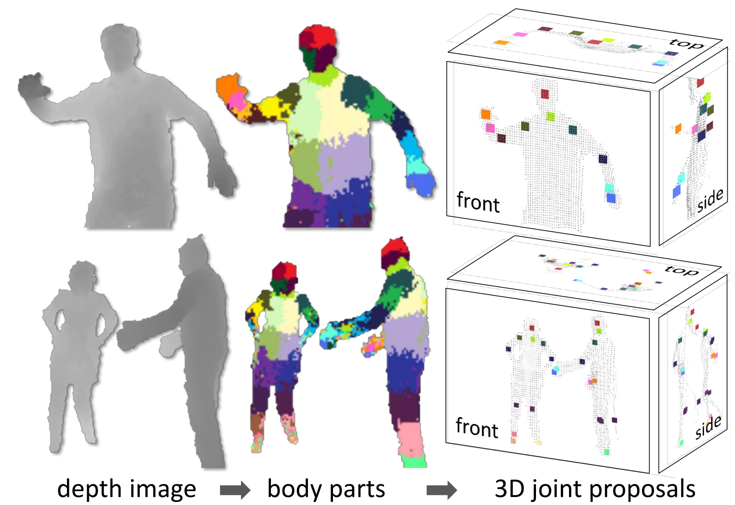


Figure 1. **Overview.** From an single input depth image, a per-pixel body part distribution is inferred. (Colors indicate the most likely part labels at each pixel, and correspond in the joint proposals). Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users.

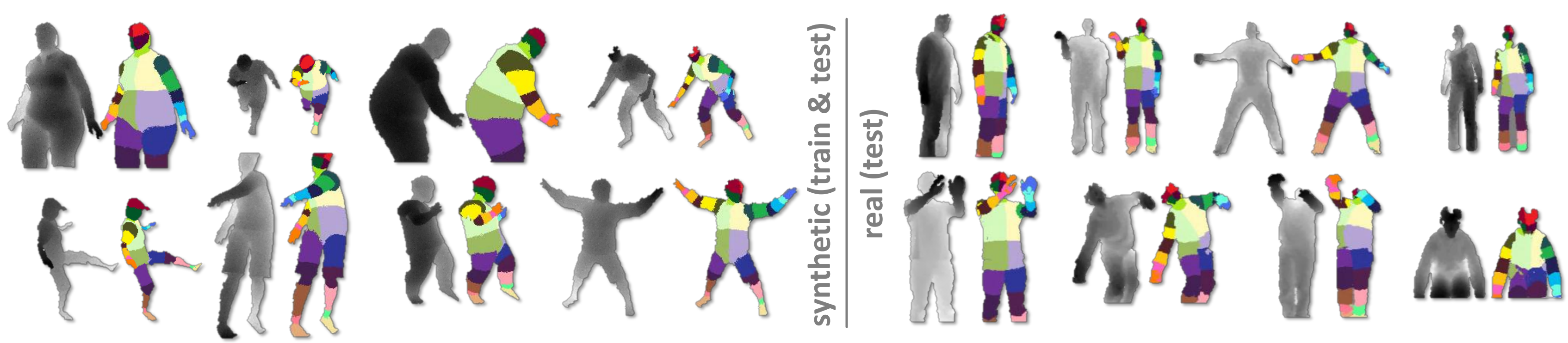


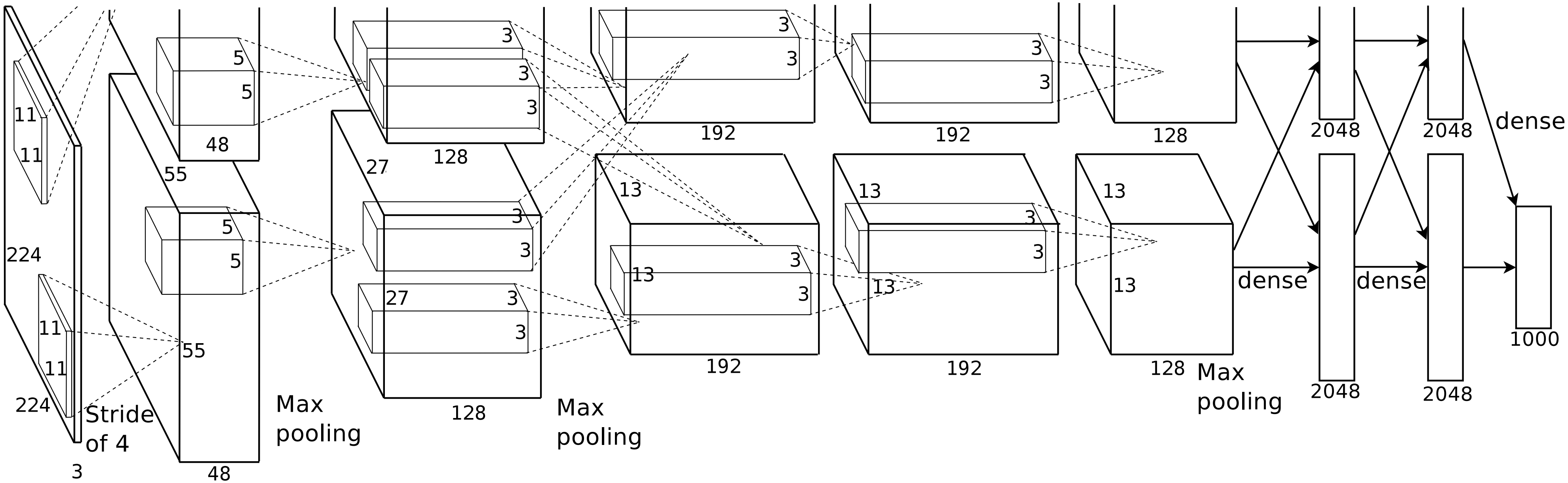
Figure 2. **Synthetic and real data**. Pairs of depth image and ground truth body parts. Note wide variety in pose, shape, clothing, and crop.

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
Ilya Sutskever
Geoffrey Hinton

University of Toronto
Canada

Paper with same name to appear in NIPS 2012



- Convolutional Network with 60 Million parameters
- Trained on RGB raw pixels (-mean)
- 15 % error on ImageNet challenge ... 10% lower than anybody else

Pedestrian detection at 100 frames per second (in the exercise)

Rodrigo Benenson, Markus Mathias, Radu Timofte and Luc Van Gool

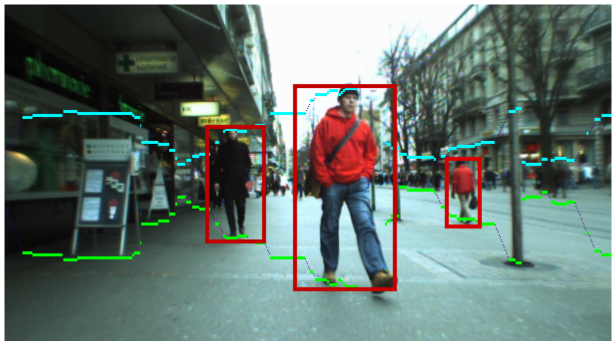
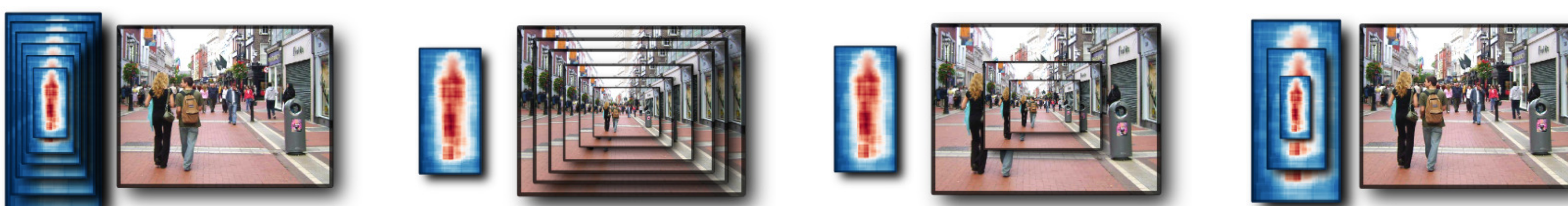


Figure 1: Example result on the Bahnhof sequence. Green line indicates the stixels bottom, blue line the stixels top and the red boxes are the obtained detections.

Detector aspect	Relative speed	Absolute speed
Baseline detector (§2)	1×	1.38 Hz
+Single scale detector (§3)	2×	2.68 Hz
+Soft-cascade (§4)	20×	50 Hz
+Estimated ground plane (§5)	2×	100 Hz
+Estimated stixels (§5)	1.35×	135 Hz
Our monocular detector	-	50 Hz
Our stereo (stixels) detector	-	135 Hz



N models, 1 image scale
(a) Naive approach

1 model, N image scales
(b) Traditional approach

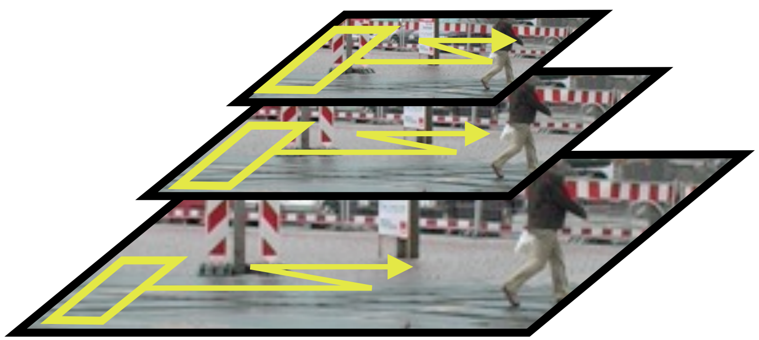
1 model, N/K image scales
(c) FPDW approach

N/K models, 1 image scale
(d) Our approach

Sliding Window Methods - Overview

- Sliding Window Based People Detection:

Scan Image → Extract Feature Vector → Classify Feature Vector → Non-Maxima Suppression



‘slide’ detection window over all positions & scales

Two Important Questions:
1) which feature vector
2) which classifier

Today:
- Viola Jones Face Detector
  - Haar Features
  - Boosting (cascade)
- HOG Pedestrian Detector
  - HOG descriptor
  - linear SVM

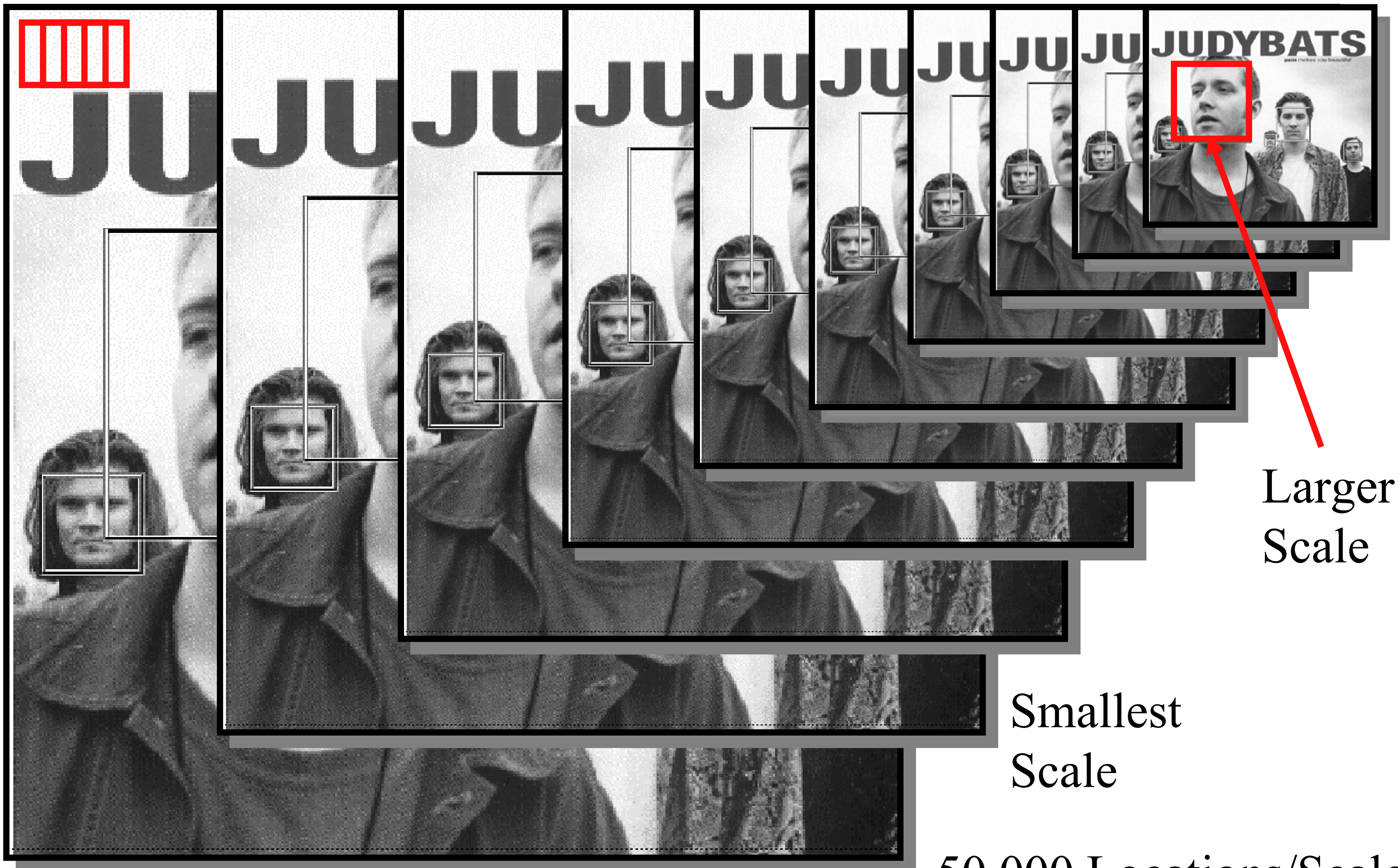
max planck institut informatik

Rapid Object Detection Using a Boosted Cascade of Simple Features

Paul Viola Michael J. Jones
Mitsubishi Electric Research Laboratories (MERL)
Cambridge, MA

Most of this work was done at Compaq CRL before the authors moved to MERL

The Classical Face Detection Process



50,000 Locations/Scales

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Classifier is Learned from Labeled Data

- Training Data
  - 5000 faces
    - All frontal
  - 10^8 non faces
  - Faces are normalized
    - Scale, translation
- Many variations
  - Across individuals
  - Illumination
  - Pose (rotation both in plane and out)

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

What is novel about this approach?

- Feature set (… is huge about 16,000,000 features)
- Efficient feature selection using AdaBoost
- New image representation: Integral Image
- Cascaded Classifier for rapid detection
  - Hierarchy of Attentional Filters

The combination of these ideas yields the fastest known face detector for gray scale images.

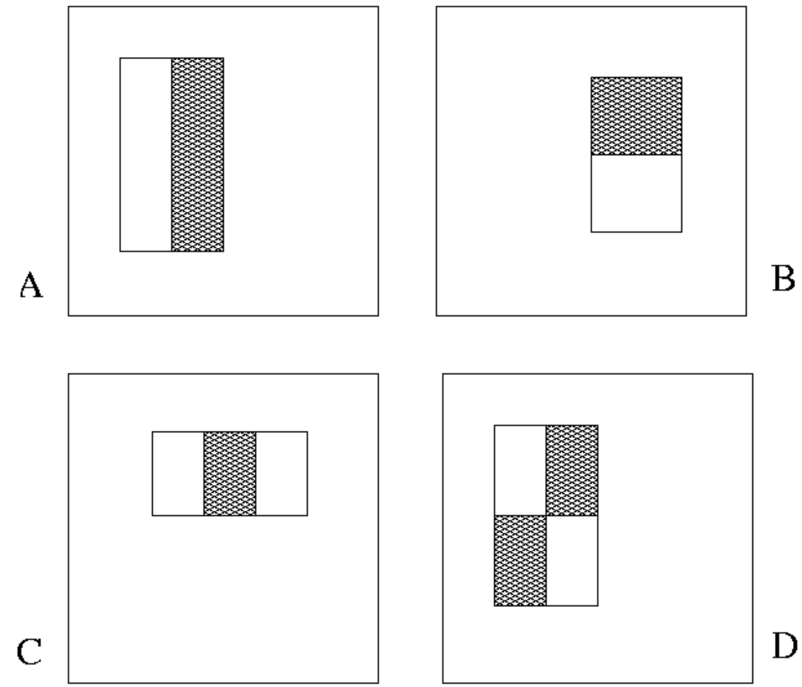
Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Image Features

"Rectangle filters"

Similar to Haar wavelets

Differences between sums of pixels in adjacent rectangles



$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ -1 & \text{otherwise} \end{cases}$$

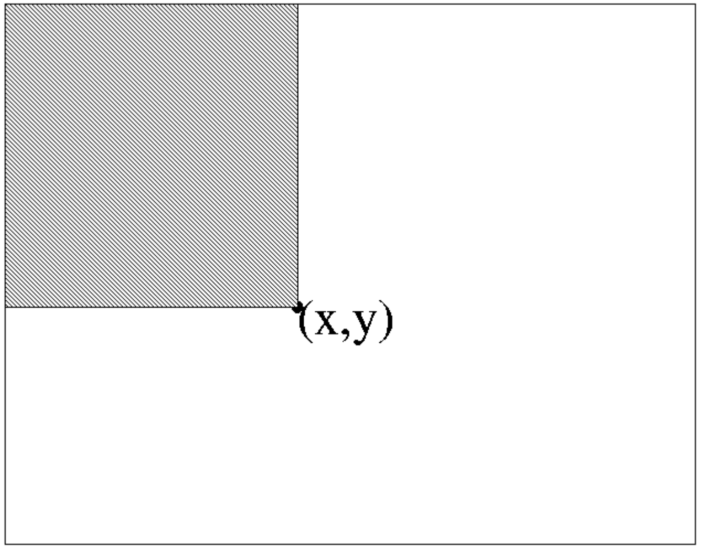
$$\boxed{160{,}000 \times 100 = 16{,}000{,}000}$$
Unique Features

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Integral Image

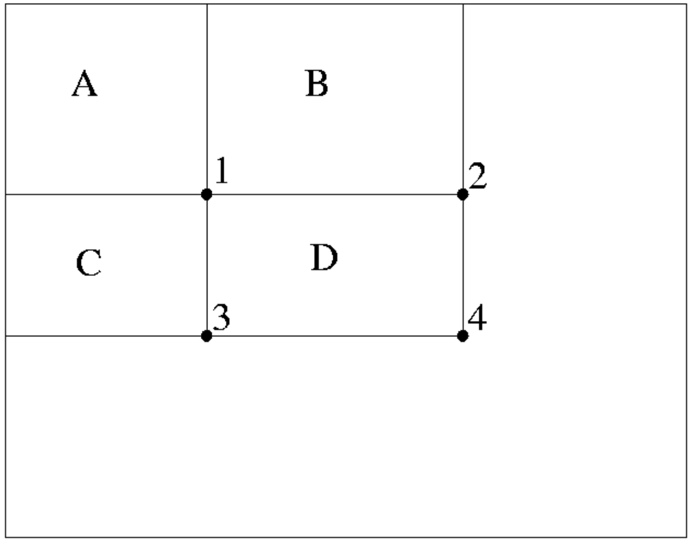
- Define the Integral Image

$$I'(x,y) = \sum_{\substack{x' \le x \\ y' \le y}} I(x', y')$$



- Any rectangular sum can be computed in constant time:

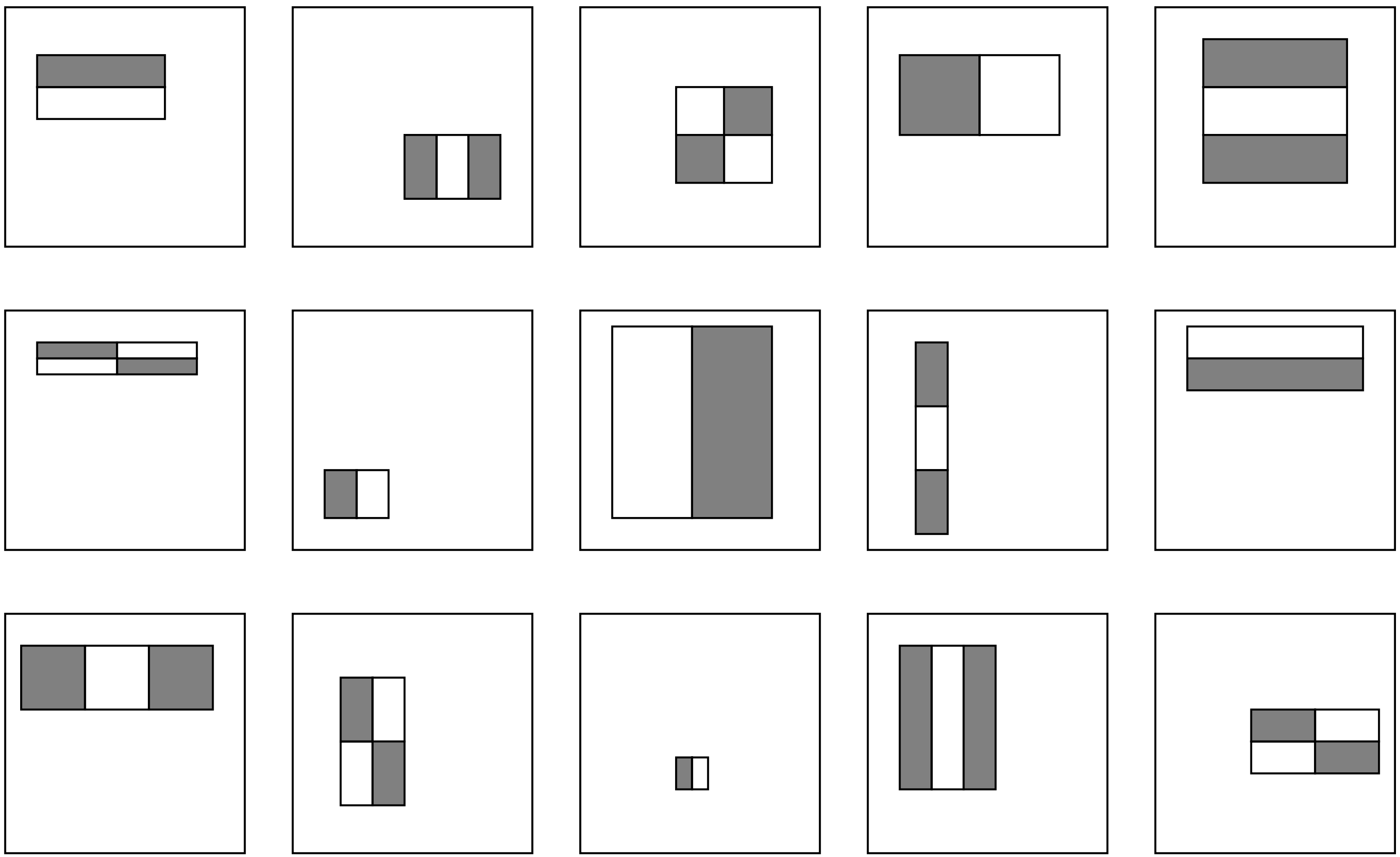
$$\begin{aligned} D &= 1 + 4 - (2+3) \\ &= A + (A+B+C+D) - (A+C+A+B) \\ &= D \end{aligned}$$



- Rectangle features can be computed as differences between rectangles

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Huge "Library" of Filters



Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Constructing Classifiers

- Perceptron yields a sufficiently powerful classifier

$$C(x) = \theta\left(\sum_i \alpha_i h_i(x) + b\right)$$

- Use AdaBoost to efficiently choose best features

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

AdaBoost

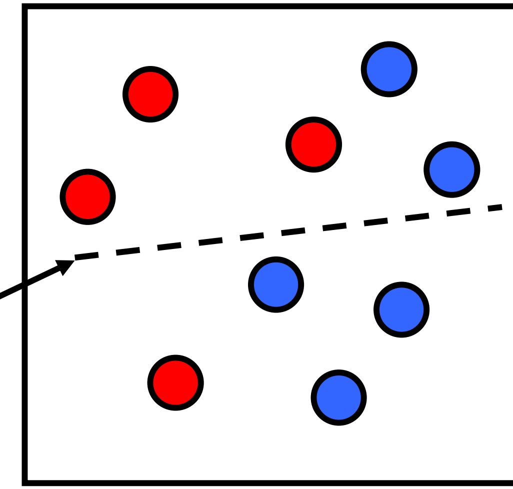
(Freund & Shapire '95)

$$f(x) = \theta\left(\sum_t \alpha_t h_t(x)\right)$$

$$\alpha_t = 0.5 \log\left(\frac{error_t}{1 - error_t}\right)$$

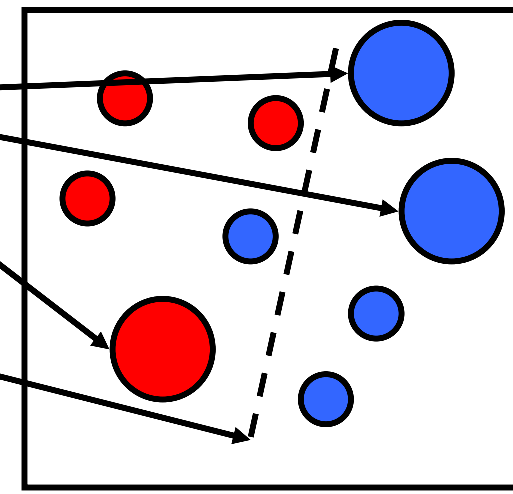
Initial uniform weight on training examples

weak classifier 1



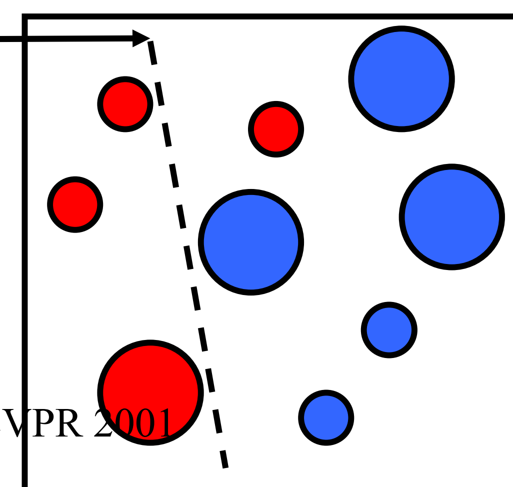
Incorrect classifications re-weighted more heavily

weak classifier 2



weak classifier 3

Final classifier is weighted combination of weak classifiers



Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

AdaBoost (Freund & Shapire 95)

- Given examples $(x_1, y_1), \ldots, (x_N, y_N)$ where $y_i = 0,1$ for negative and positive examples respectively.
- Initialize weights $w_{t=1,i} = 1/N$

- For $t=1, \ldots, T$
  - Normalize the weights, $w_{t,i} = w_{t,i} / \sum_{j=1}^{N} w_{t,j}$
  - Find a weak learner, i.e. a hypothesis, $h_t(x)$ with weighted error less than .5
  - Calculate the error of h_t : $e_t = \sum w_{t,i} \mid h_t(x_i) - y_i \mid$
  - Update the weights: $w_{t,i} = w_{t,i} B_t^{(1-d_i)}$ where $B_t = e_t / (1- e_t)$ and $d_i = 0$ if example x_i is classified correctly, $d_i = 1$ otherwise.

- The final strong classifier is

$$h(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) > 0.5 \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log(1/ B_t)$

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

AdaBoost for Efficient Feature Selection

- Our Features = Weak Classifiers
- For each round of boosting:
  - Evaluate each rectangle filter on each example
  - Sort examples by filter values
  - Select best threshold for each filter (min error)
    - Sorted list can be quickly scanned for the optimal threshold
  - Select best filter/threshold combination
  - Weight on this feature is a simple function of error rate
  - Reweight examples
  - (There are many tricks to make this more efficient.)

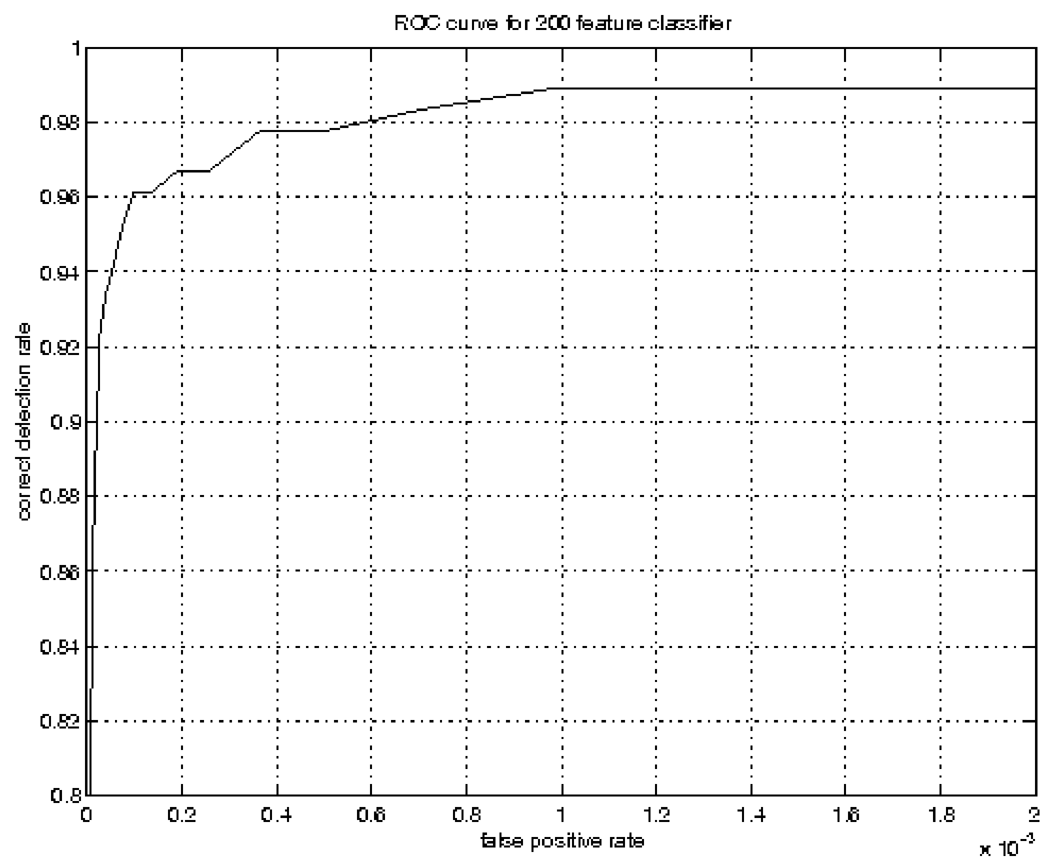
Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Example Classifier for Face Detection

A classifier with 200 rectangle features was learned using AdaBoost

95% correct detection on test set with 1 in 14084
false positives.

Not quite competitive...

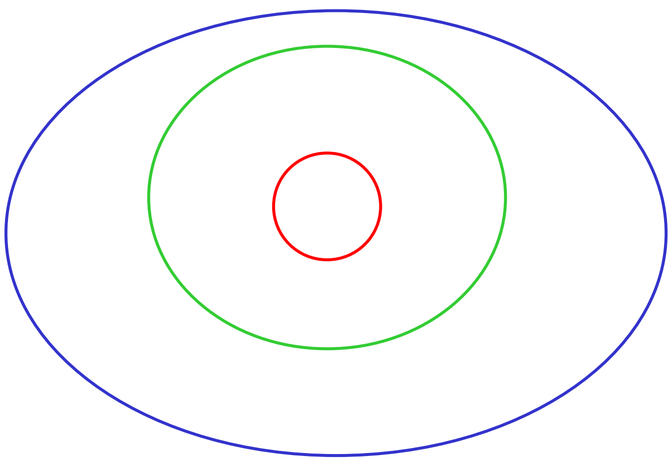


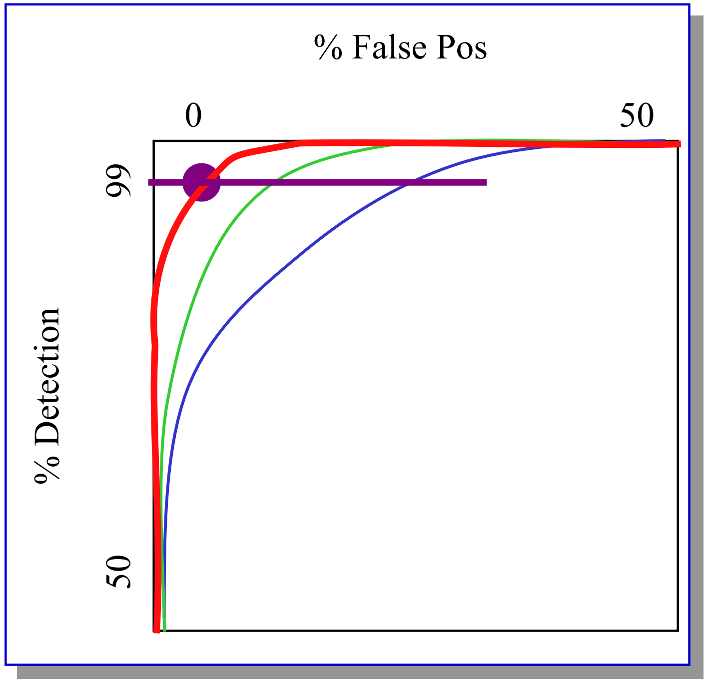
ROC curve for 200 feature classifier

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

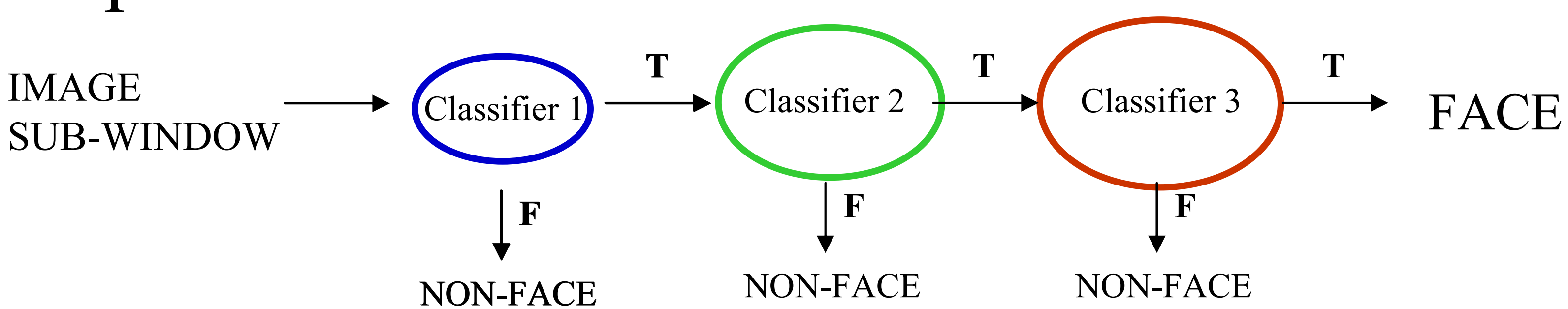
Trading Speed for Accuracy

- Given a nested set of classifier hypothesis classes



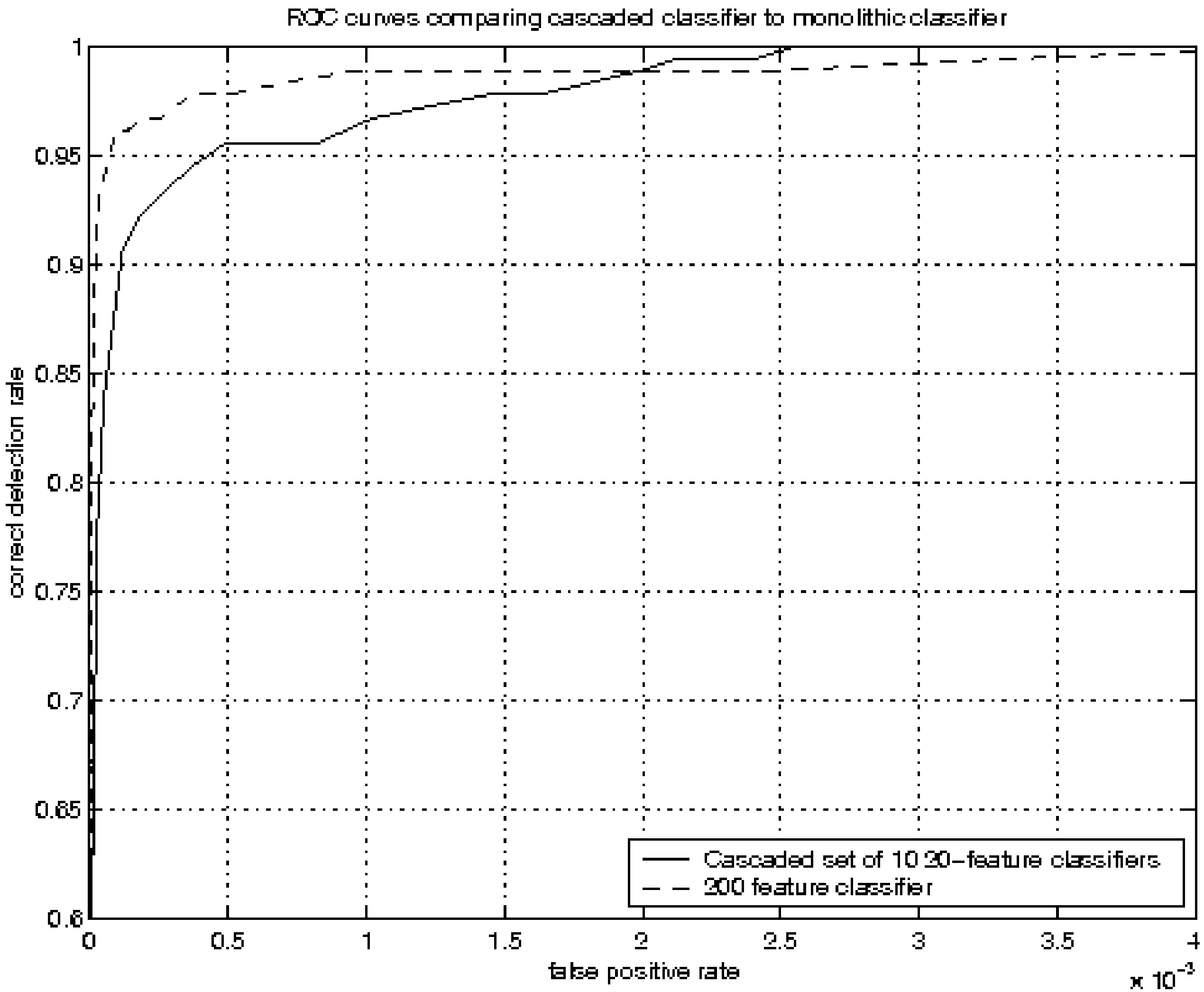


- Computational Risk Minimization



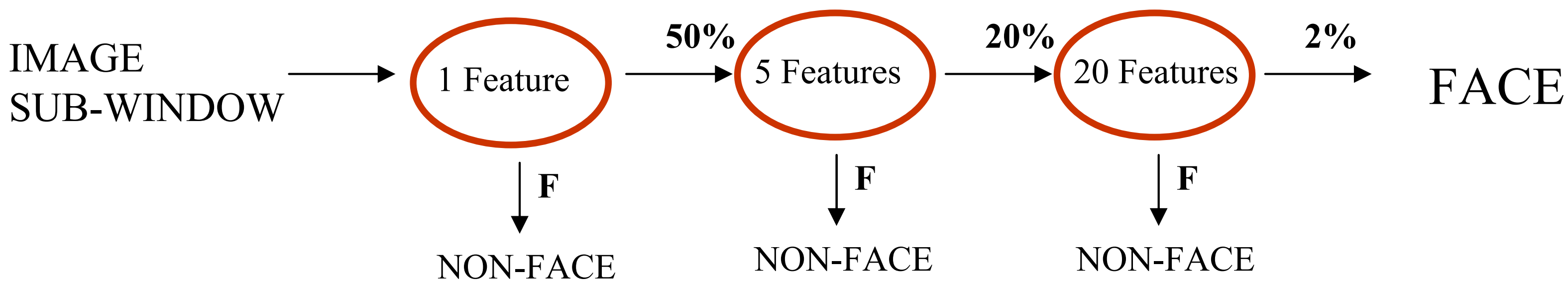
Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Experiment: Simple Cascaded Classifier



Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Cascaded Classifier



- A 1 feature classifier achieves 100% detection rate and about 50% false positive rate.
- A 5 feature classifier achieves 100% detection rate and 40% false positive rate (20% cumulative)
  - using data from previous stage.
- A 20 feature classifier achieve 100% detection rate with 10% false positive rate (2% cumulative)

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

A Real-time Face Detection System

Training faces: 4916 face images (24 x 24 pixels) plus vertical flips for a total of 9832 faces

Training non-faces: 350 million sub-windows from 9500 non-face images

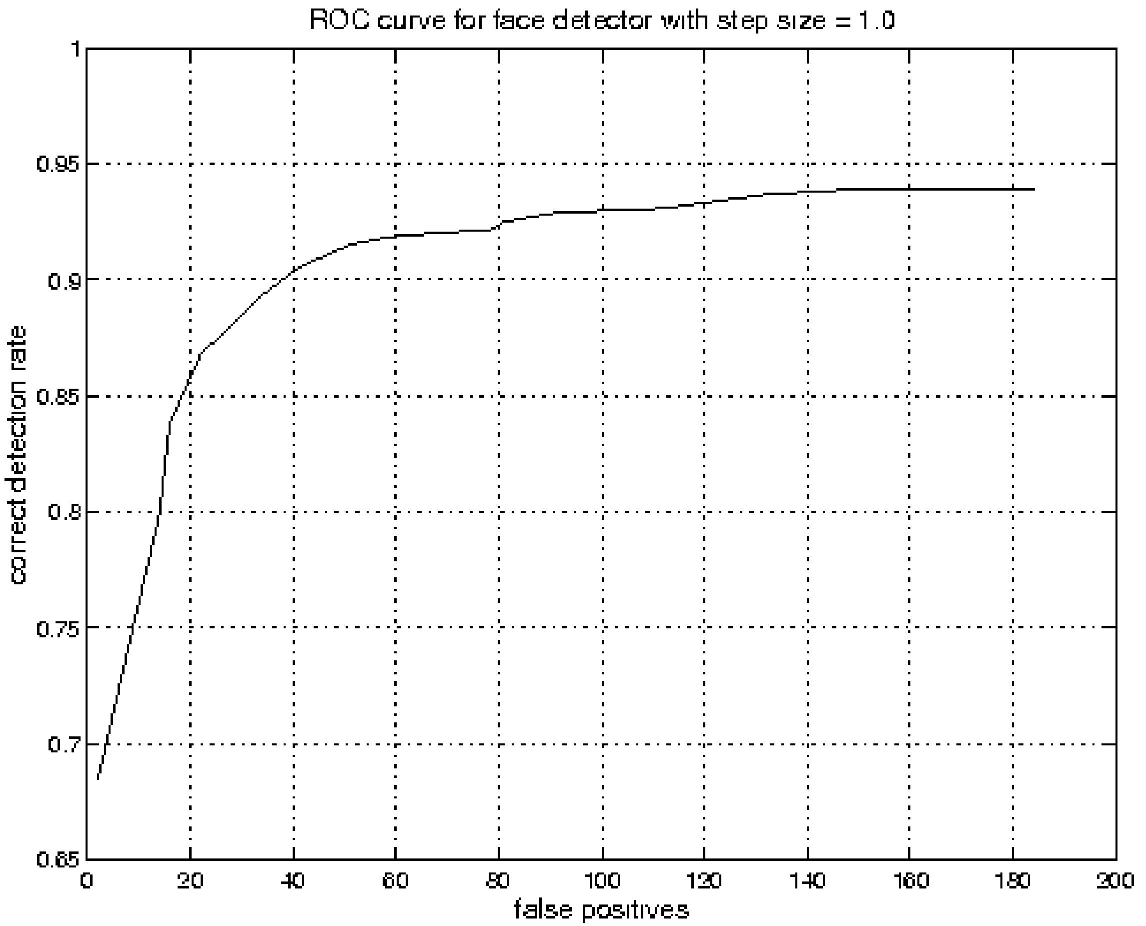
Final detector: 38 layer cascaded classifier
The number of features per layer was 1, 10, 25, 25, 50, 50, 50, 75, 100, …, 200, …

Final classifier contains 6061 features.

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Accuracy of Face Detector

Performance on MIT+CMU test set containing 130 images with 507 faces and about 75 million sub-windows.



Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Comparison to Other Systems

False Detections / Detector	10	31	50	65	78	95	110	167
Viola-Jones	76.1	88.4	91.4	92.0	92.1	92.9	93.1	93.9
Viola-Jones (voting)	81.1	89.7	92.1	93.1	93.1	93.2	93.7	93.7
Rowley-Baluja-Kanade	83.2	86.0				89.2		90.1
Schneiderman-Kanade				94.4				

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Speed of Face Detector

Speed is proportional to the average number of features computed per sub-window.

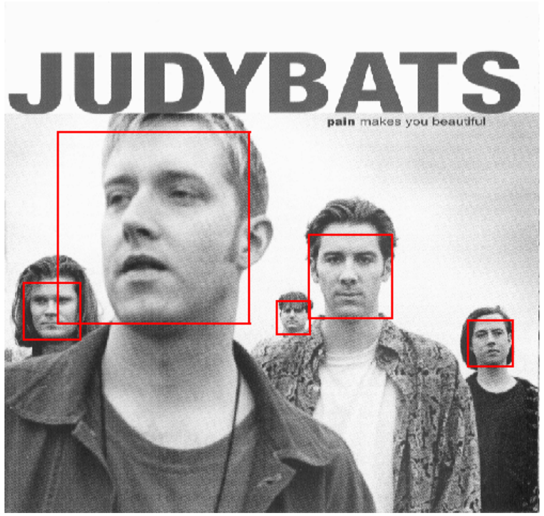
On the MIT+CMU test set, an average of 9 features out of a total of 6061 are computed per sub-window.

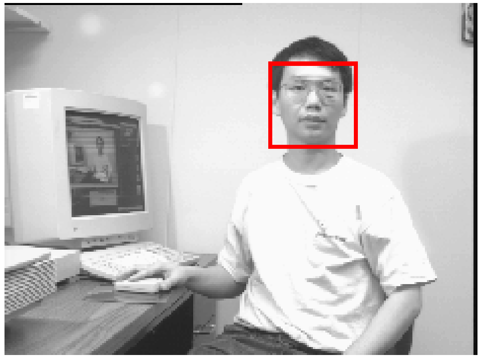
On a 700 Mhz Pentium III, a 384x288 pixel image takes about 0.067 seconds to process (15 fps).

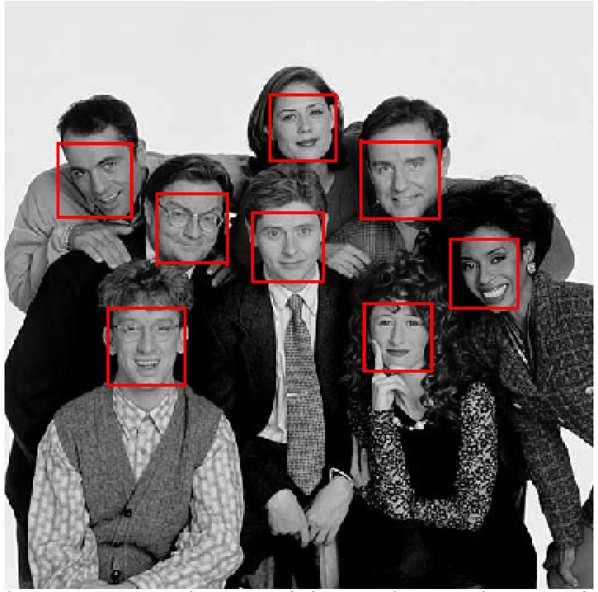
Roughly 15 times faster than Rowley-Baluja-Kanade and 600 times faster than Schneiderman-Kanade.

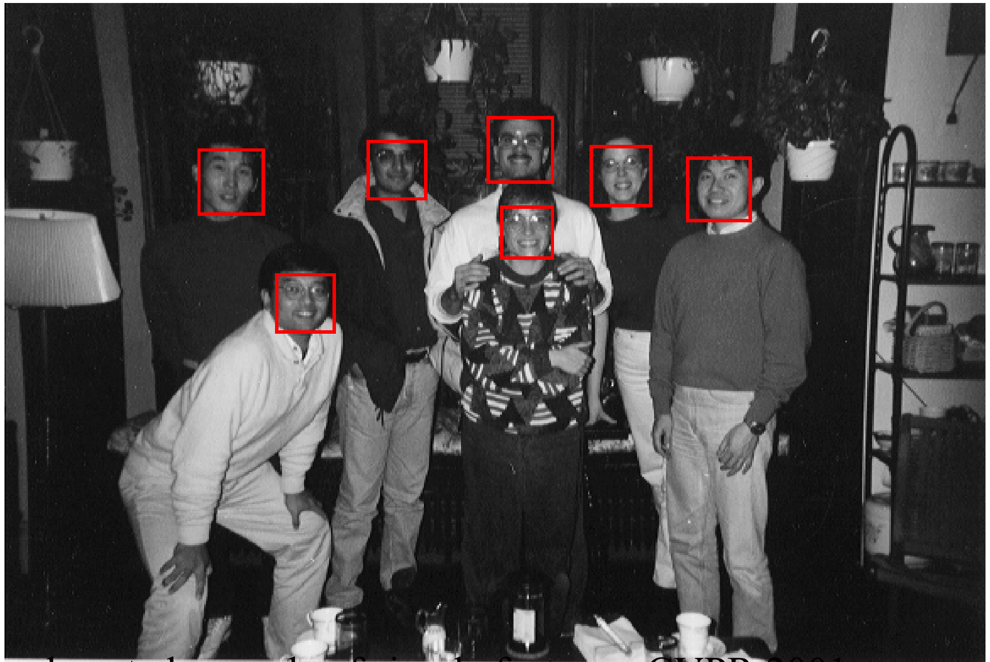
Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Output of Face Detector on Test Images



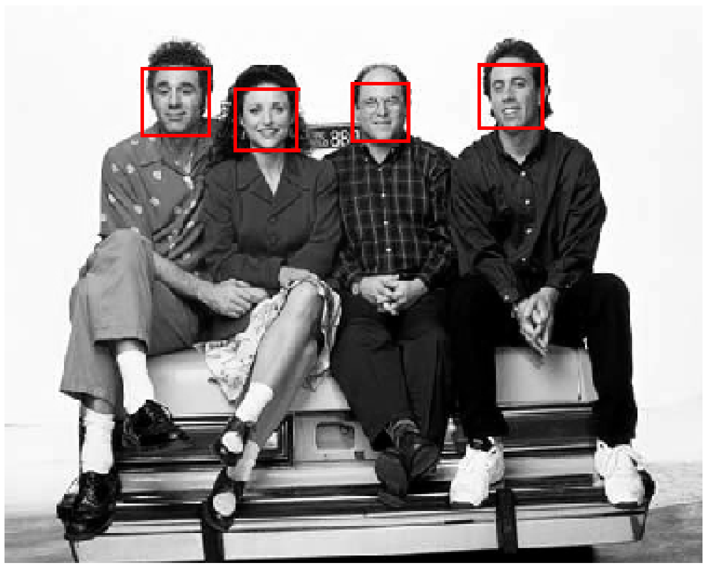


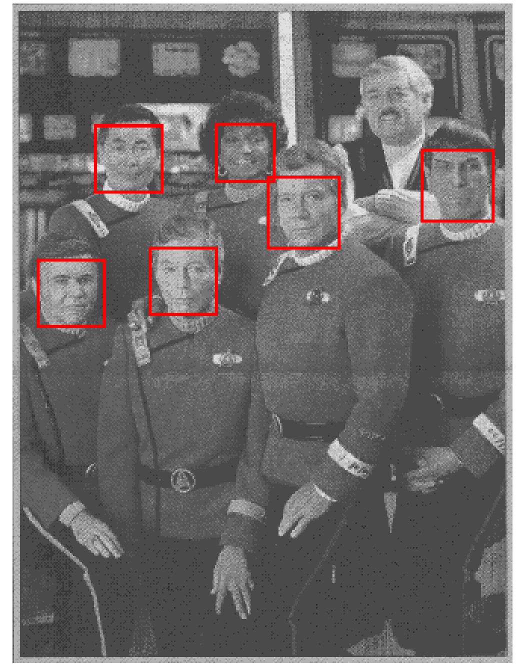


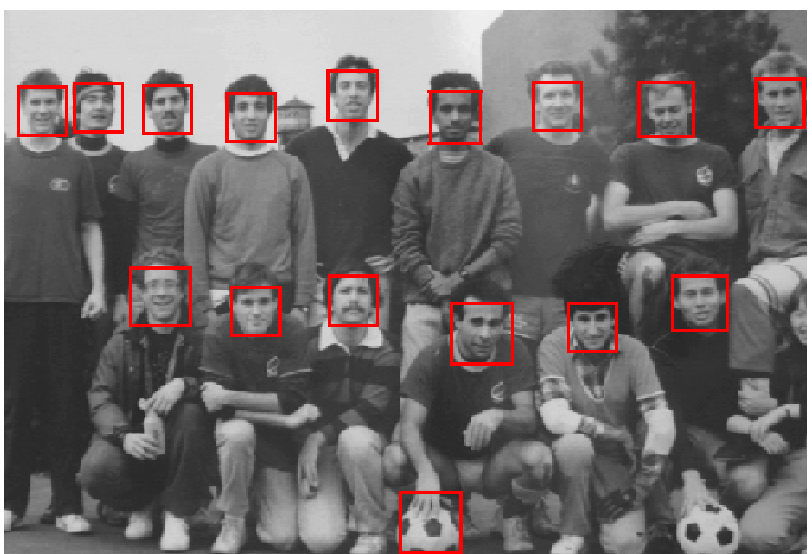


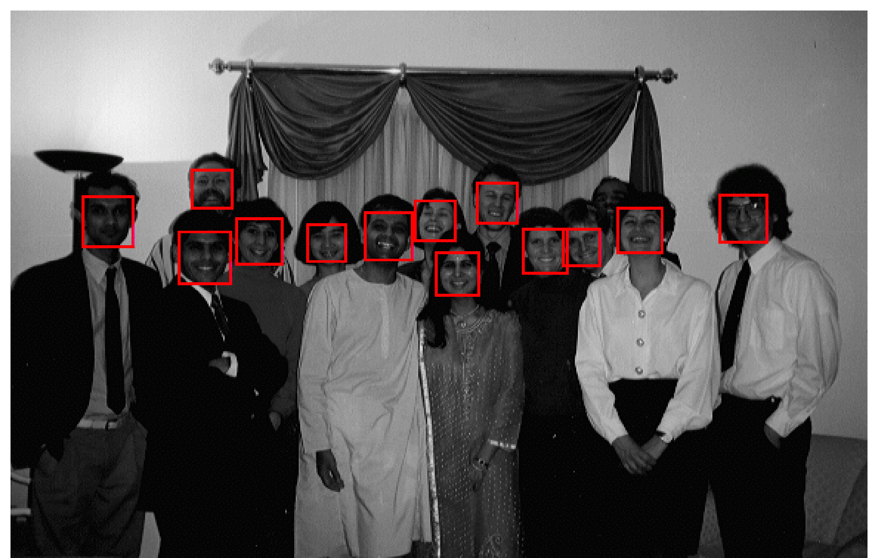
Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

More Examples









Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Conclusions

- We [they] have developed the fastest known face detector for gray scale images
- Three contributions with broad applicability
  - Cascaded classifier yields rapid classification
  - AdaBoost as an extremely efficient feature selector
  - Rectangle Features + Integral Image can be used for rapid image analysis

Viola and Jones, Robust object detection using a boosted cascade of simple features, CVPR 2001

Viola Jones Detector

- try it out
  - implementations available, e.g. opencv
  - works in real-time on reasonable image sizes

HOG Pedestrian Detector

Goals & Applications of HOG

- Original Goal:
  - Detect and Localize people in Images and Videos
- Applications:
  - Images, films & multi-media analysis
  - Pedestrian detection for autonomous cars
  - Visual surveillance, behavior analysis

Difficulties of People / Object Detection

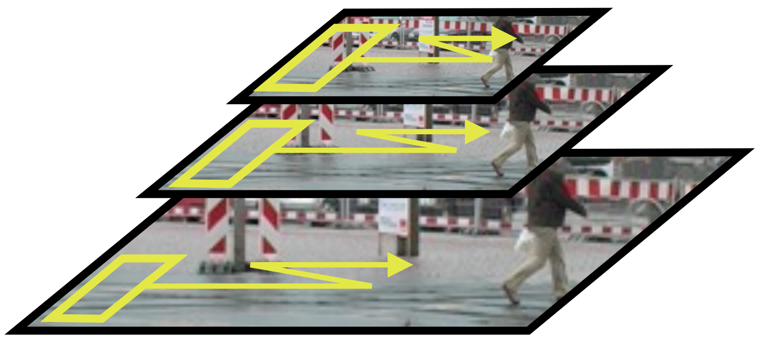
- Some of the Difficulties
  - Wide variety of articulated poses
  - Variable appearance and clothing
  - Complex backgrounds
  - Unconstrained illumination
  - Occlusions, different scales
  - Videos sequences involves motion of the subject, the camera and the objects in the background

- Main assumption for HOG: upright fully visible people

Sliding Window Methods - Overview

- Sliding Window Based People Detection:

Scan Image → Extract Feature Vector → Classify Feature Vector → Non-Maxima Suppression



‘slide’ detection window over all positions & scales

Two Important Questions:
1) which feature vector
2) which classifier

Today:
- Viola Jones Face Detector
  - Haar Features
  - Boosting (cascade)
- HOG Pedestrian Detector
  - HOG descriptor
  - linear SVM

HOG: Static Feature Extraction

Input Image

Detection Window

Normalise gamma

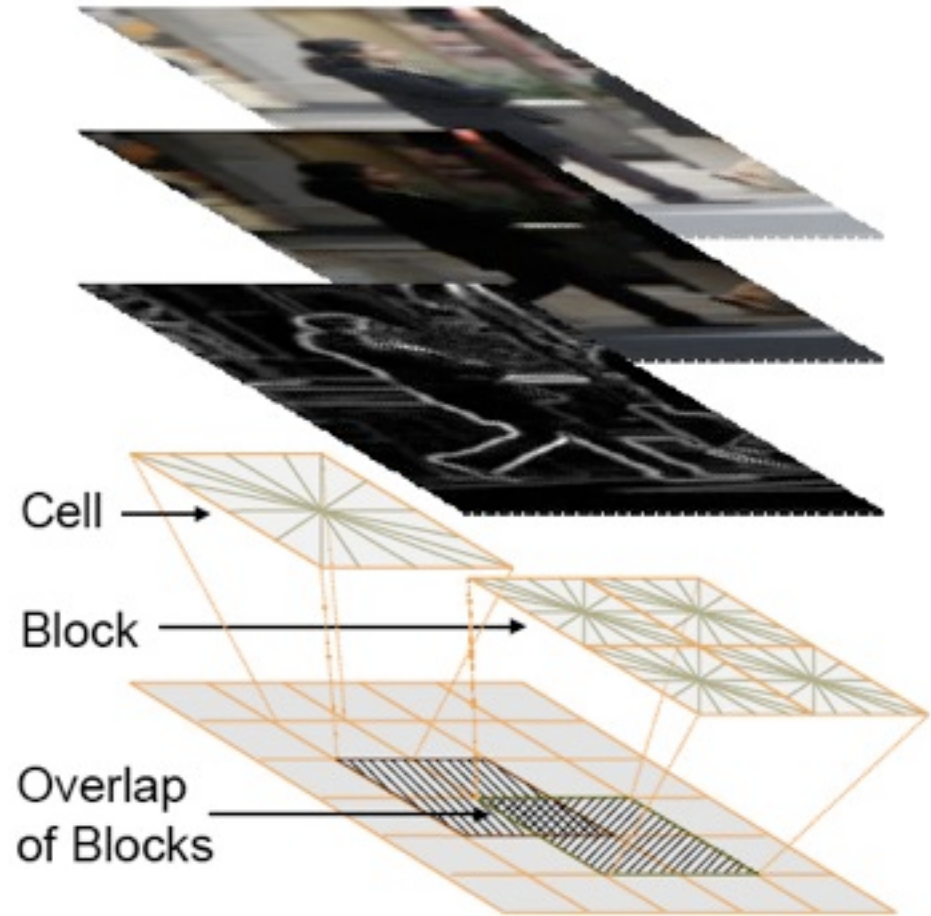
Compute gradients

Weighted vote in spatial & orientation cells

Contrast normalise over overlapping spatial cells

Collect HOGs over detection window

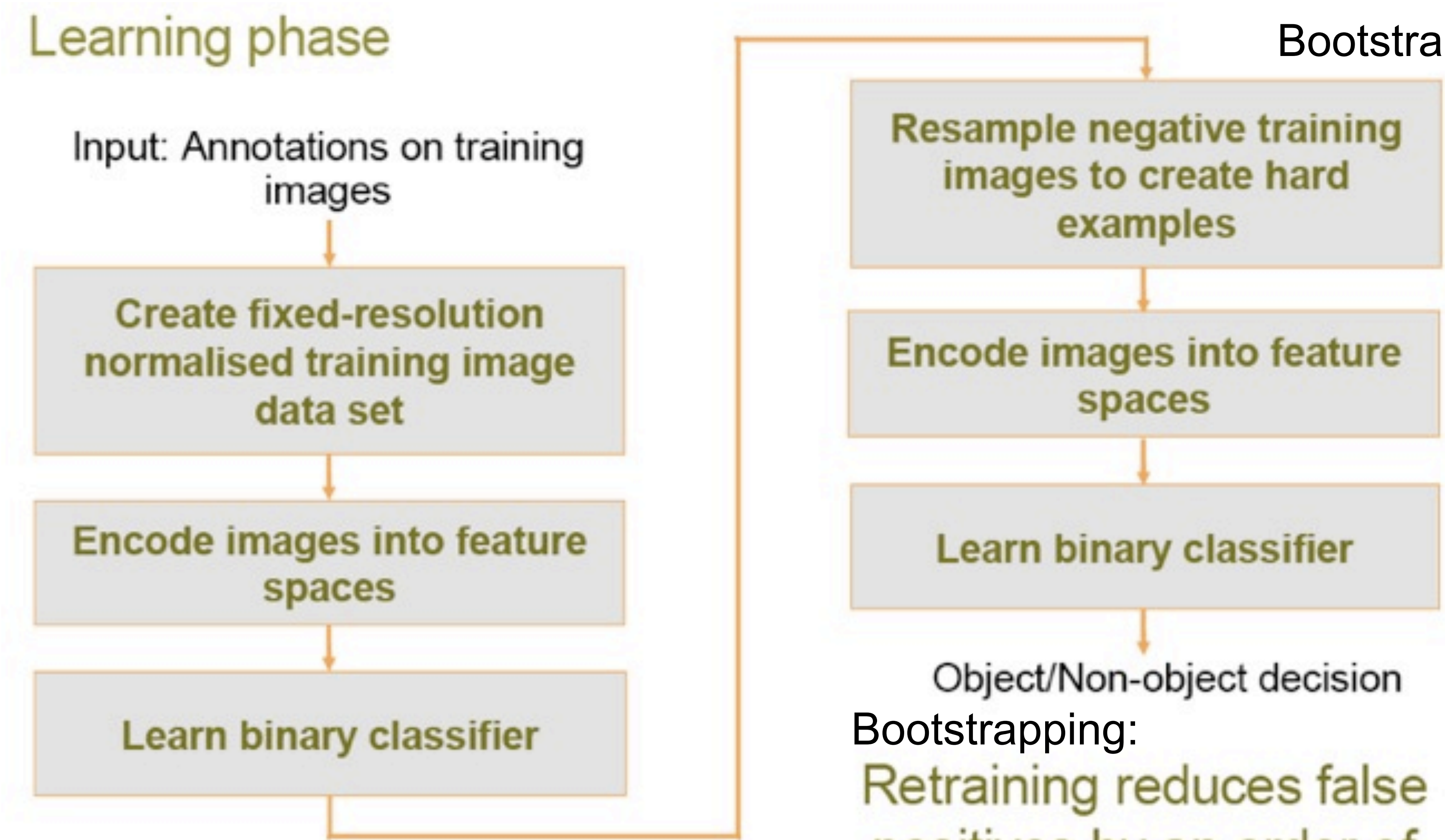
Linear SVM



Feature vector $f = [\ \ldots,\ \ldots,\ \ \ \ldots]$

Overview of Learning

Bootstrapping



Bootstrapping:

Retraining reduces false positives by an order of magnitude!

max planck institut
informatik

Sparselet Model for Efficient Multiclass Object Detection

Hyun-Oh Song[1], Stefan Zickler[2], Tim Althoff[1],
Ross Girshick[3], Christopher Geyer[2],
Mario Fritz[4], Pedro Felzenszwalb[5], Trevor Darrell[1]

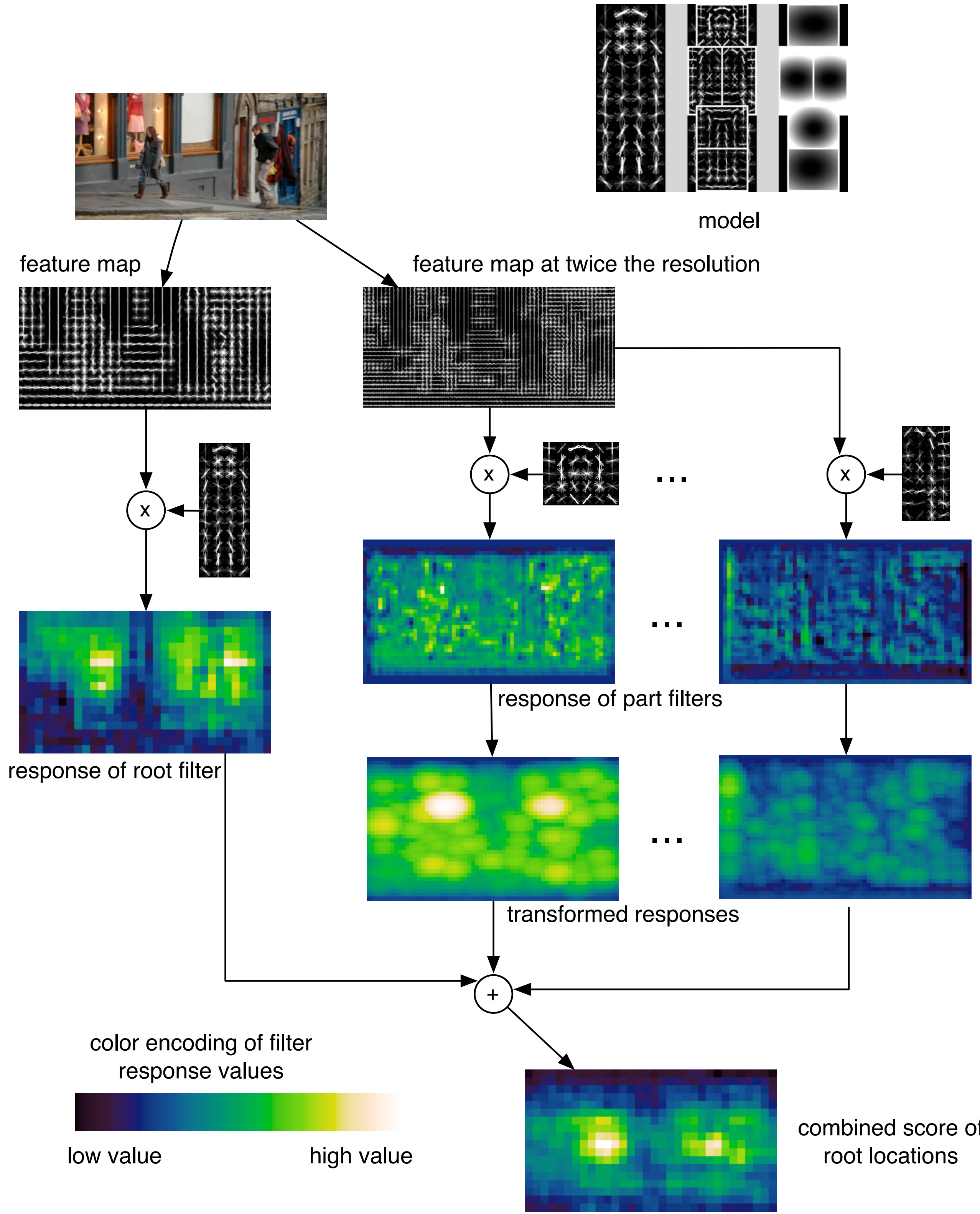
[1] UC Berkeley [2] iRobot [3] UChicago [4] MPI [5] Brown

Motivation

- Deformable Part Model [Felzenszwalb] and variants still one of the strongest detectors
- Not particularly fast
  - Lots of parts & convolutions
  - Detectors evaluated in isolation

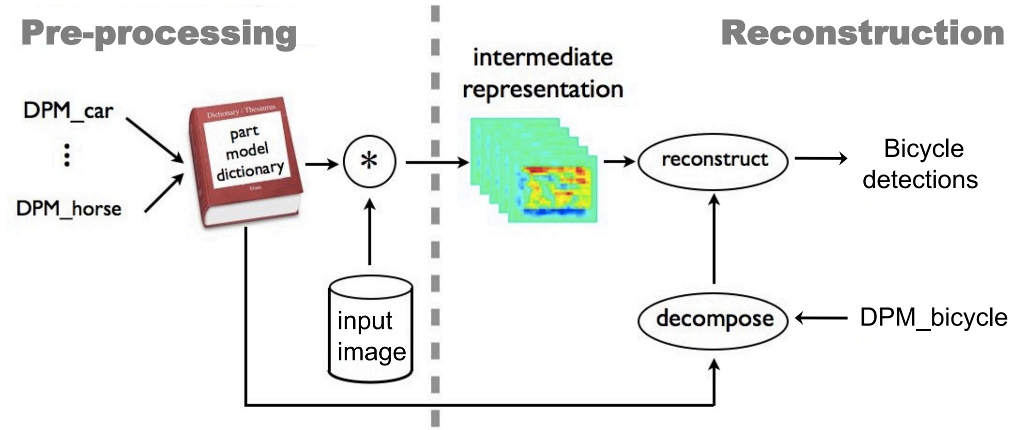
- Goal:
  - intermediate representation that shares computation across classes
  - leverage power of GPUs by parallel execution
  - realize different performance-speed-tradeoffs
  - Explore offline setting (post-hoc detection)

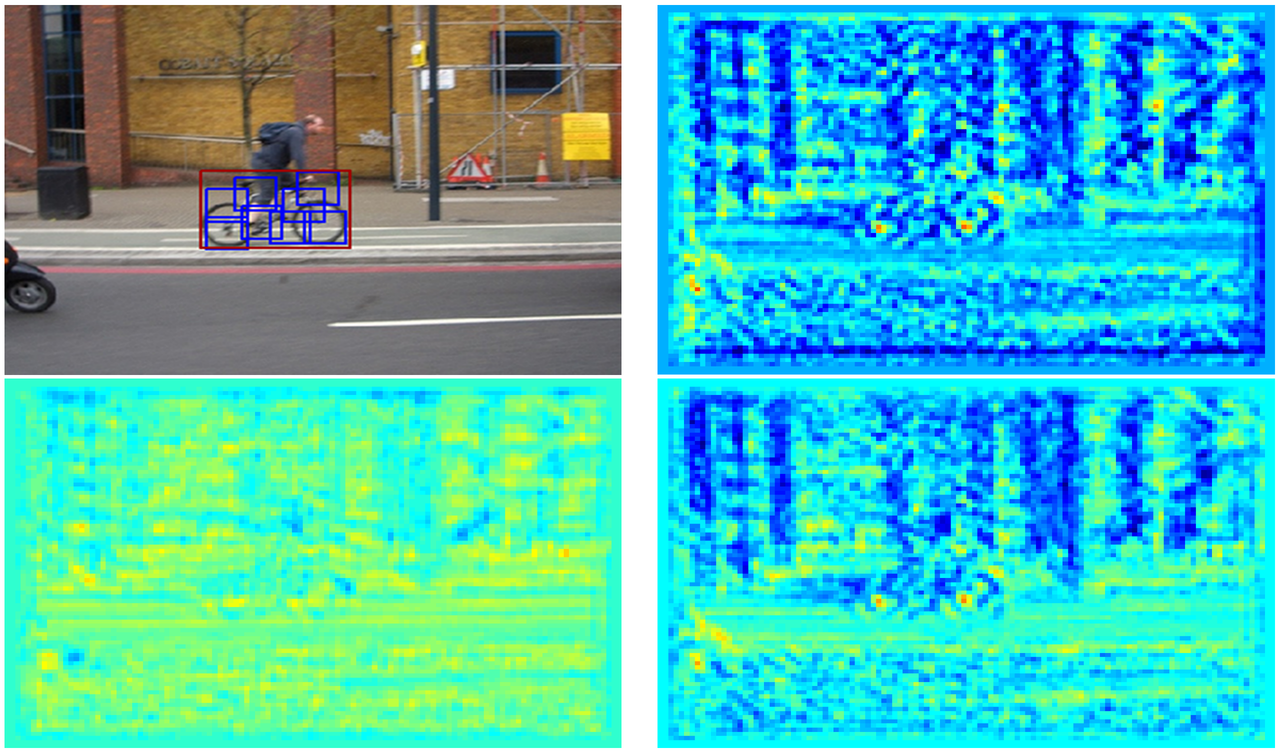
Deformable Part Model: The Big Picture



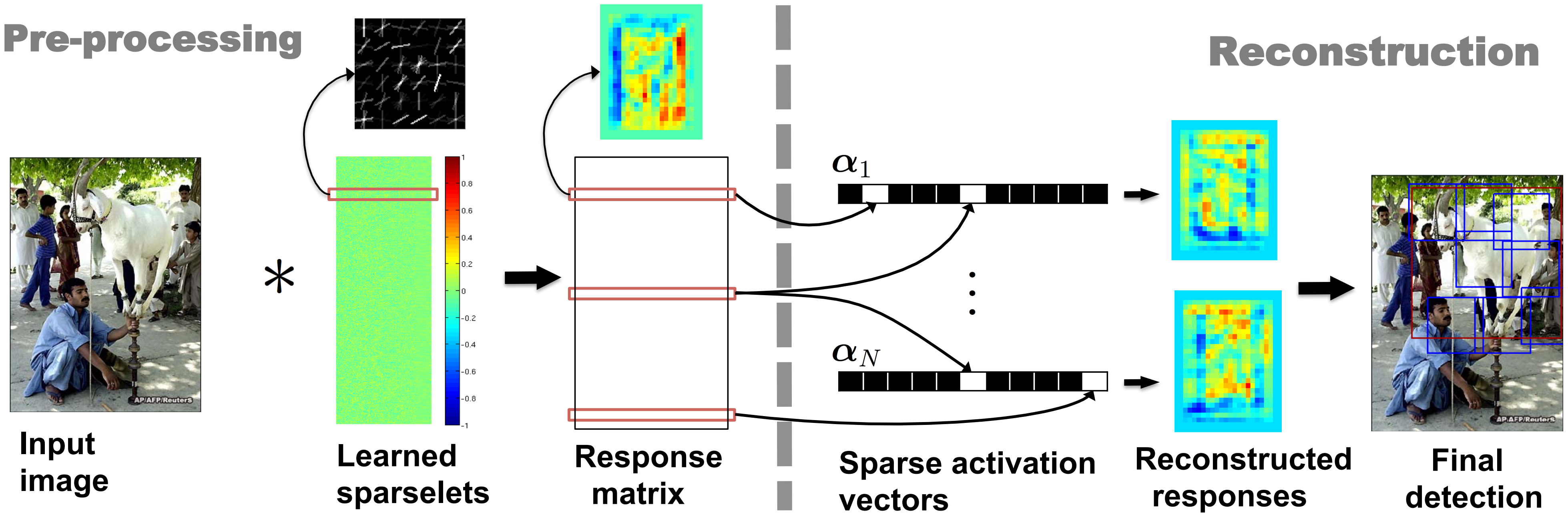
Main Idea

- Intermediate Representation that allows to share computation between parts
- True part responses can be reconstructed from a sparse subset of the basis
- General concept: exploit redundancy when doing lots of convolutions





Main Idea



- Reconstruction and convolutions are carried out in parallel on GPU

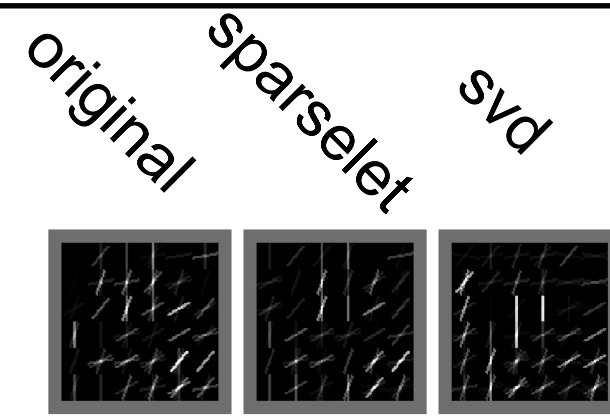
Sparselets: Sparse Representation for Part Filters

$$\min_{\alpha_{ij}, Z_j} \sum_{i=1}^{N} ||P_i - \sum_{j=1}^{J} \alpha_{ij} Z_j||_2^2$$

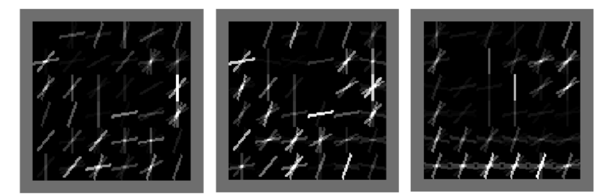
$$\text{subject to} \quad ||\boldsymbol{\alpha_i}||_0 \leq \epsilon \qquad \forall i = 1, ..., N$$

$$||Z_j||_2^2 \leq 1 \qquad \forall j = 1, ..., J$$

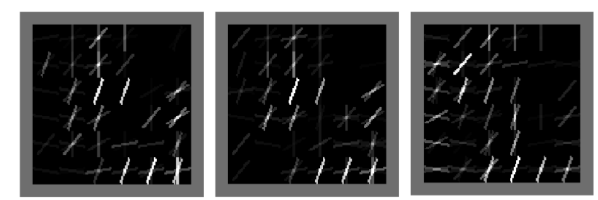
- part filters: $P_i \in \mathbb{R}^{p \times p \times h}$
- activations: $\boldsymbol{\alpha}_i \in \mathbb{R}^J$
- sparselet: $Z_j \in \mathbb{R}^{p \times p \times h}$
- maximum number of activations ϵ



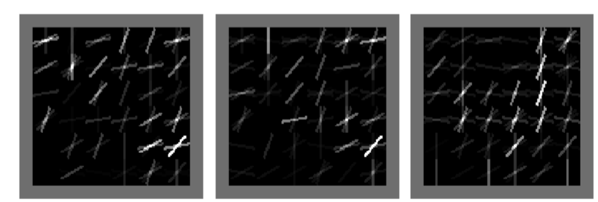
(a) Motorbike part 10



(c) Aeroplane part 27



(b) Sofa part 25



(d) Cat part 34

Sparselets: Sparse Representation for Part Filters

- Reconstruction of filter responses from sparselet filters

$$\Psi * P_i = \Psi * \left(\sum_j \alpha_{ij} D_j\right) = \sum_j \alpha_{ij} \left(\Psi * D_j\right)$$

$$\begin{bmatrix} -\Psi * P_1- \\ -\Psi * P_2- \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ -\Psi * P_n- \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \boldsymbol{\alpha}_n \end{bmatrix} \begin{bmatrix} -\Psi * D_1- \\ -\Psi * D_2- \\ \vdots \\ \vdots \\ -\Psi * D_J- \end{bmatrix} = A\ M$$

sparse reconstruction vector

sparselet responses

- Efficient sparse matrix multiplication

- Reconstructed DPM score

$$\text{score}_{\text{recon}}(\omega) = m_0(\omega) + \sum_{i=1}^{N} \max_{\delta} s_i(\omega + \delta) - d_i(\delta)$$

$$\text{where } s_i(\omega) = \sum_{\substack{j=1 \\ \forall \alpha_{ij} \neq 0}}^{|D|} \alpha_{ij} \left(\Psi\left(\omega\right) * D_j\right).$$

Speedup

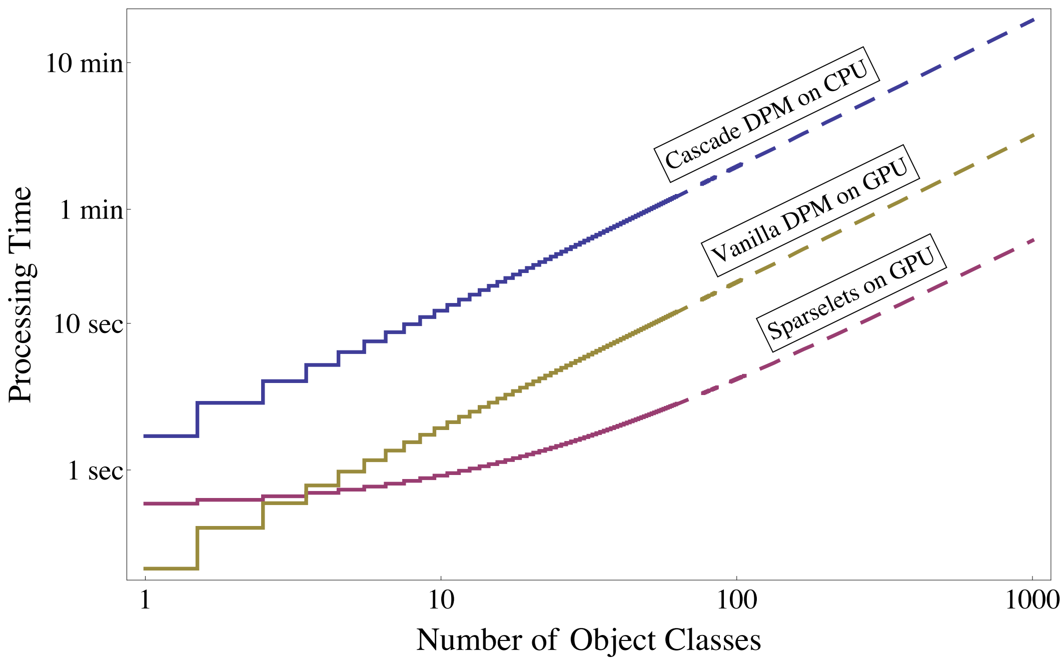
- online scenario #filters #features patch size
  - traditional: Nhp^2
  - sparselet: $|D|hp^2 + N\mathbb{E}[||\boldsymbol{\alpha_i}||_0]$

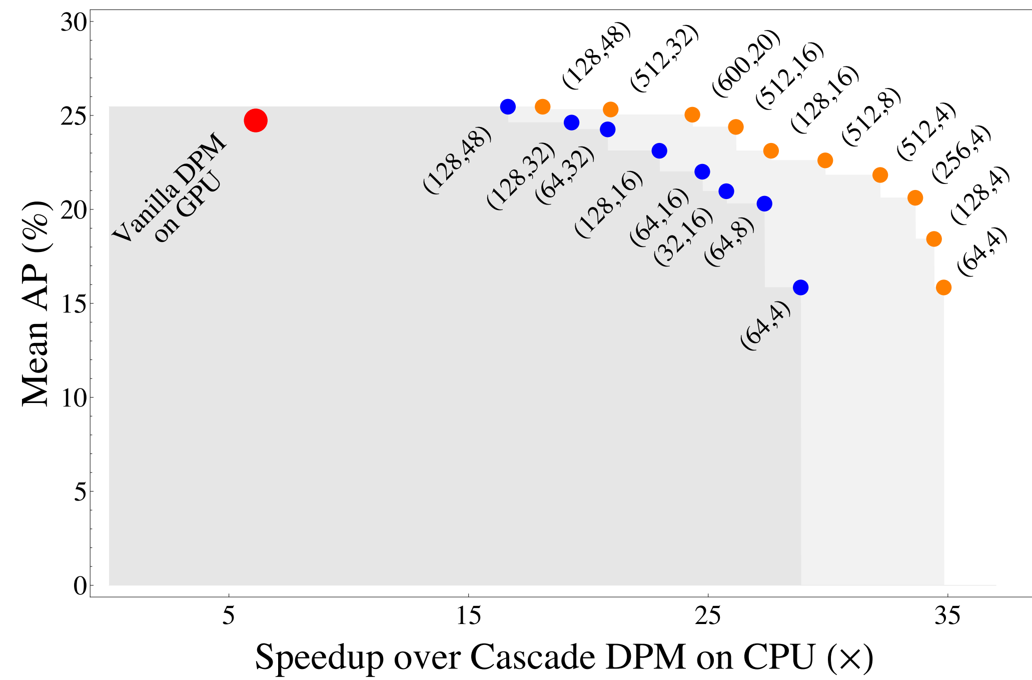
dictionary size

- offline/posthoc scenario: $\frac{hp^2}{\mathbb{E}[\ ||\boldsymbol{\alpha_i}||_0\ ]}$

- example:
  - 6x6 part, average activation of 20 at least 10x speedup

Speed Ups

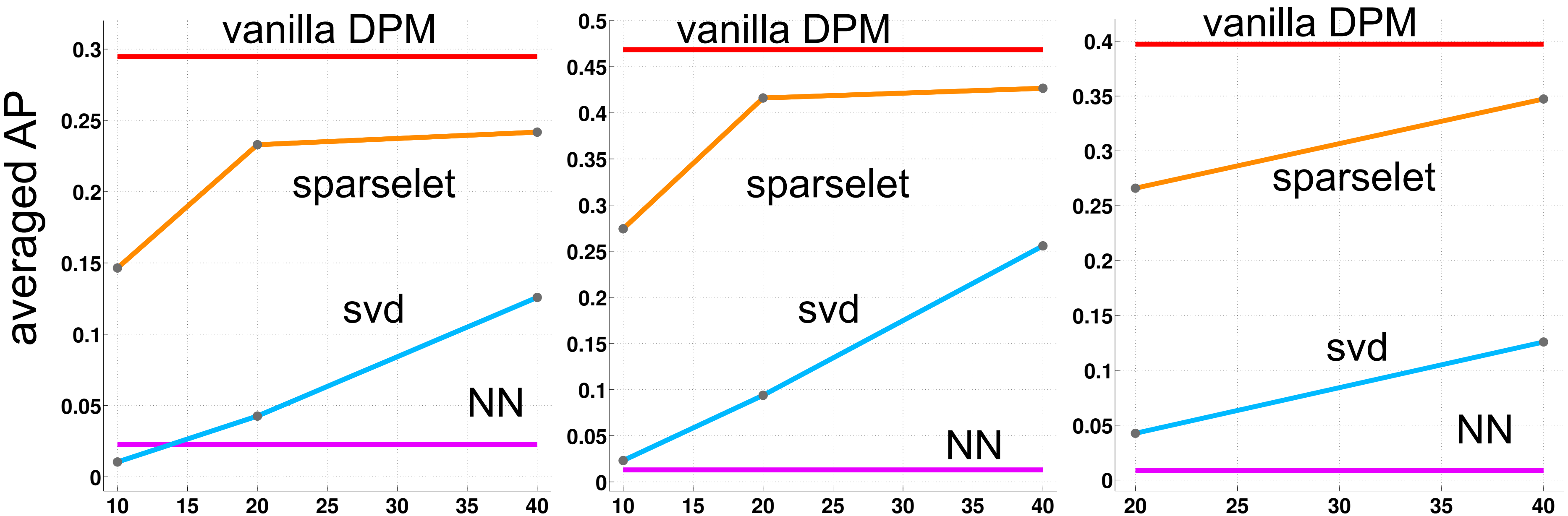




- 640x480 resolution
- 60 classes obtained by repetition of VOC
- dashed extrapolation

- method can realize different trade-offs by choosing number of basis and reconstruction quality
- orange: posthoc, retrieval setting
- 25x for online; 35x for offline

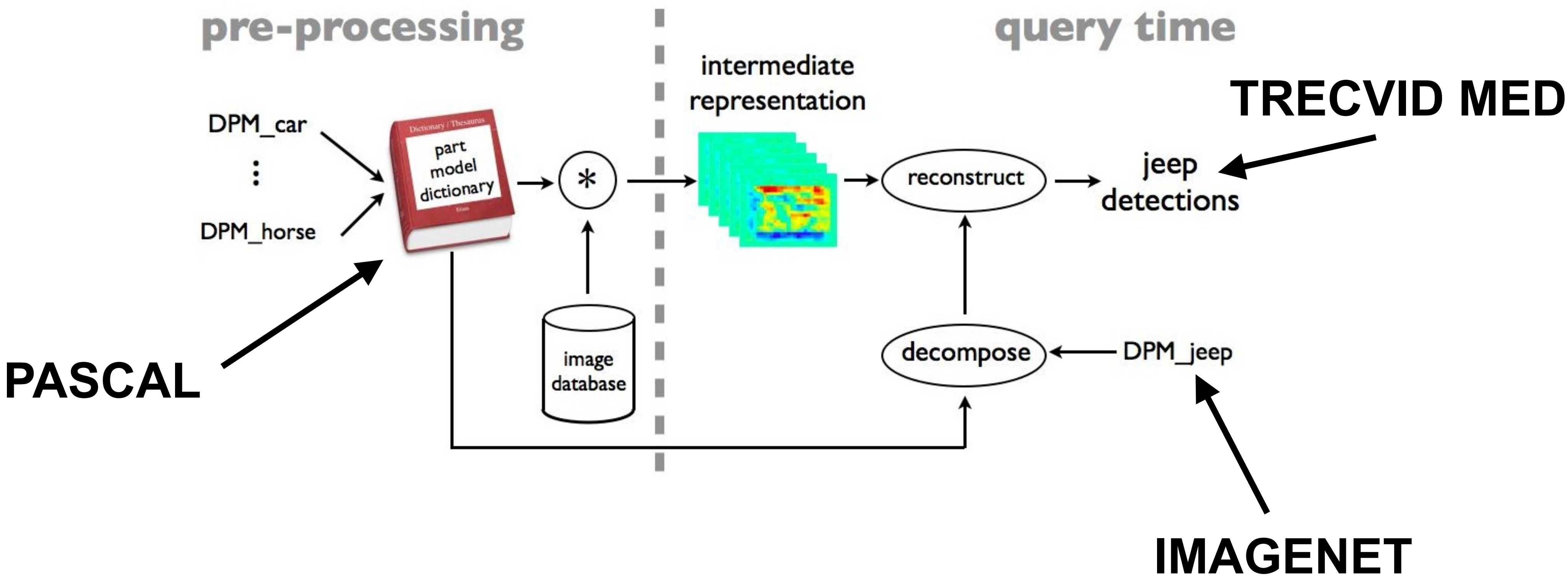
Generalization Performance



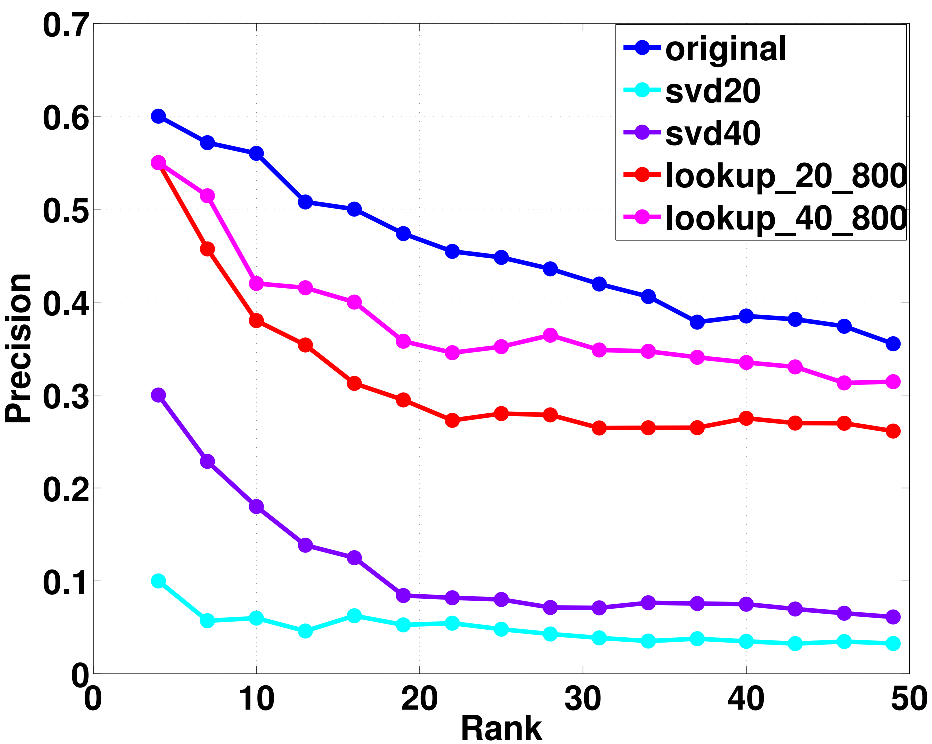
(a) 20 PASCAL categories (b) 9 PASCAL categories (c) 9 ImageNet categories

- averaged AP performance vs. number of coefficients
- training dictionary on PASCAL; learn and test on imagenet

Generalization to New Classes: Retrieval Setting

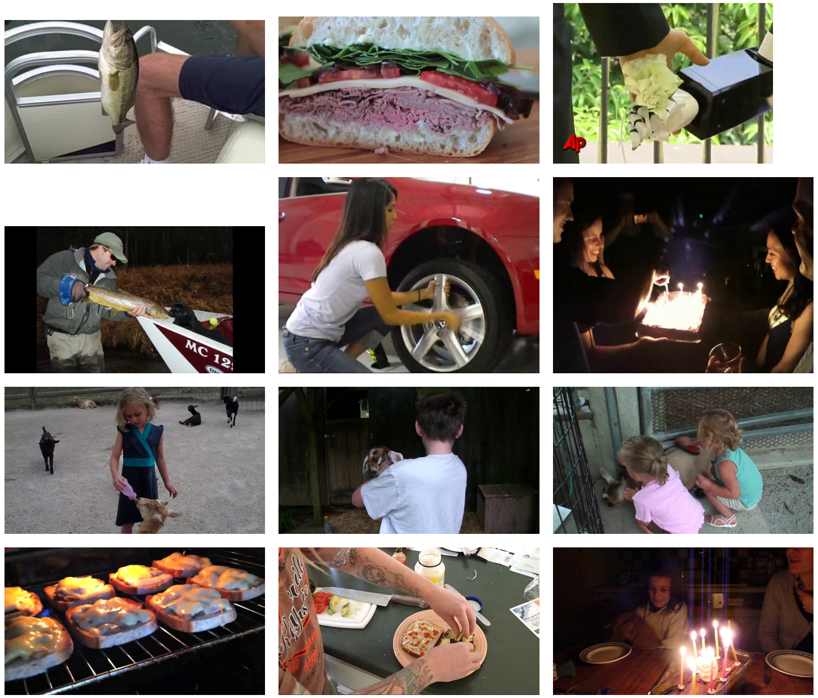


Final Experiment



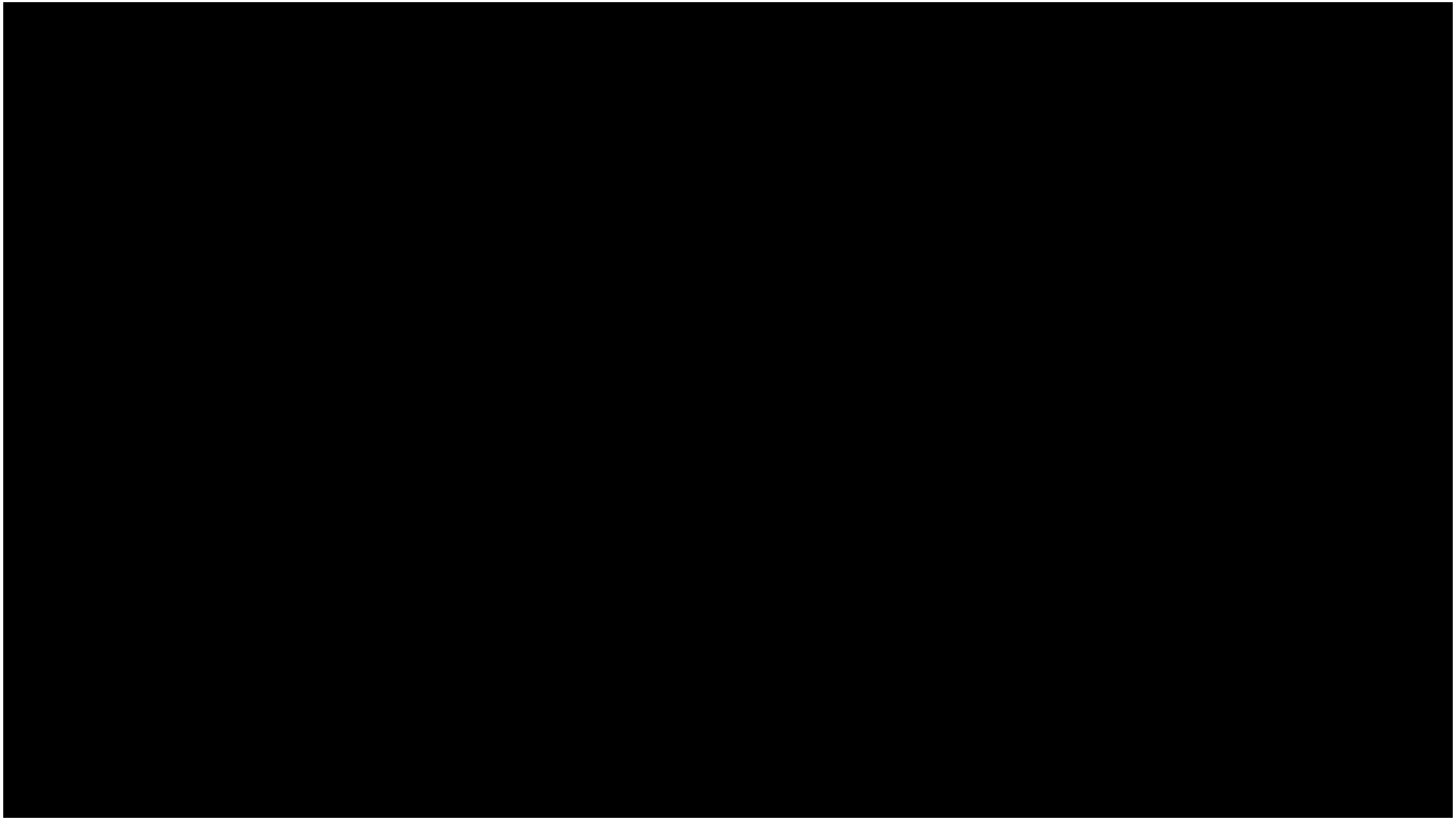
- PASCAL -> Dictionary
- Imagenet -> Object Classes
- Detection on TRECVID
- Precompute Representation
- 35x speedup with GPU

TRECVID dataset



manually selected object classes related to events: sailboat, bread, cake, candle, fish, goat, jeep, scissors, and tire

Demo



Conclusion

- General formulation to share computation and exploit correlations
- Total detection speedup up to 25x
- PASCAL VOC detection in real-time
- Offline-retrieval task up to 35x speedup

Rodrigo Benenson

KATHOLIEKE UNIVERSITEIT LEUVEN

VISICS

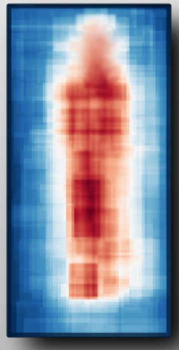
100 classifiers to detect a single class

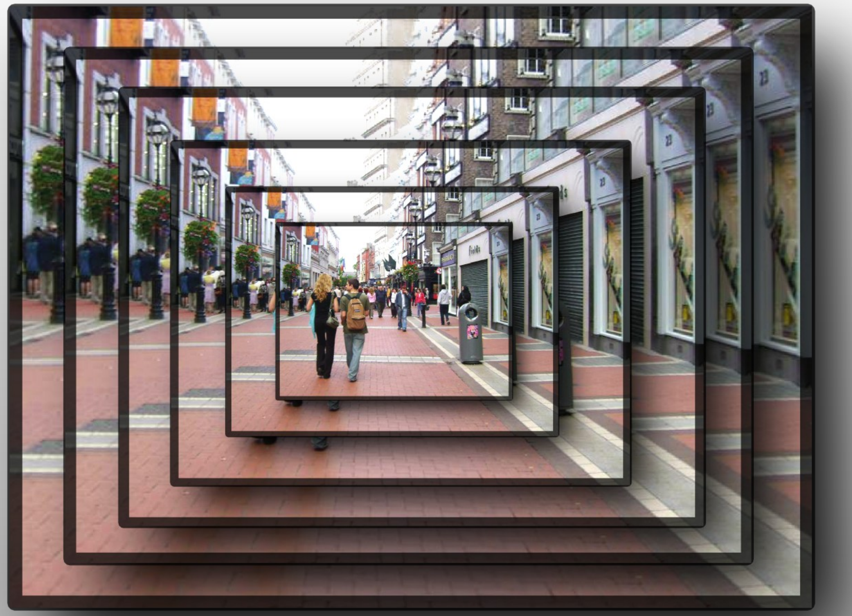
Rodrigo Benenson

KATHOLIEKE UNIVERSITEIT LEUVEN

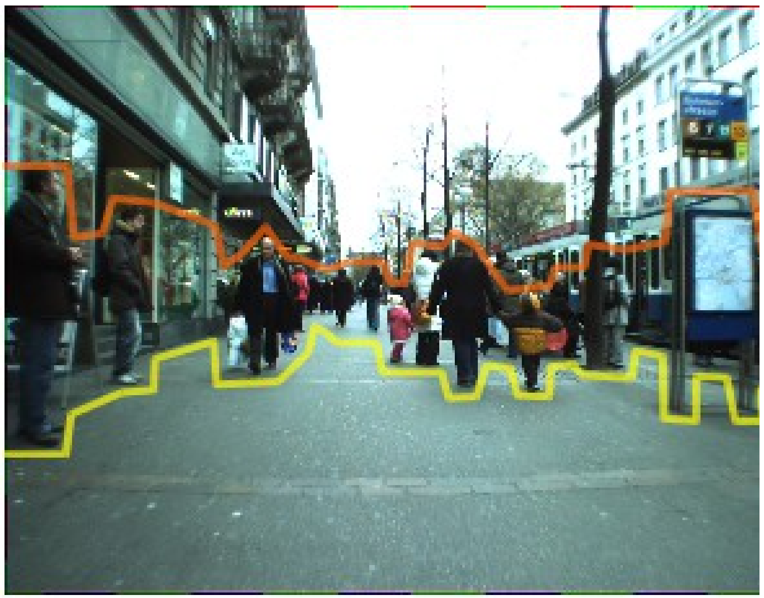
VISICS

Part III
50 classifiers are faster than 1

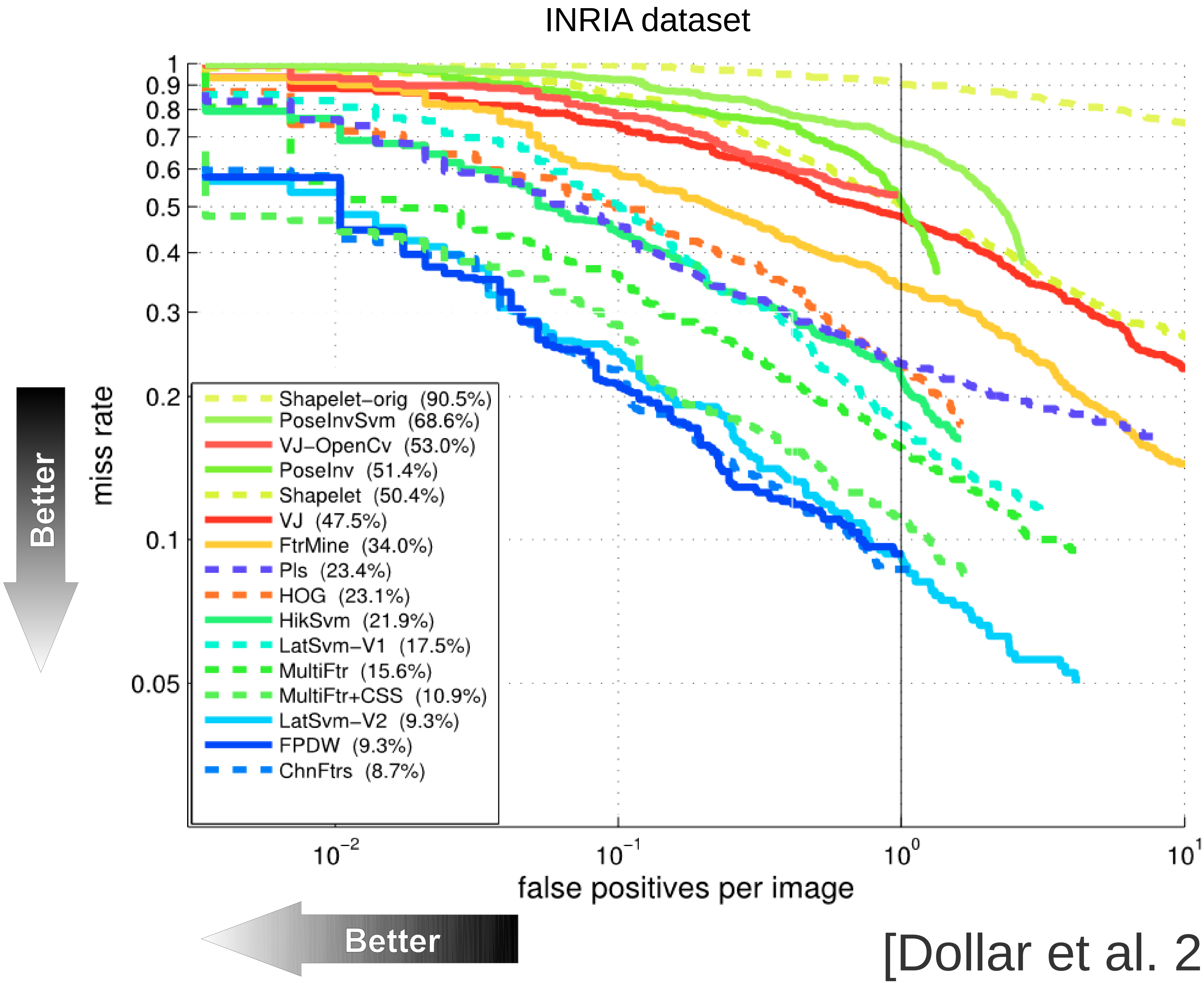




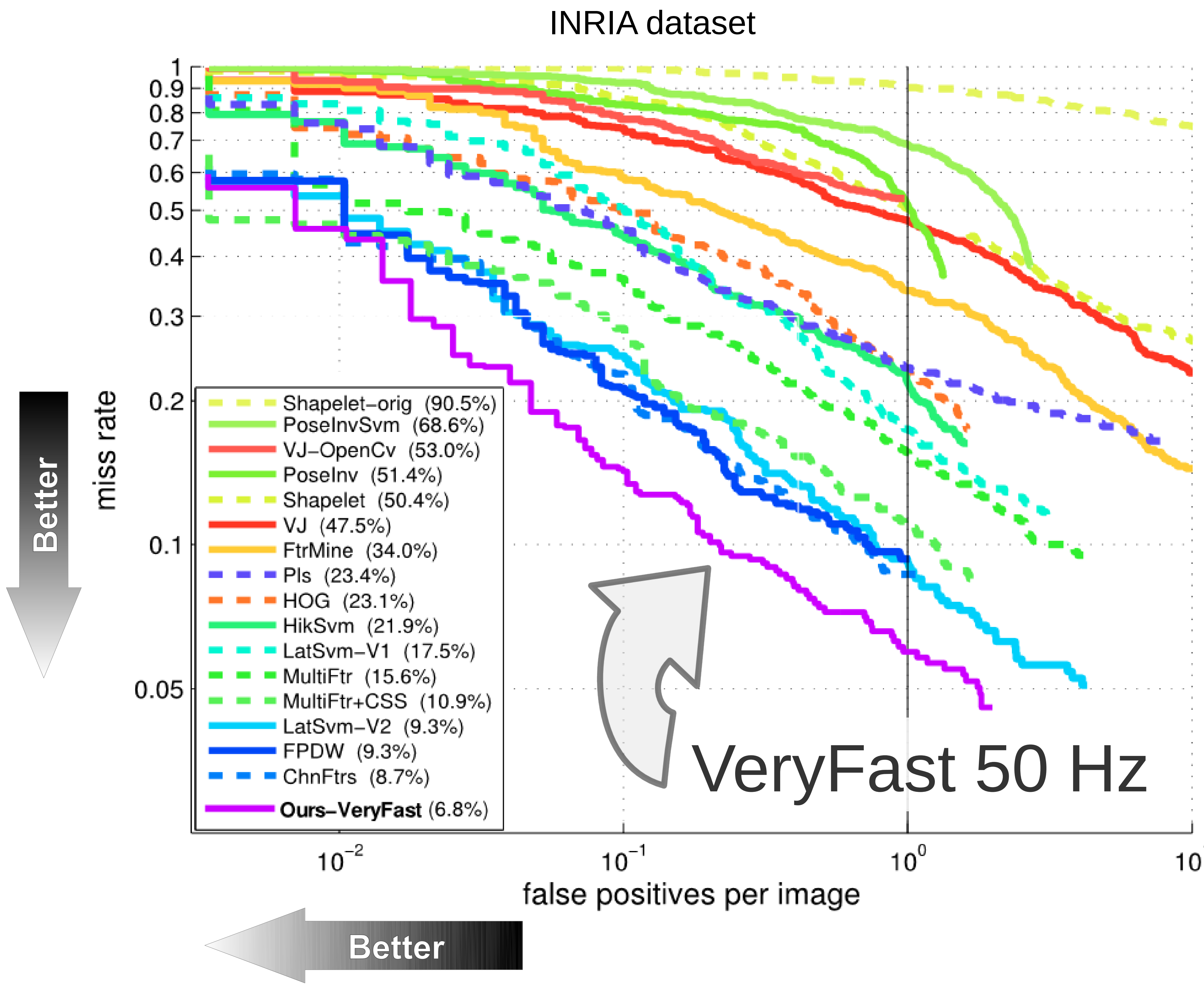
Third version: detector is the bottleneck

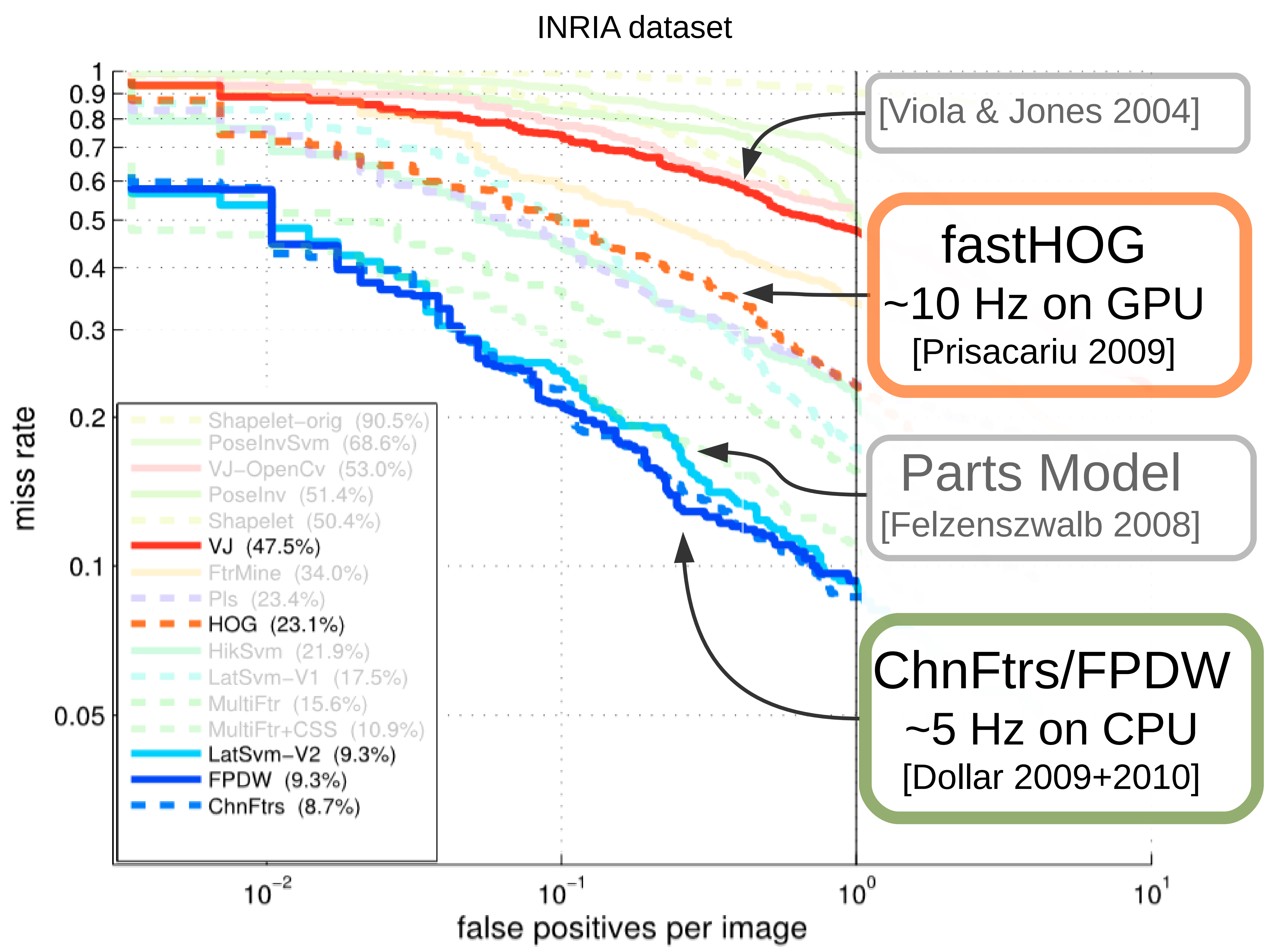


Milliseconds per frame



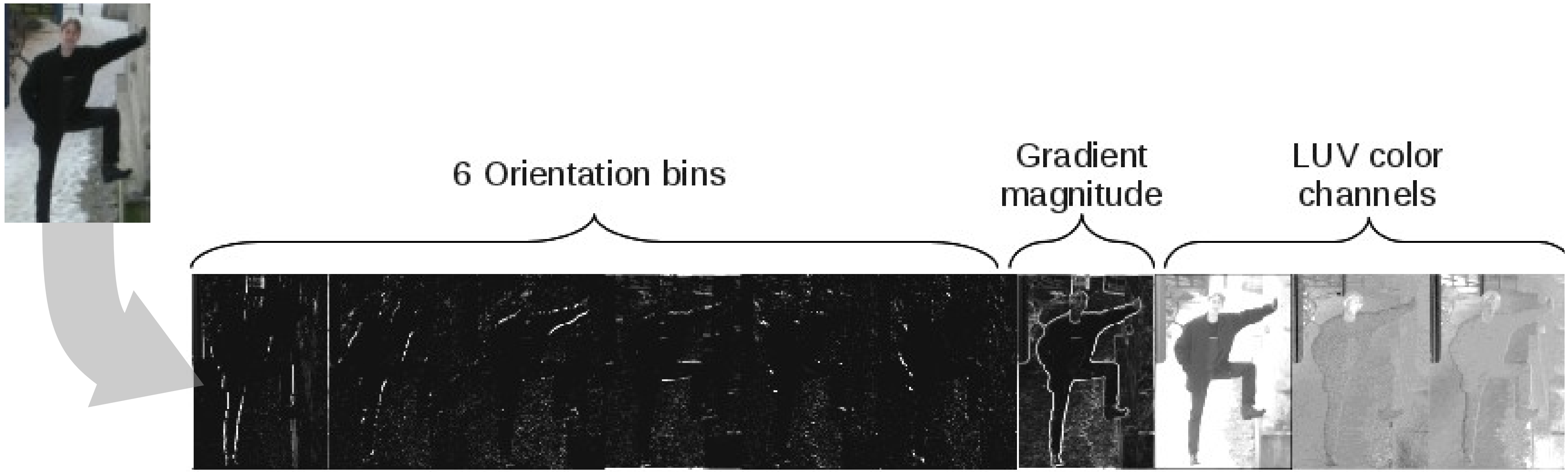
[Dollar et al. 2011]

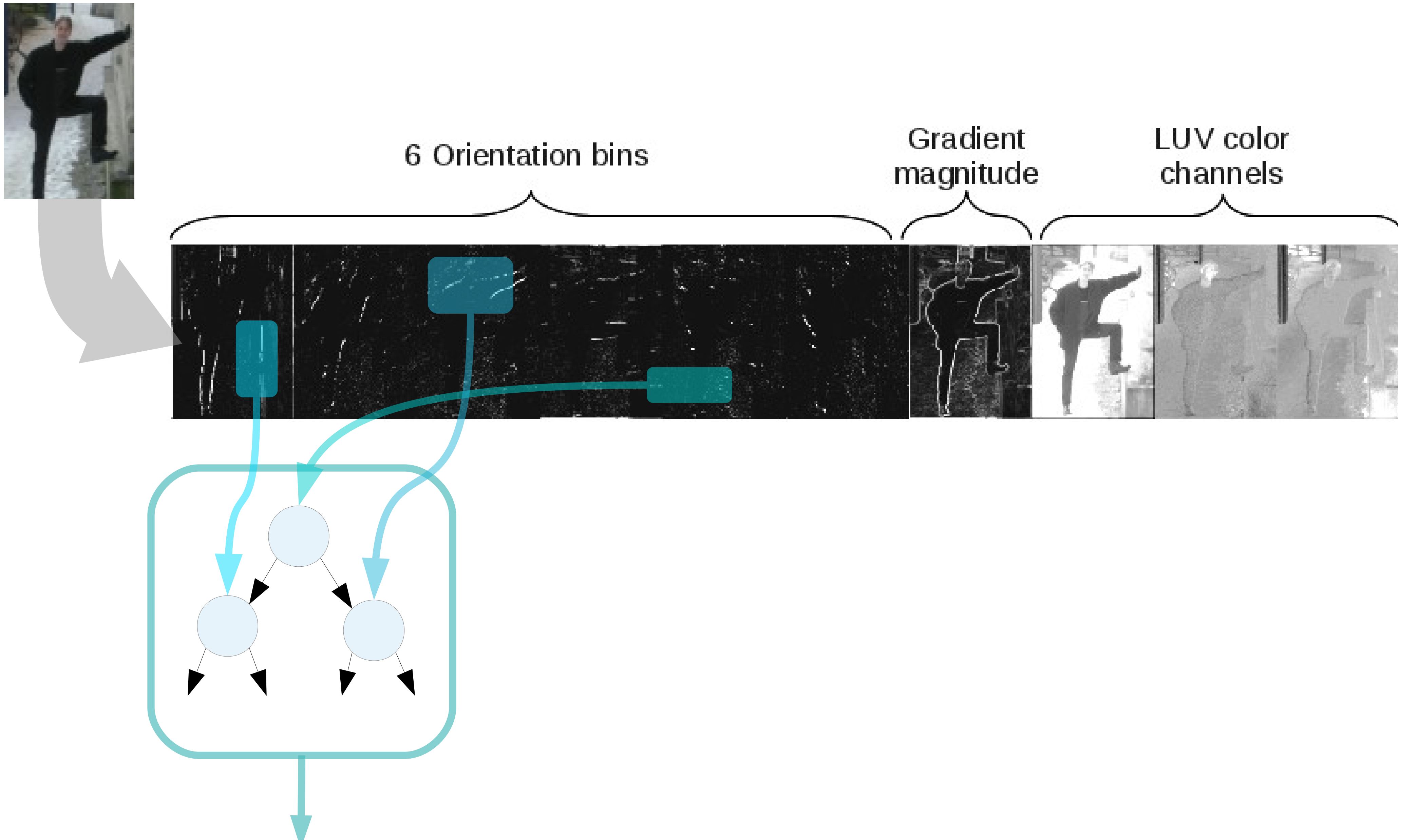

INRIA dataset
miss rate
false positives per image
Better
Better
VeryFast 50 Hz
Shapelet−orig (90.5%)
PoseInvSvm (68.6%)
VJ−OpenCv (53.0%)
PoseInv (51.4%)
Shapelet (50.4%)
VJ (47.5%)
FtrMine (34.0%)
Pls (23.4%)
HOG (23.1%)
HikSvm (21.9%)
LatSvm−V1 (17.5%)
MultiFtr (15.6%)
MultiFtr+CSS (10.9%)
LatSvm−V2 (9.3%)
FPDW (9.3%)
ChnFtrs (8.7%)
Ours−VeryFast (6.8%)



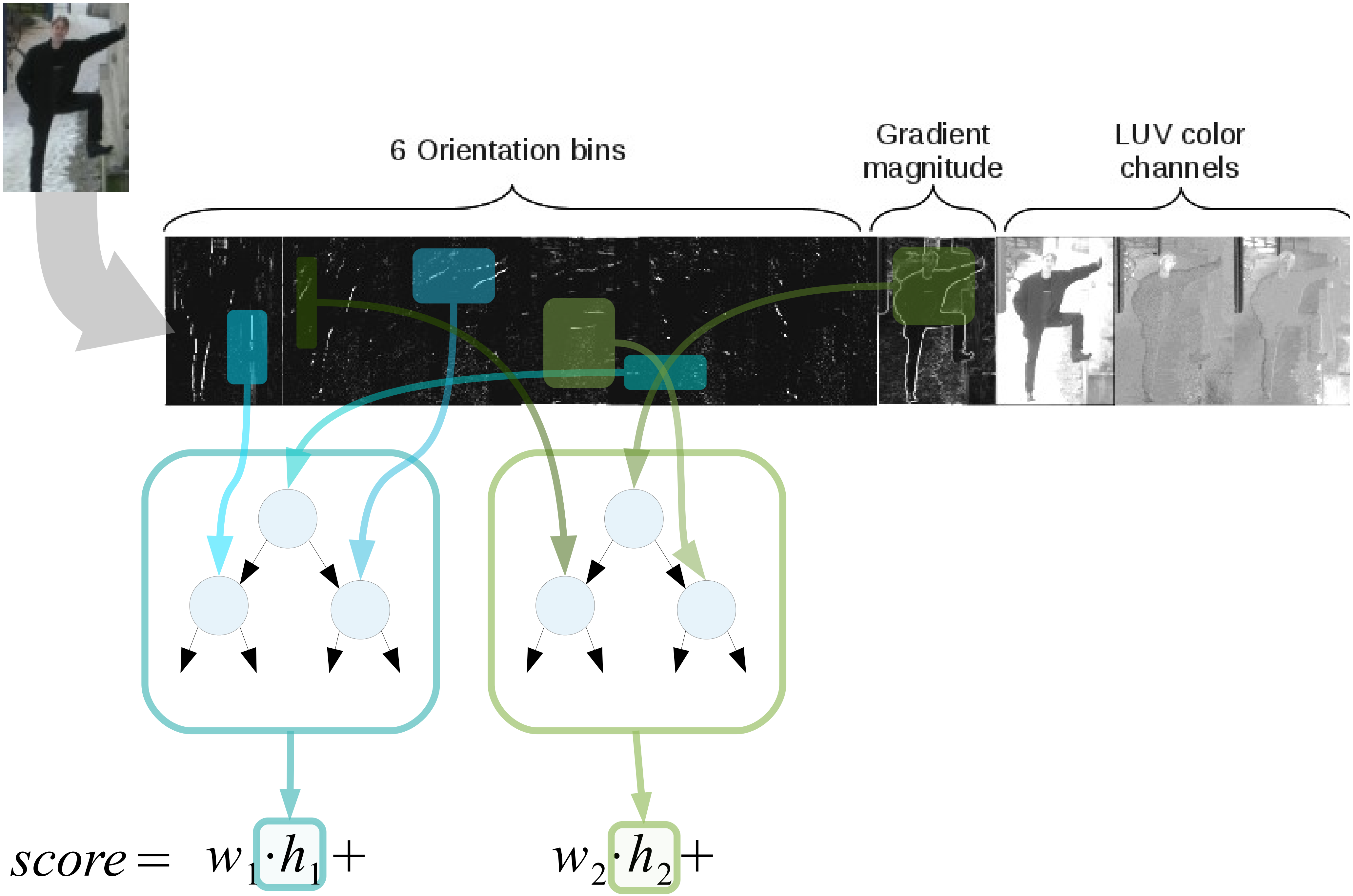
INRIA dataset
miss rate
false positives per image
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.05
10^{-2}
10^{-1}
10^{0}
10^{1}
Shapelet−orig (90.5%)
PoseInvSvm (68.6%)
VJ−OpenCv (53.0%)
PoseInv (51.4%)
Shapelet (50.4%)
VJ (47.5%)
FtrMine (34.0%)
Pls (23.4%)
HOG (23.1%)
HikSvm (21.9%)
LatSvm−V1 (17.5%)
MultiFtr (15.6%)
MultiFtr+CSS (10.9%)
LatSvm−V2 (9.3%)
FPDW (9.3%)
ChnFtrs (8.7%)
[Viola & Jones 2004]
fastHOG
~10 Hz on GPU
[Prisacariu 2009]
Parts Model
[Felzenszwalb 2008]
ChnFtrs/FPDW
~5 Hz on CPU
[Dollar 2009+2010]

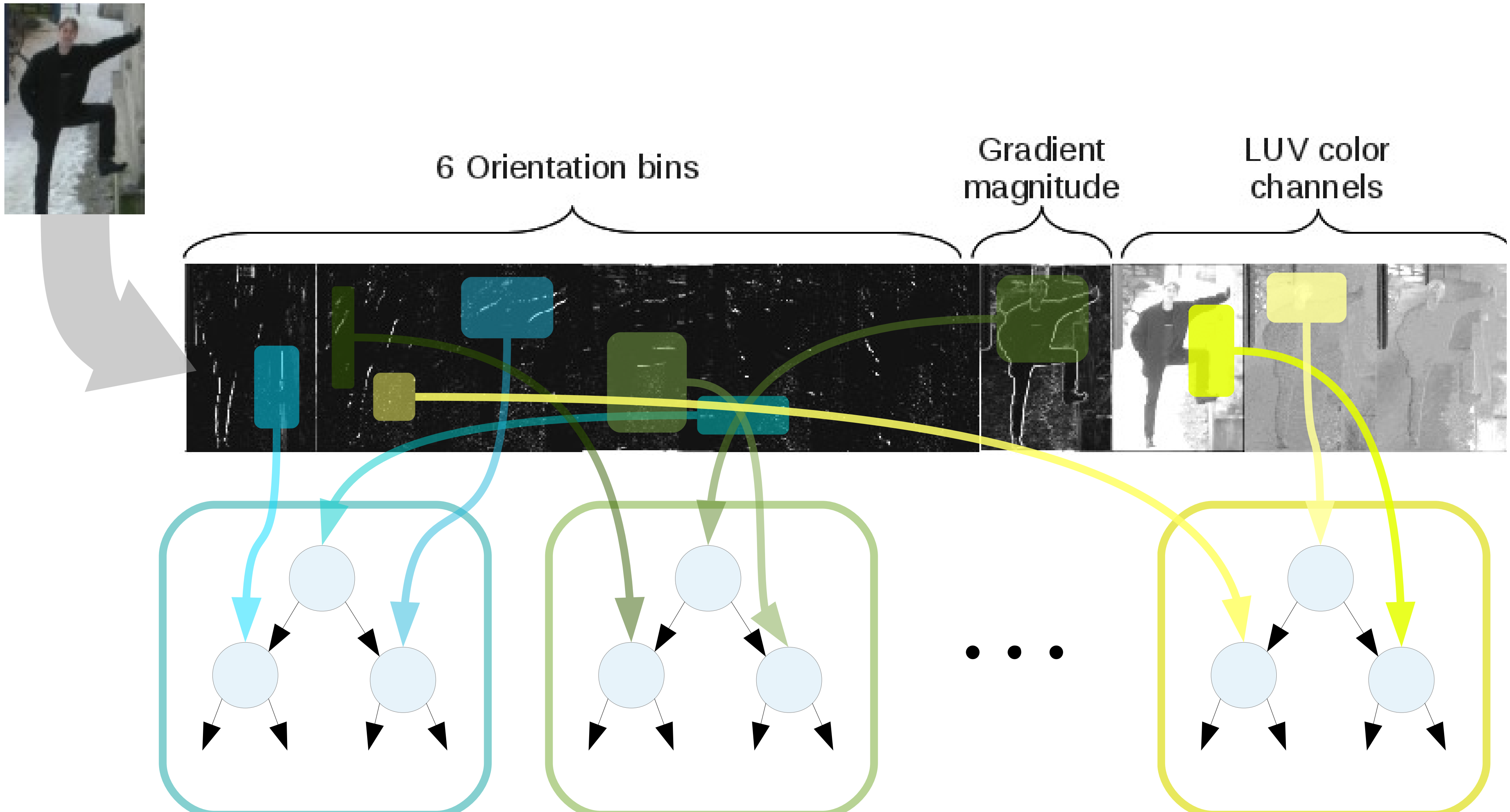
EXPLAIN HOW CHNFTRS WORK
(versus HOG)
mention speed on GPU: ~3 Hz


6 Orientation bins
Gradient magnitude
LUV color channels



$$score = w_1 \cdot \boxed{h_1} +$$

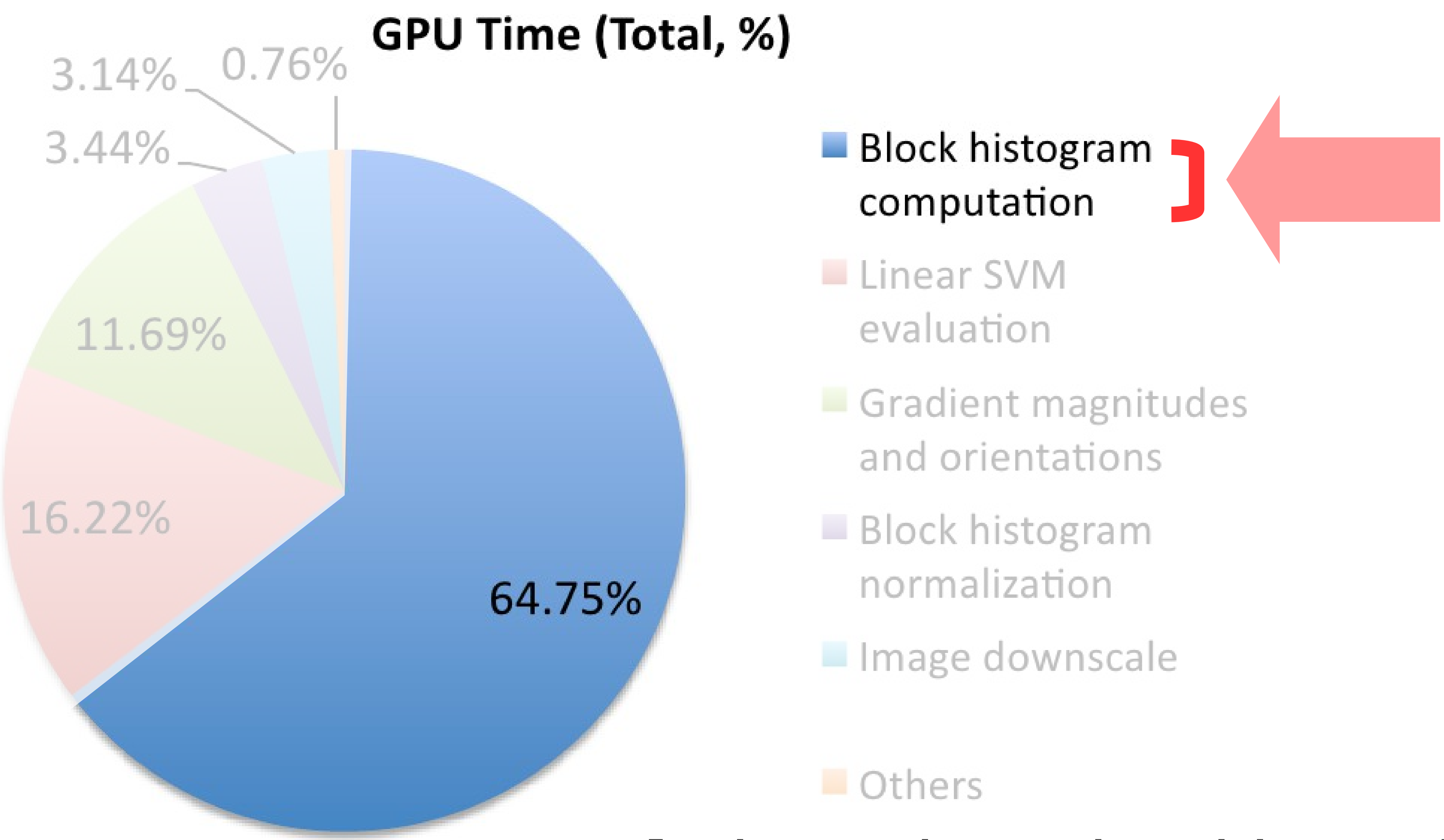

6 Orientation bins
Gradient magnitude
LUV color channels
score = w1·h1 + w2·h2 +



$$score = w_1 \cdot \boxed{h_1} + w_2 \cdot \boxed{h_2} + \cdots + w_N \cdot \boxed{h_N}$$

[ChnFtrs, Dollar et al. 2009] (~3 Hz on GPU)

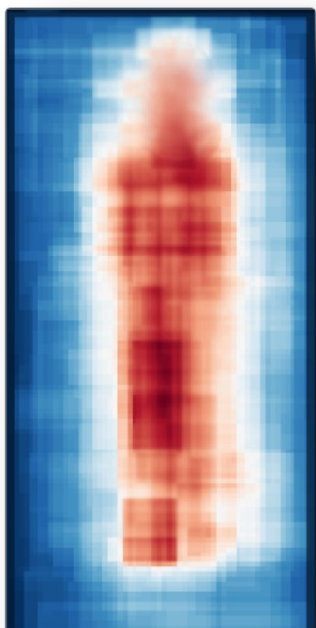
What slows down fastHOG ?

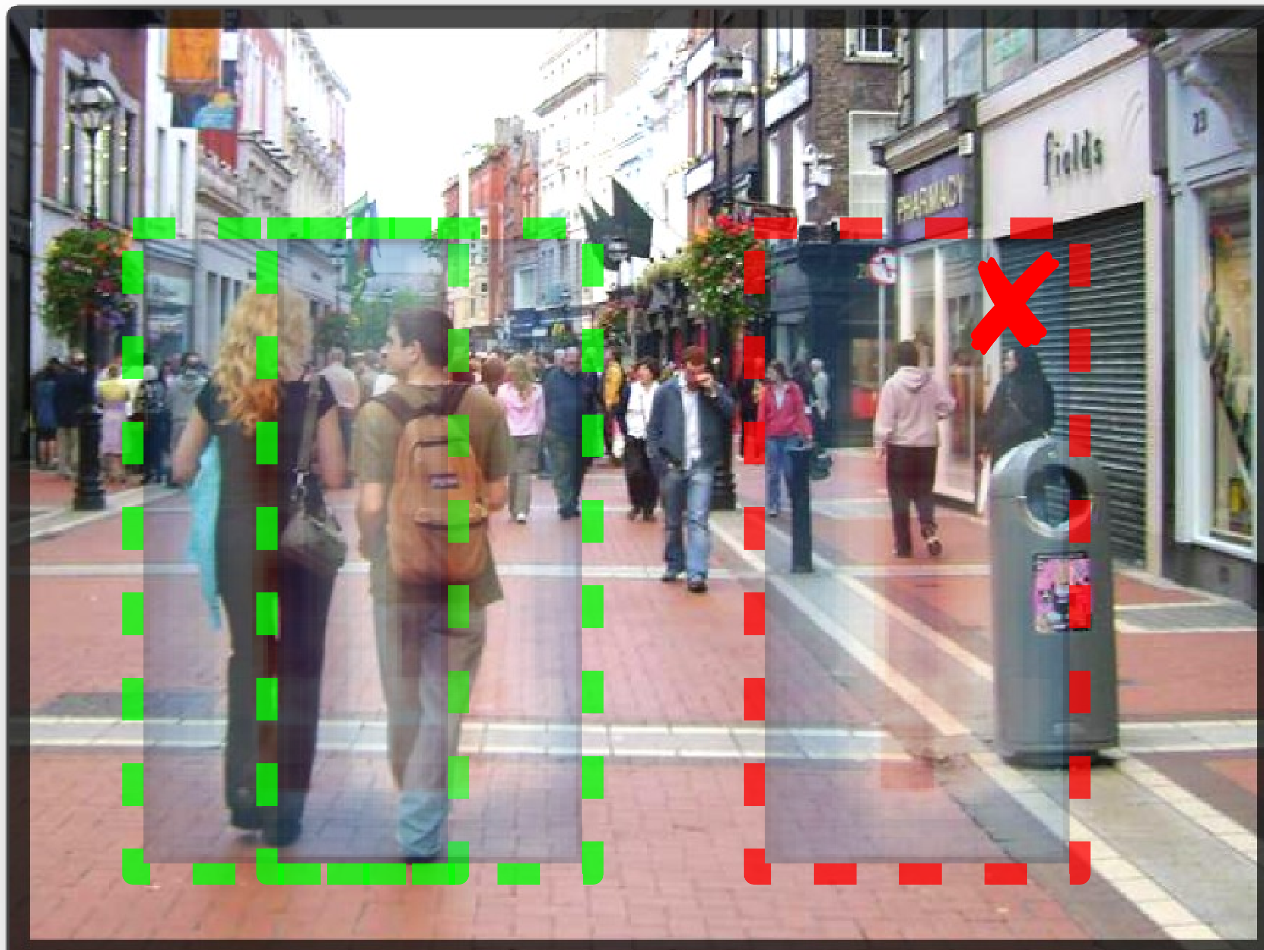


[Prisacariu and Reid 2009]

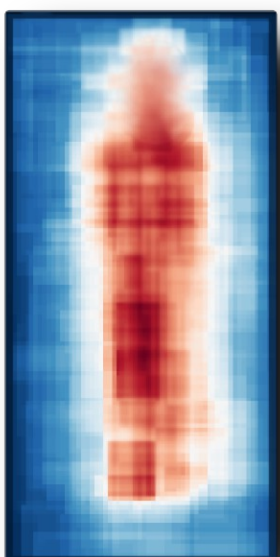
How to make
features computation
faster ?

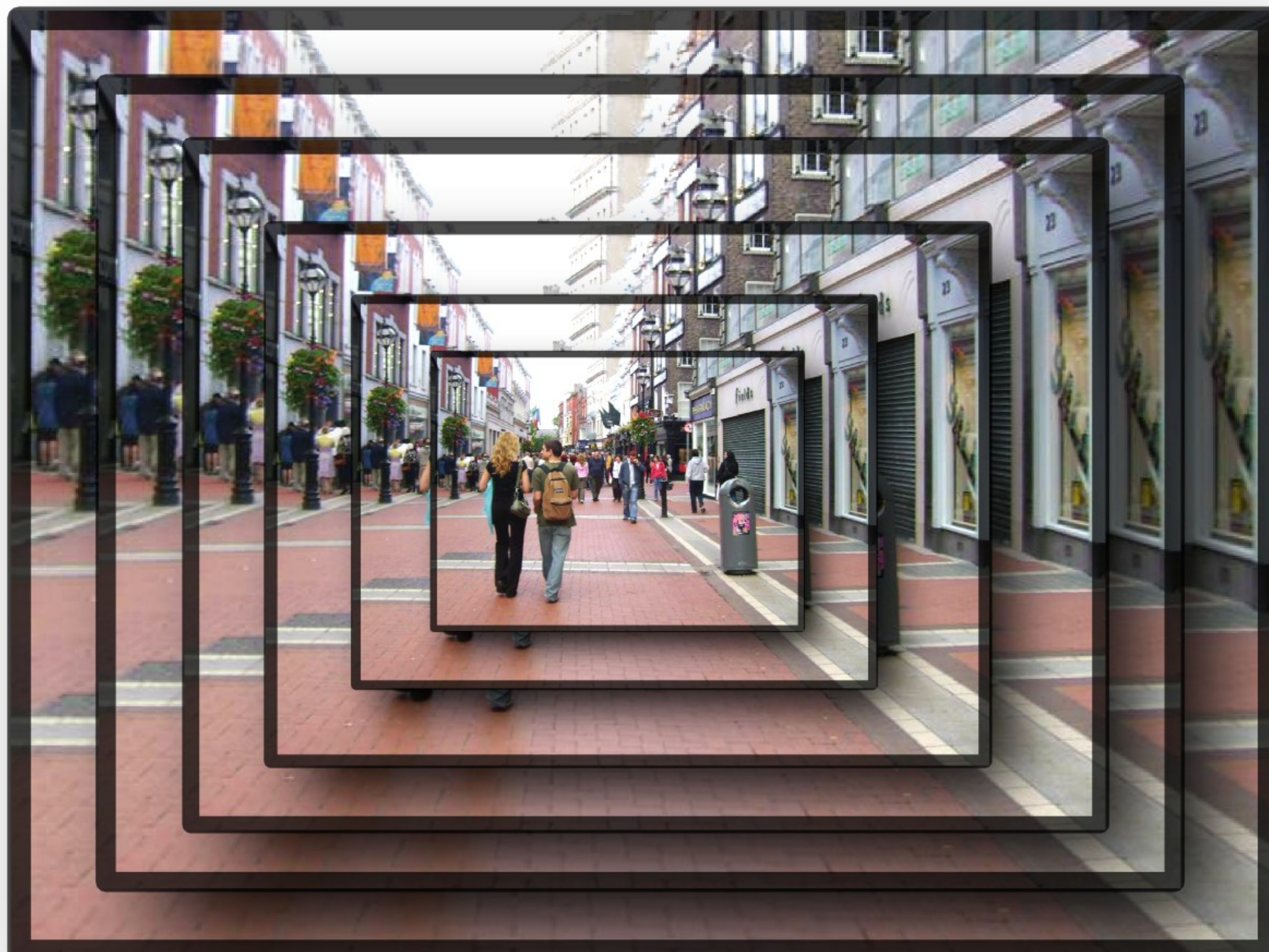
One template cannot detect at multiple scales





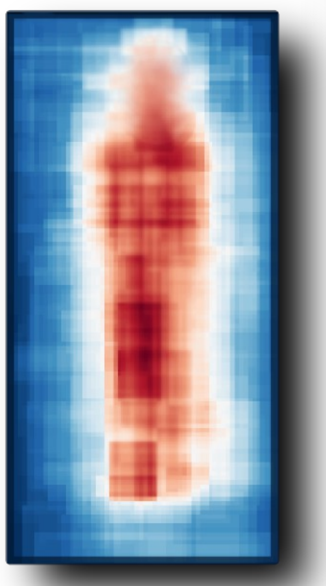
Traditionally, features are computed many times

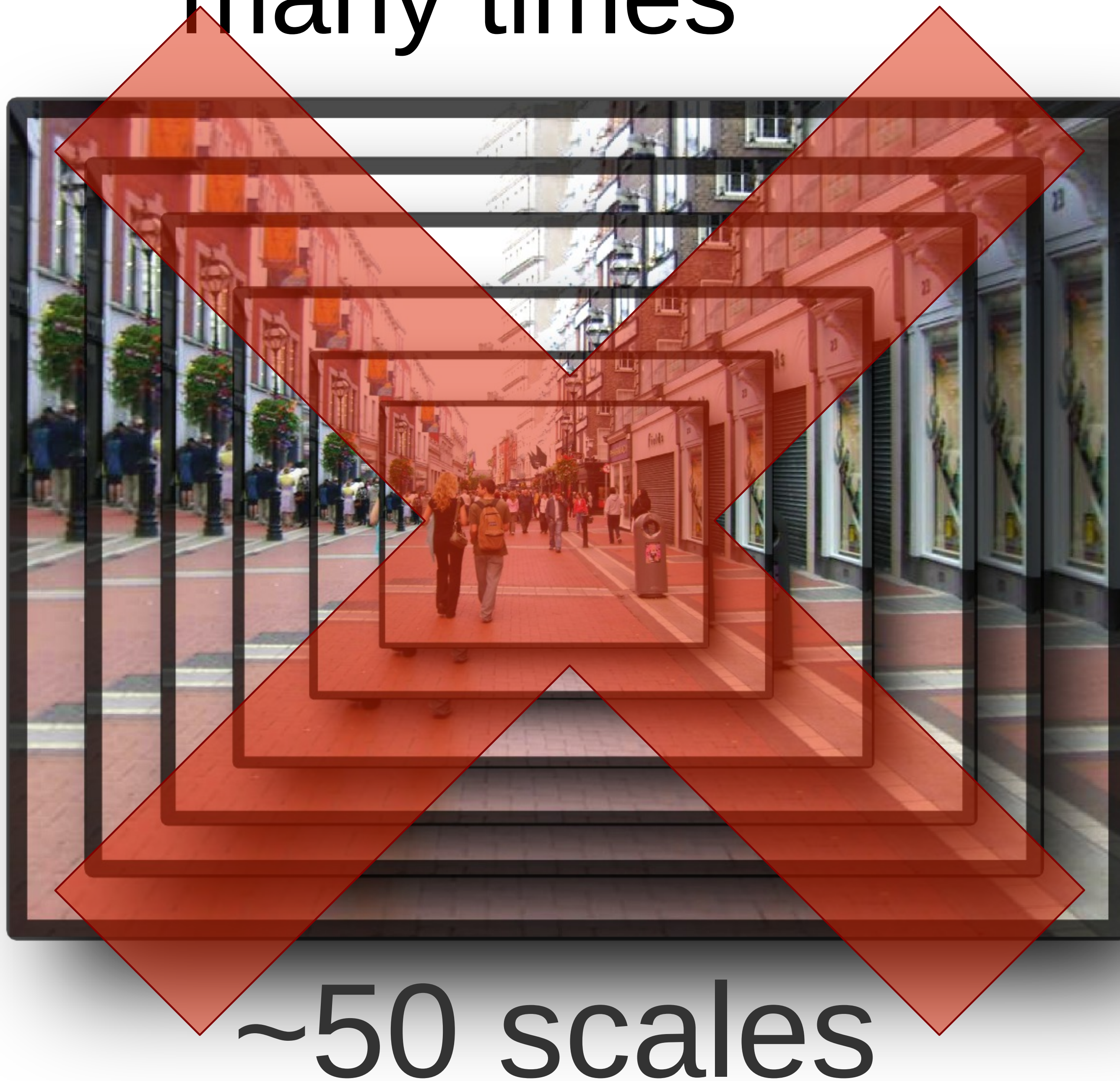




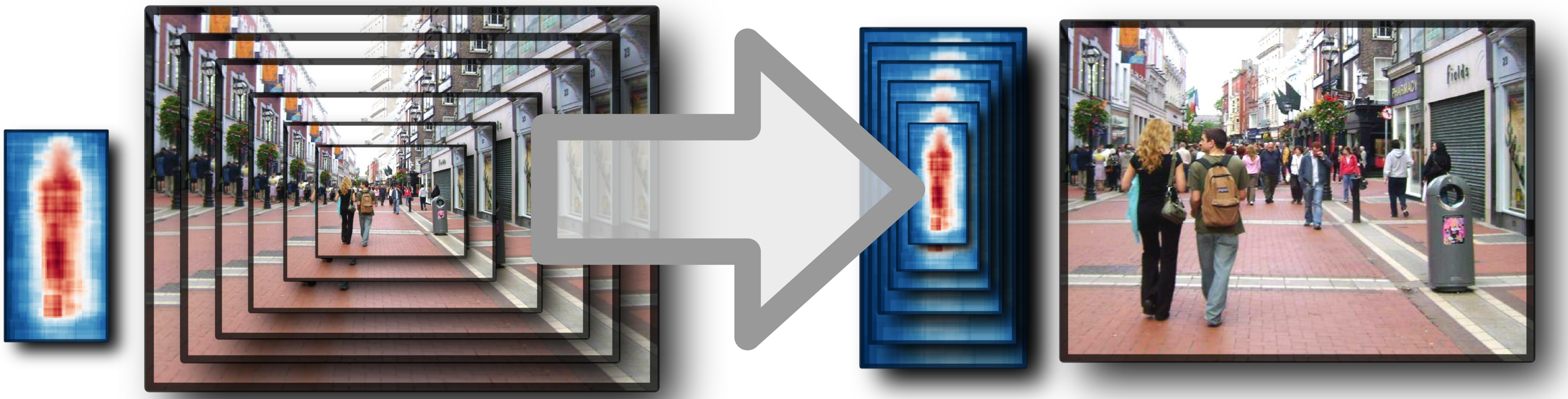
~50 scales

Traditionally, features are computed many times





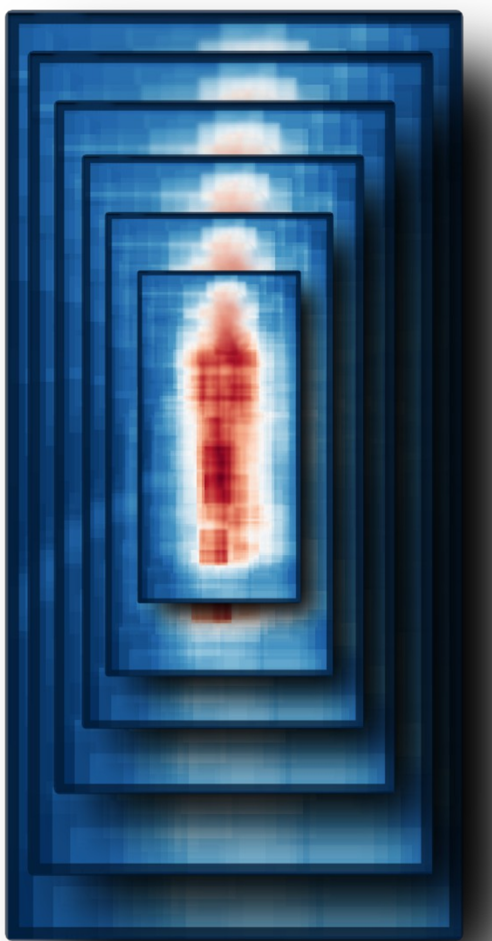
We invert the relation



1 model,
50 image scales

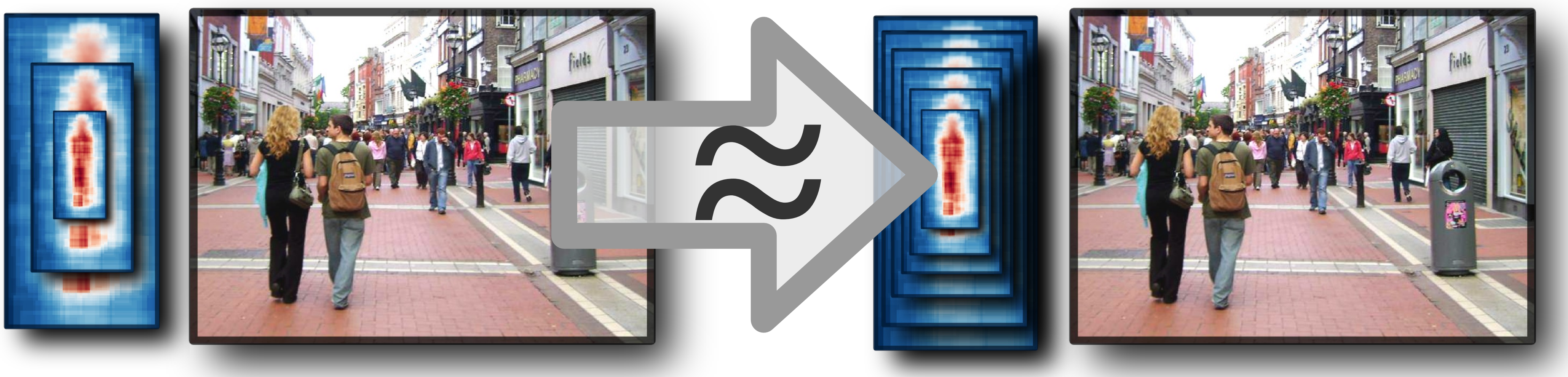
50 models,
1 image scale

Training one model per scale is too expensive



~50 scales

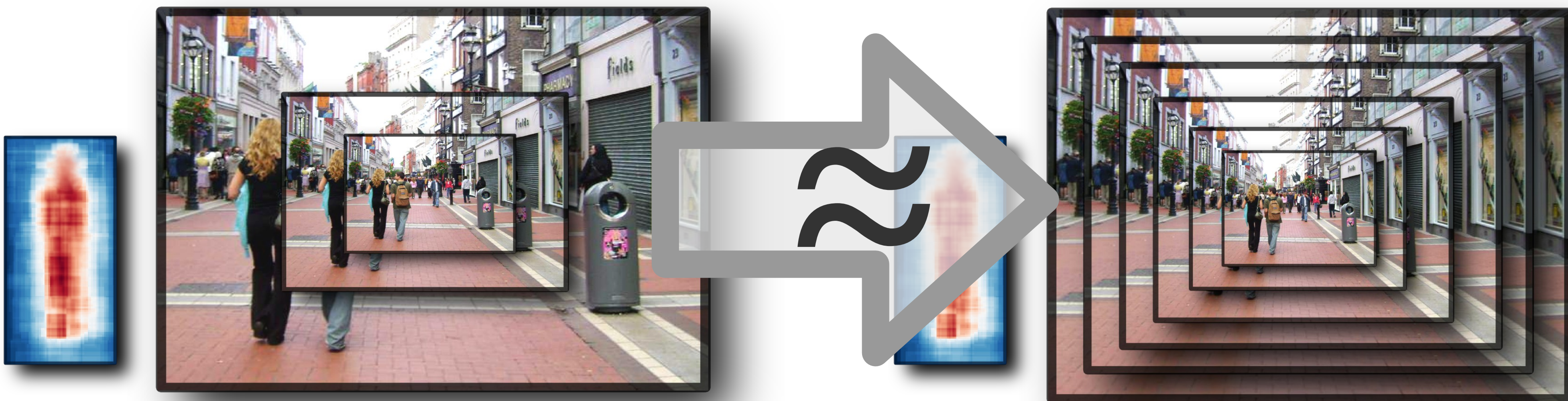
We propose a method to reduce training time 10x



5 models,
1 image scale

50 models,
1 image scale

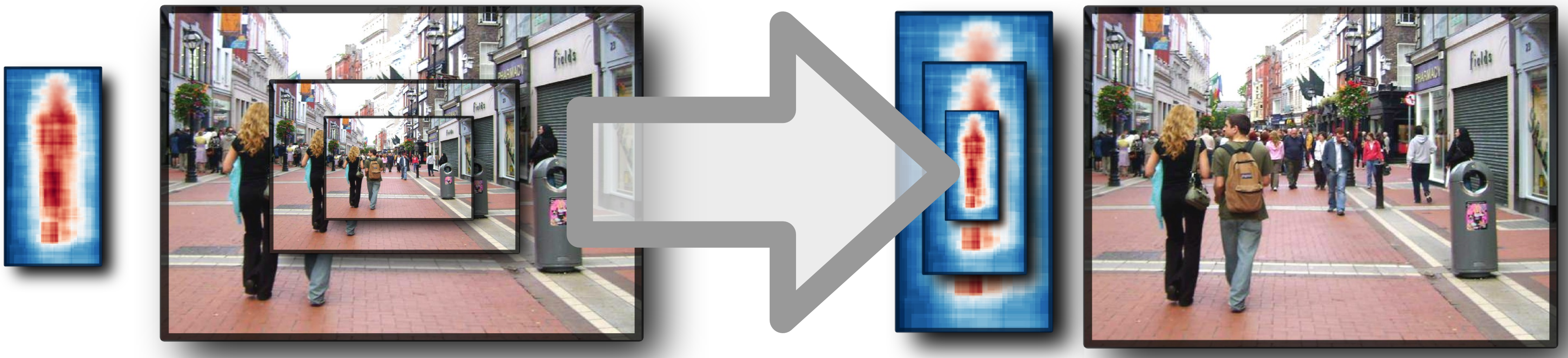
Features can be approximated across scales



~5 scales ~50 scales

[Dollar et al. 2010]

We transfer test time computation to training time



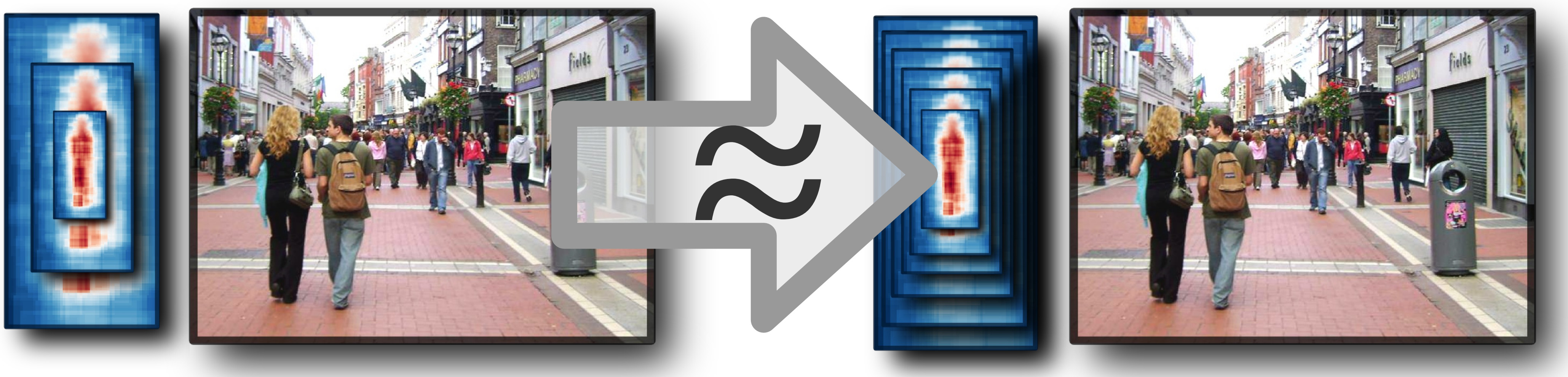
1 model,
5 image scales

5 models,
1 image scale

(3x reduction in features computation)

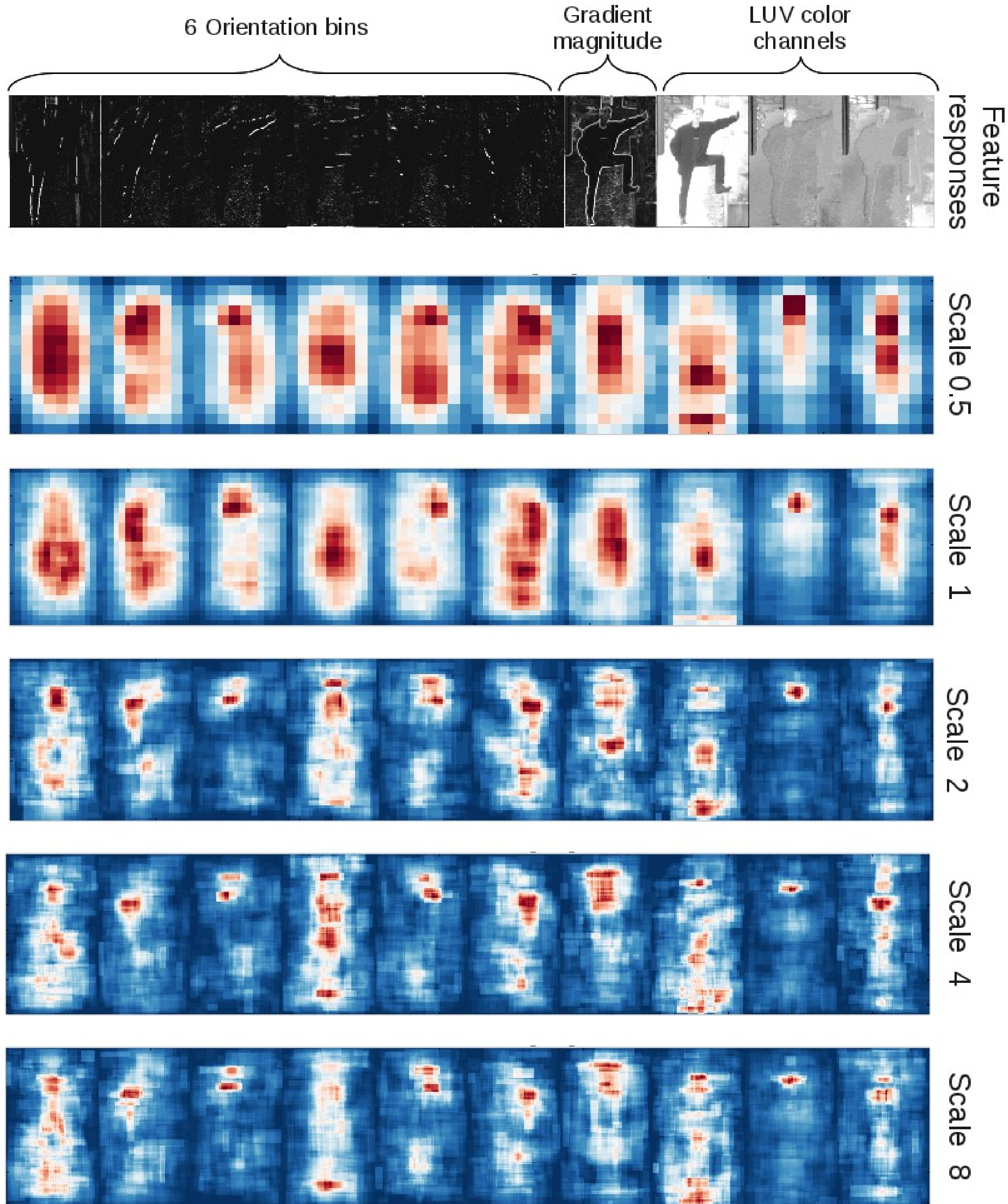
Key insight

At runtime, we use as many models as scales

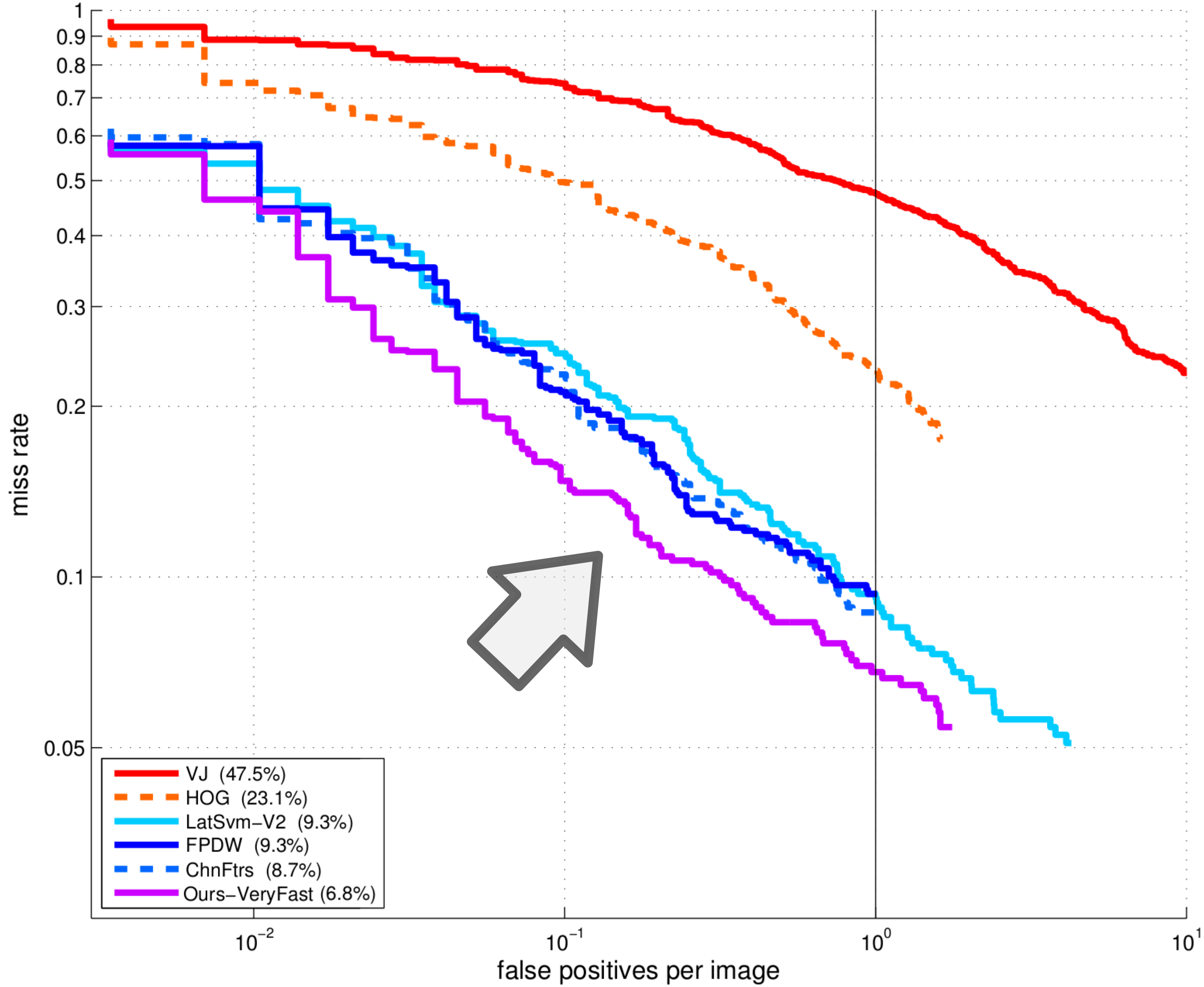


5 models,
1 image scale

50 models,
1 image scale


6 Orientation bins
Gradient magnitude
LUV color channels
Feature responses
Scale 0.5
Scale 1
Scale 2
Scale 4
Scale 8

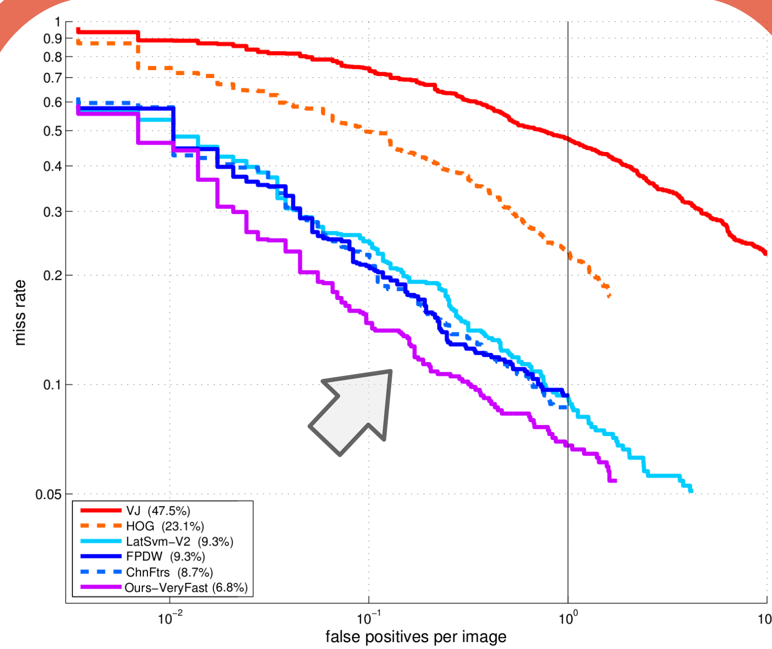
Detecting without resizing provides quality

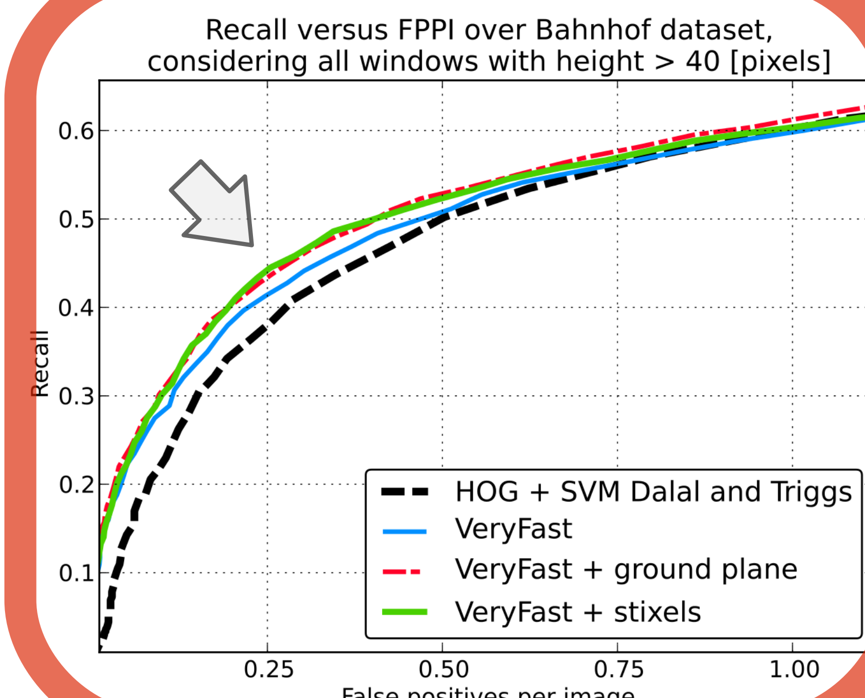


Detecting without resizing provides speed

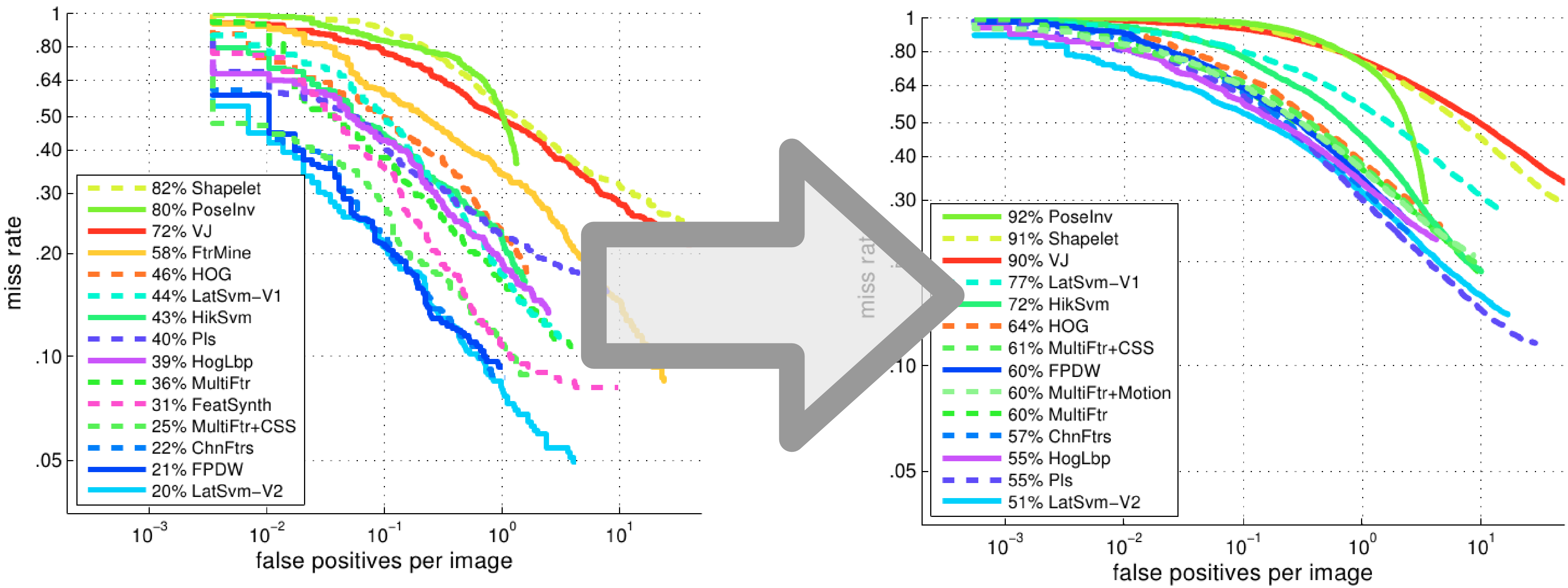
Monocular
50 Hz

Stereo *with stixels*
160 Hz, 80 Hz on laptop





ETH's dataset results have less variance than INRIA's



INRIA

ETH

Monocular

Stereo

Tasks for the exercise

- Take version 0 of Rodrigos code and optimize
- This should roughly take you from 1Hz to 50Hz
- Note that there is a license on the code!

- Current code is cmake & linux
- Compiling on win might be tricky
- If you have an MPI or IMPRS account
  - you can use one of our GPU machines (ruegen, ganymede (open))
- If there is no way for you - tell us and we try to get you an account

Tasks for the exercise

- figure out right number of threads
  - CUDA occupancy calculator (and http://en.wikipedia.org/wiki/CUDA)
  - nvcc compiler flag --ptxas-options=-v
- implement cascade
- parallelize execution over scales
- pre-read all 3 decision values for tree stump
- experiment with texture and global memory (in the code)
- look into additional optimizations
- extra points for fixing exception at the end of code ;)
- vary stride

- always check output (detections)