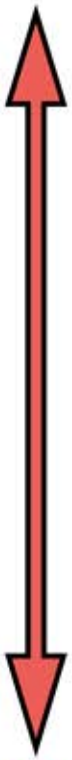
30Hz Object Detection with DPM V5



ALL 20 PASCAL templates
in 0.03s per frame

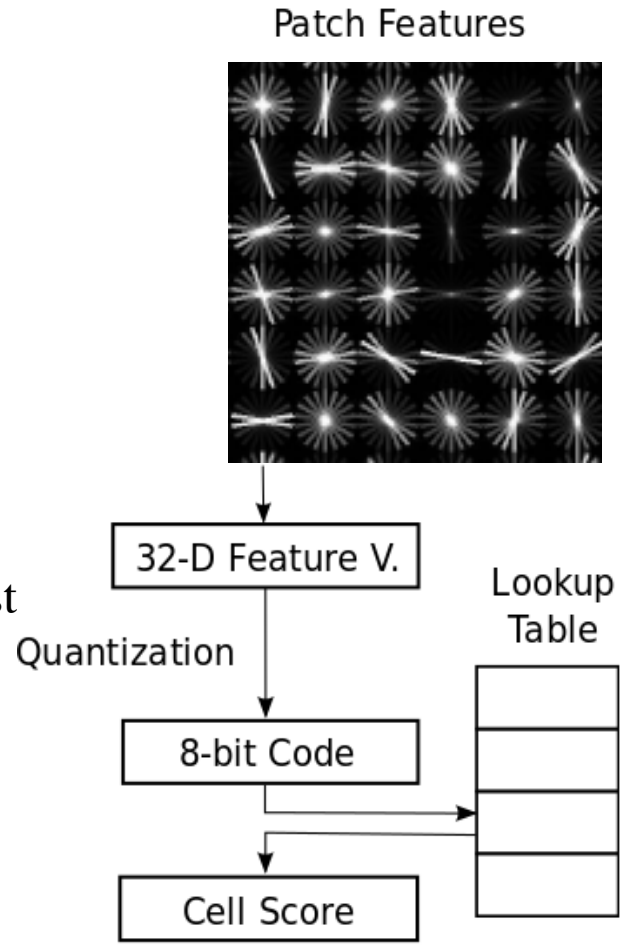
Amin Sadeghi
David Forsyth

ILLINOI
UNIVERSITY OF ILLINOIS AT URBANA-CHAMP

Why speed up DPM?

- Worth doing, because DPM is effective, mature and stable
  - yes, I know that there are more accurate CNN's - but they keep changing!
- Fast DPM V5 opens a door for applications
  - on mobile devices
  - with large datasets
  - with previously implausible numbers of templates
- Constraint: No weird hardware
  - to avoid being locked out of applications

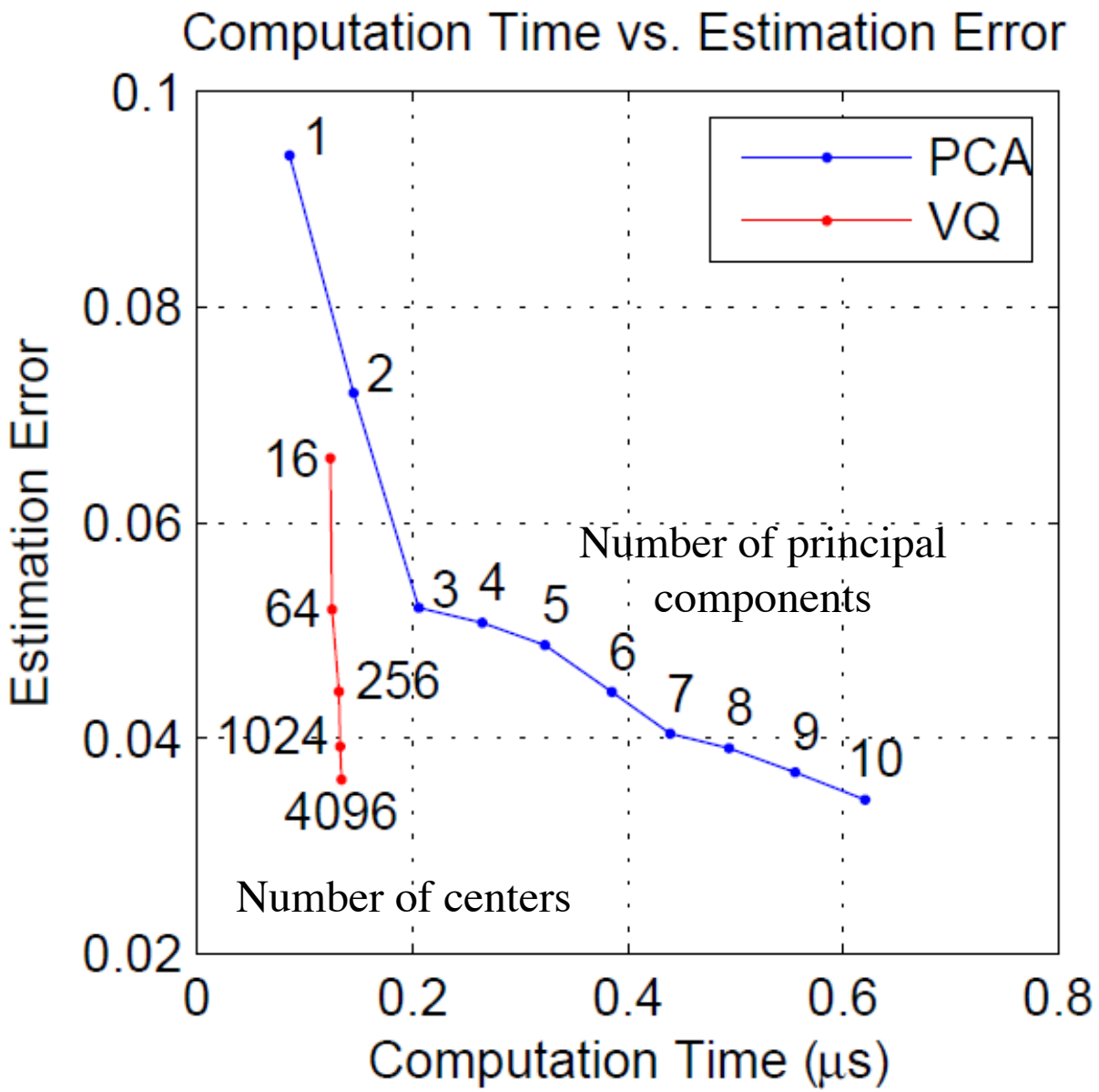
Review



- What needs to be done?
  - Extract HOG features (used to be thought of as fast
  - Apply templates (used to be thought of as slow)
- So, make “Apply templates” faster
  - by vector quantizing the HOG cells
  - benefit:
    - replacing multiply+add with lookup
  - cost:
    - less accurate template values (not much; doesn’t seem to matter)
    - construct VQ (offline - no issue)
    - find nearest neighbor for HOG

Estimation error from VQ

Absolute value of error



Most time is going into HOG computation

Table from Sadeghi Forsyth 13

	HOG features	per image	per (image×category)	per category
Original DPM [2]	40ms	0ms	665ms	0ms
DPM Cascade [7]	40ms	6ms	84ms	3ms
FFLD [9]	40ms	7ms	91ms	43ms
Our+rescoring	40ms	76ms	21ms	6ms
Our-rescoring	40ms	76ms	9ms	6ms

Image feature times

Template work

PASCAL'07 Detection Challenge, Intel Xeon E5-1650 processor, All algorithms using 6 cores

Desirable Features of any Approach

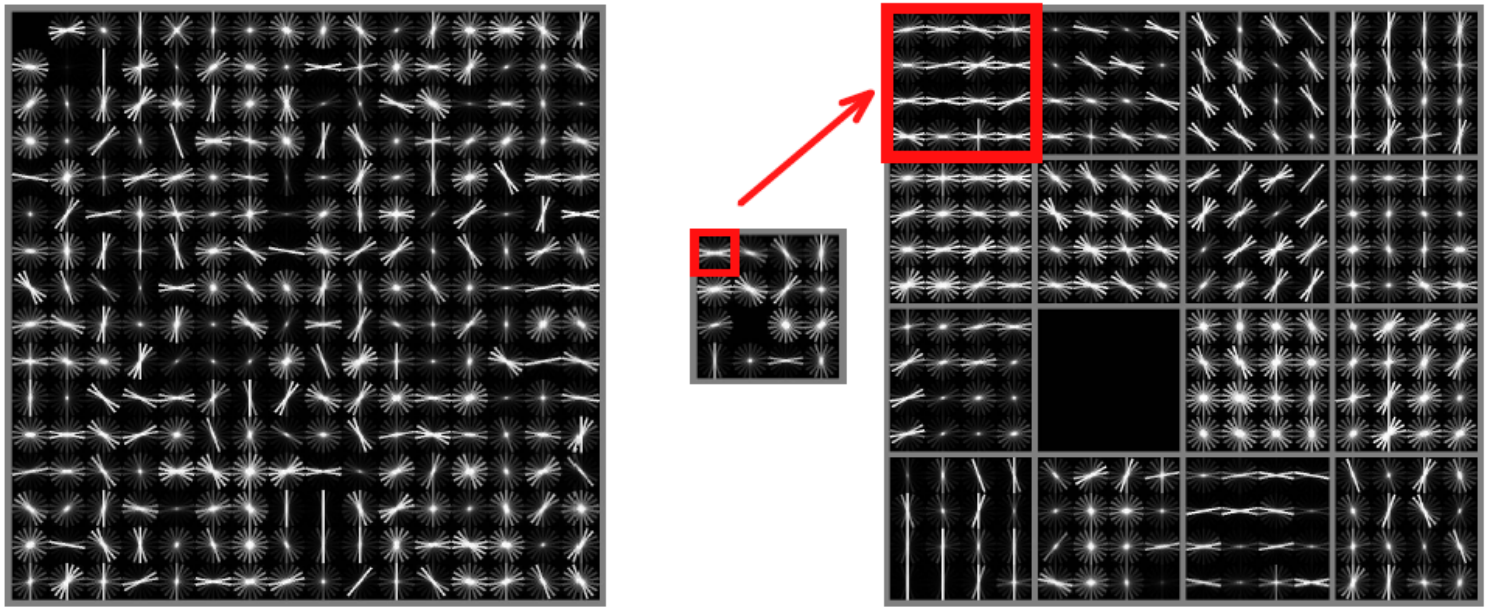
- Can work with legacy templates
  - No retraining required
- Random access to image
  - otherwise we're locked out of pruning/cascade strategies
- Can trade accuracy vs. speed
  - so application developers can look for a sweet spot
- Anytime property
  - so that plausible/tolerable results are returned whenever interrupted

Strategies to speed up

- Accept some loss of numerical precision
  - quantize cells hierarchically
- Manage Scale carefully
  - More templates, fewer HOG features
- Prioritize
  - Avoid evaluating templates at all locations
  - By “peeking” and hashing

Hierarchical Quantization

- 8-times faster computation, loss of 0.001 in mAP



(a) (b) (c)

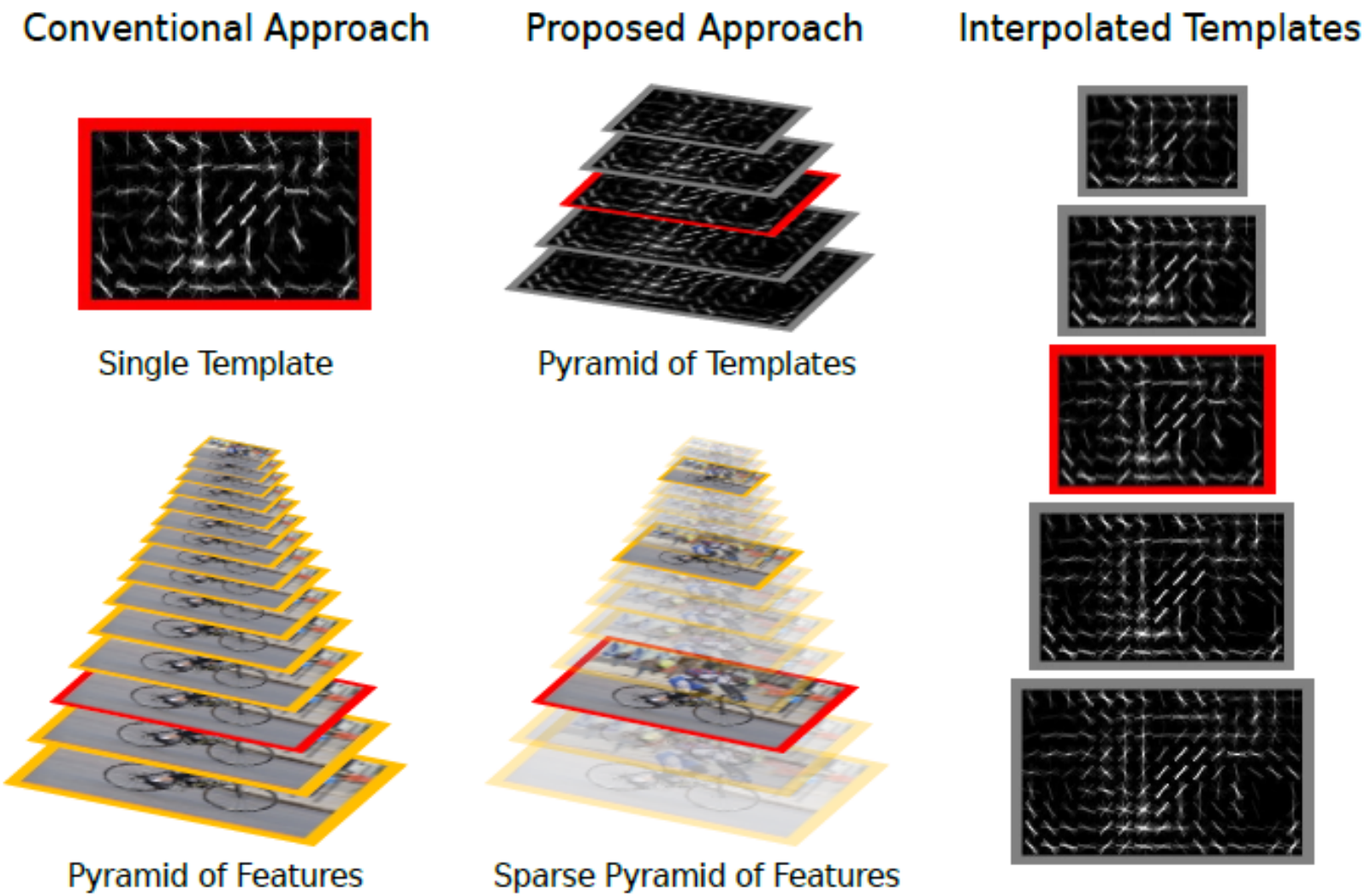
Original: NN to 256 centers

Current: NN to 16 centers, then NN again to 16 centers dep. on first

Strategies to speed up

- Accept some loss of numerical precision
  - quantize cells hierarchically
- Manage Scale carefully
  - More templates, fewer HOG features
- Prioritize
  - Avoid evaluating templates at all locations
  - By “peeking” and hashing

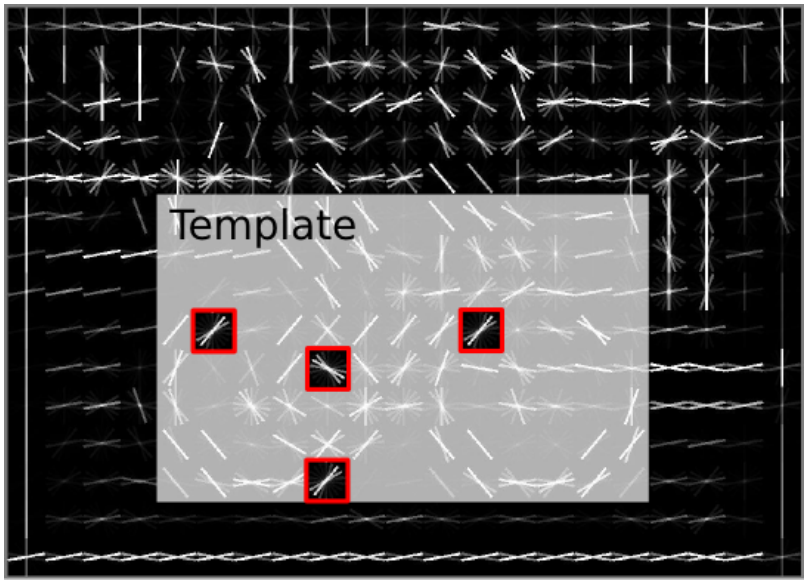
Pyramid of Templates



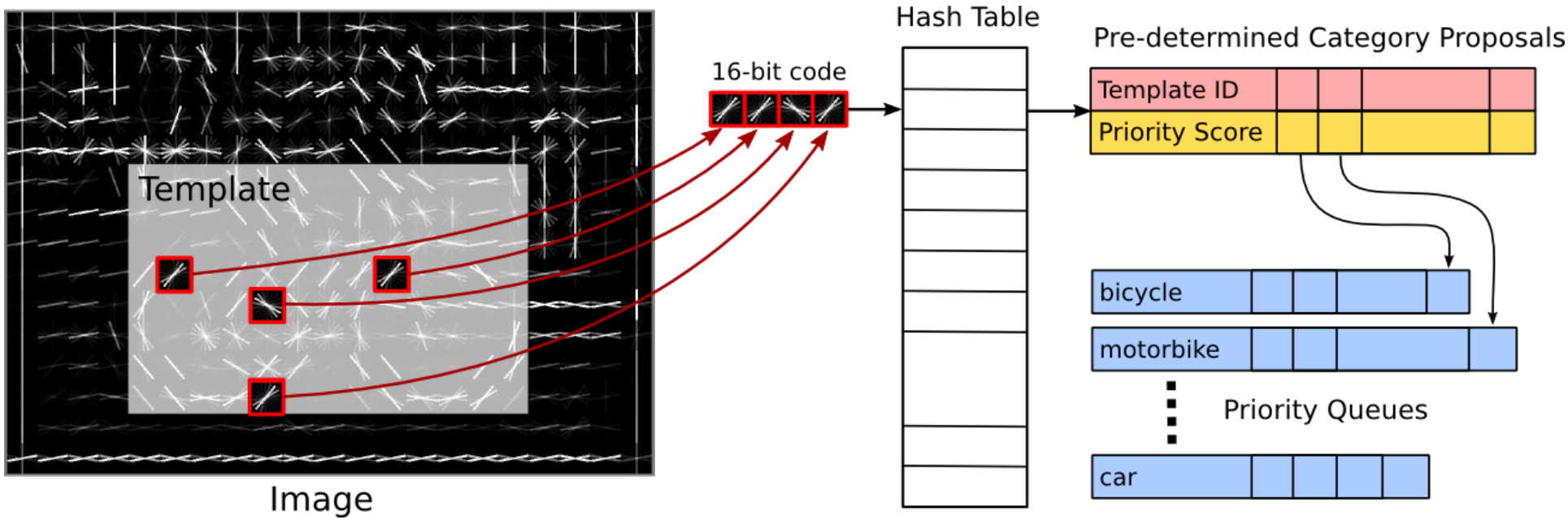
Strategies to speed up

- Accept some loss of numerical precision
  - quantize cells hierarchically
- Manage Scale carefully
  - More templates, fewer HOG features
- Prioritize
  - Avoid evaluating templates at all locations
  - By “peeking” and hashing

“Peeking” to speed up template evaluation

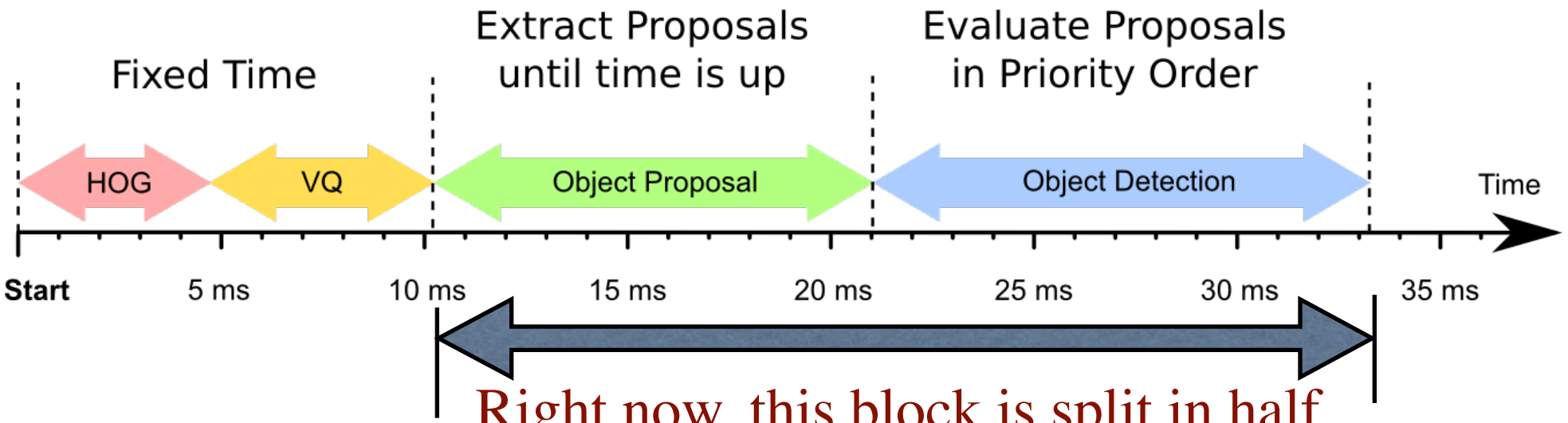


Object Proposal



Any-time Property

- Prioritize tasks
- Halt when time is up



Right now, this block is split in half,
but this might not be optimal?

Speed up vs Accuracy

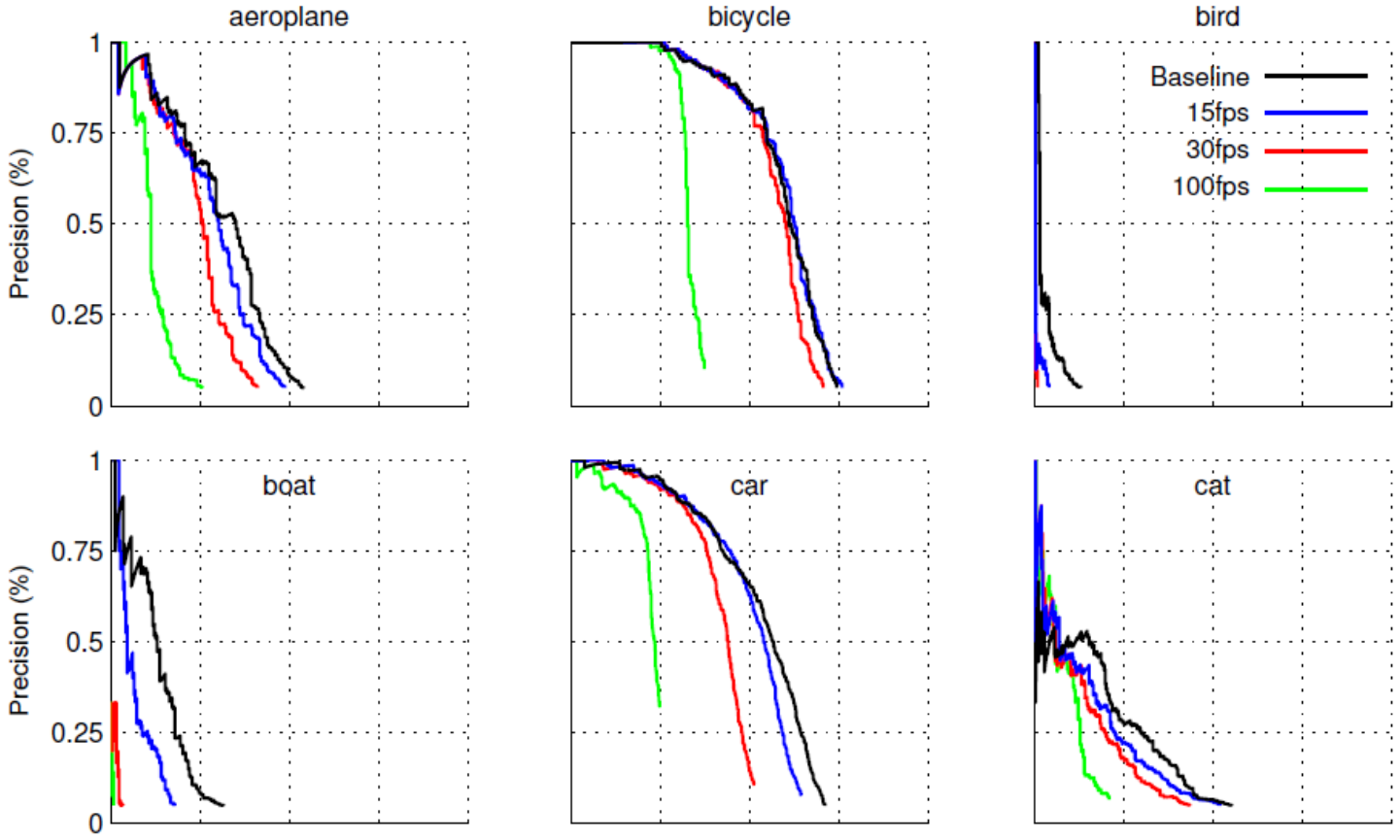
Algorithm	mAP	Time	Authors
Original DPM	0.33	13.3	Felzenszwalb et al. ‘10
DPM Cascade	0.331	1.7	Felzenszwalb et al. ‘10
FFLD	0.323	1.8	Dubout and Fleuret ‘12
Sparse Kernel	0.277	7	Vedaldi and Zisserman ‘12
WTA	0.24	26	Felzenszwalb et al.
FTVQ	0.331	0.53	Sadeghi, Forsyth ‘13
Ours	0.261	0.03	Sadeghi, Forsyth ‘14

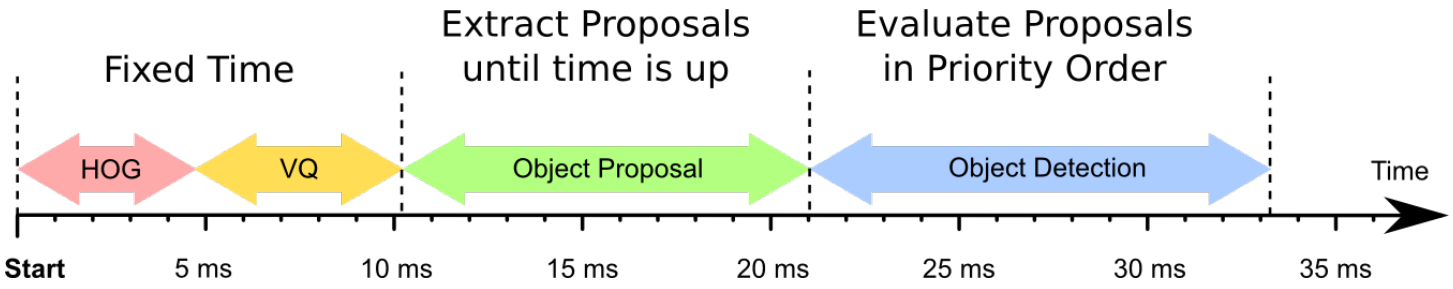
Time = time to complete the detection of all 20 categories starting at raw image

PASCAL’07 Object detection

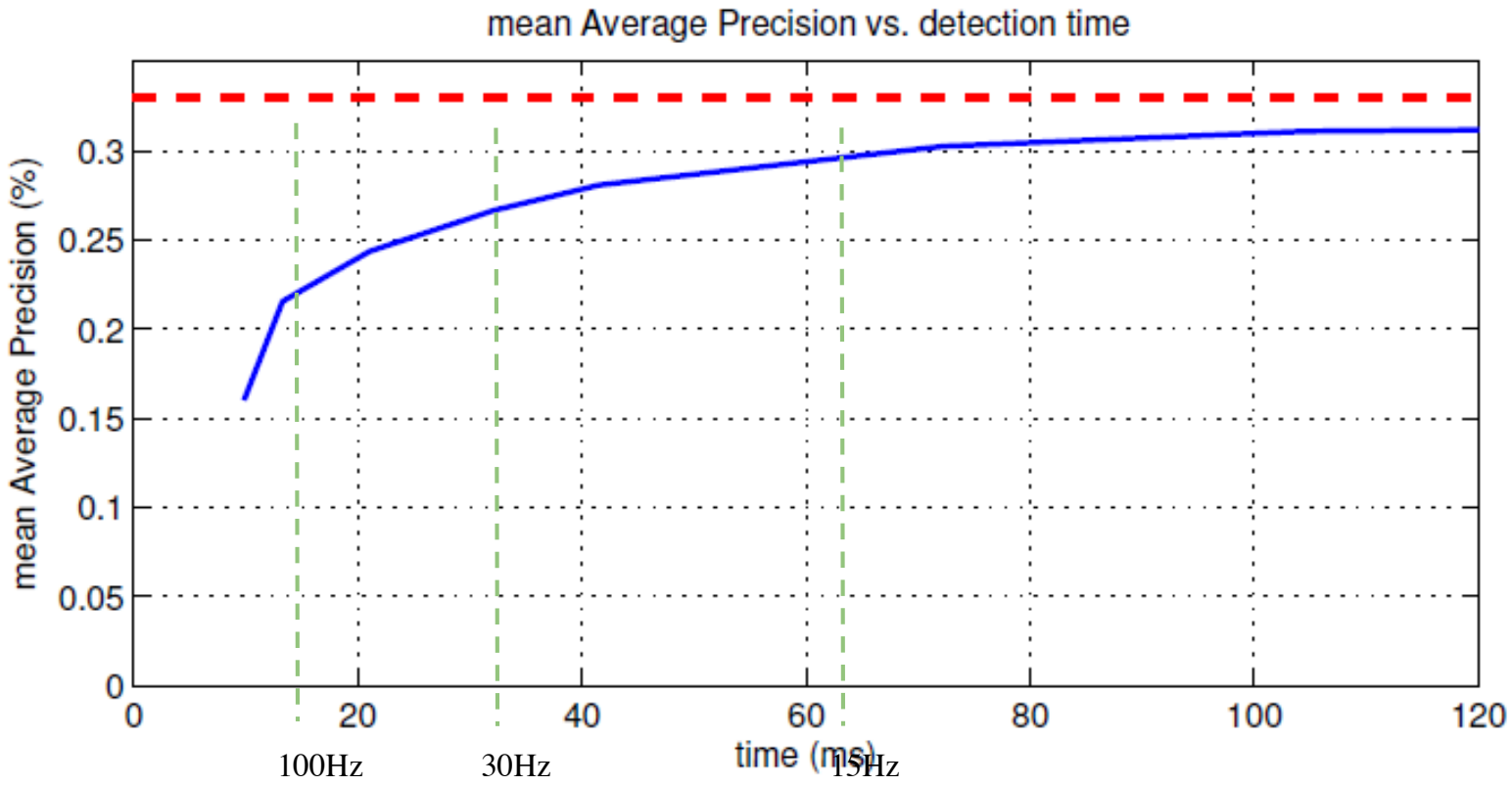
Intel Xeon E5-1650 Processor (6-cores)

Typical PR curves



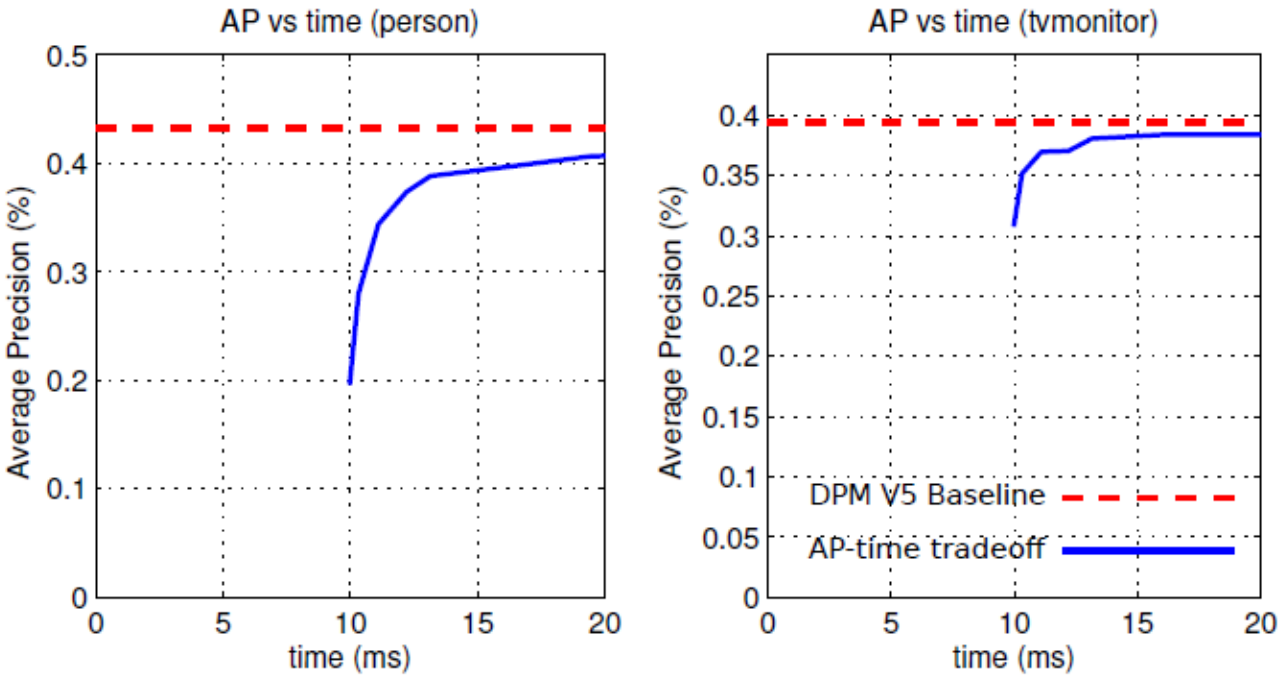


Results



Time = time to complete the detection of all 20 categories starting at raw image

AP against time



Desirable Features of any Approach

- Can work with legacy templates
  - No retraining required
- Random access to image
  - otherwise we're locked out of pruning/cascade strategies
- Can trade accuracy vs. speed
  - so application developers can look for a sweet spot
- Anytime property
  - so that plausible/tolerable results are returned whenever interrupted

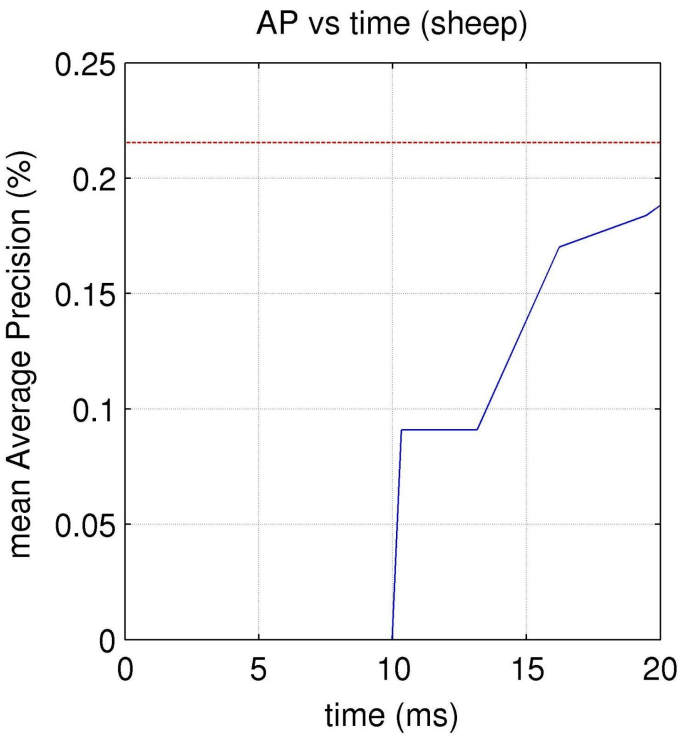
Desirable Features

Year	Algorithm	mAP	Time (s)	Legacy Templates	Random Access	Trade Accuracy vs Speed	Anytime Property
2006	Original DPM	0.33	13.3	Yes	No	No	No
2010	DPM Cascade	0.331	1.7	Yes	Yes	Yes	No
2012	FFLD	0.323	1.8	Yes	Impossible	Impossible	Impossible
2012	Sparse Kernel	0.277	7	Impossible	No	No	No
2013	WTA	0.24	26	Yes	Yes	Yes	No
2013	FTVQ	0.331	0.53	Yes	Yes	Yes	No
2014	Ours	0.261	0.03	Yes	Yes	Yes	Yes

Future Work

- More systematic resource allocation
  - Autotune:
    - hashing
    - priority values
    - internal parameters
    - to get best behavior for fixed set of templates
- Lazy HOG evaluation
- Apply similar speedups to Conv-net

Something funny going on here!



- What should we do with many templates?

Final comments

- It's fast, for quite simple reasons
  - Code URL on poster
- All fast codes should:
  - admit legacy templates
  - allow random access
  - allow speed/accuracy tradeoffs
  - be anytime
- As well as fast detection, our code is likely good for:
  - proposal algorithms
  - high precision regimes
  - Mobile-applications