Object Detection
Sliding Windows

Type of Approaches

Different approaches tackle detection differently. They can roughly be categorized into three main types:

- Find **interest points**, followed by Hough voting
- **Sliding windows**: "slide" a box around image and classify each image crop inside a box (contains object or not?) $\leftarrow$ **Let's look at a few methods for this**
- Generate **region (object) proposals**, and classify each region

Sliding Window Approaches

There are many... We will look at two in more detail:

- Dalal and Triggs (2005): HOG (Person) Detector (9,541 citations)
- Felzenswalb et al. (2010): Deformable Part-based Model (2,333 citations)

The last detector (DPM) is an extension of Dalal & Triggs. If we have time we'll also talk about the following approach (if not, I suggest you read it since it has some fantastic ideas):

- Viola and Jones (2001): (Face) Detector (10,043 citations)

Sliding Window Approaches

There are many... We will look at three in more detail:

- Dalal and Triggs (2005): HOG (Person) Detector $\rightarrow$ This first
- Felzenswalb et al. (2010): Deformable Part-based Model

The HOG Detector

N. Dalal and B. Triggs

Histograms of oriented gradients for human detection

CVPR, 2005

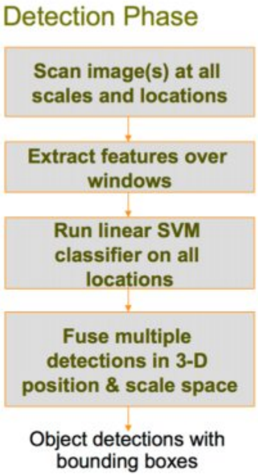
Paper: `http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf`

The HOG Detector

- We want to find all people in this image. Preferably our detections should not include trees, lamp posts and umbrellas.

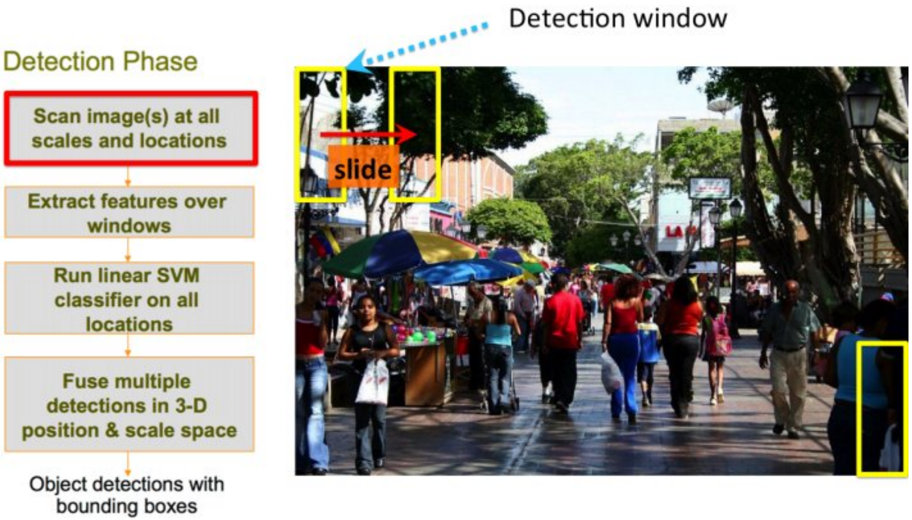
The HOG Detector

- Sliding window detectors find objects in 4 very simple steps: **(1.)** inspect every window, **(2.)** extract features in window, **(3.)** classify & accept wind. if score above threshold, **(4.)** clean-up the mess (called post-processing)



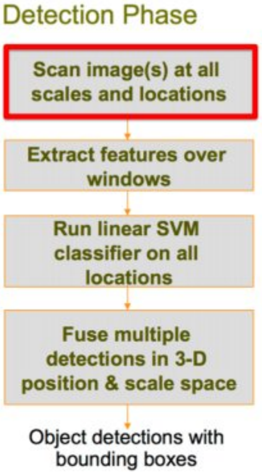
The HOG Detector – Sliding the Window

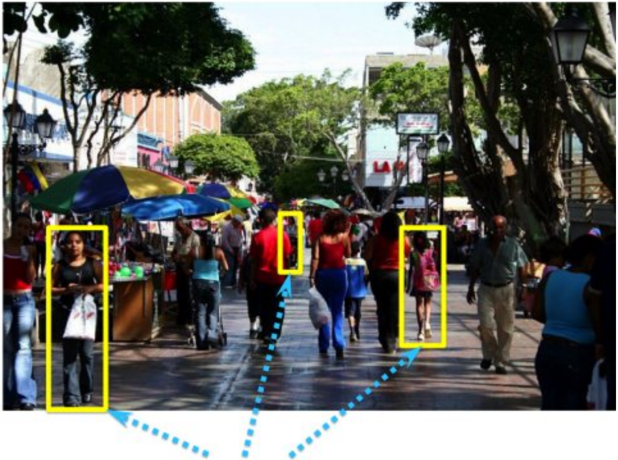
- First step: inspect every window. Typically the size of window is **fixed**.



The HOG Detector – Sliding the Window

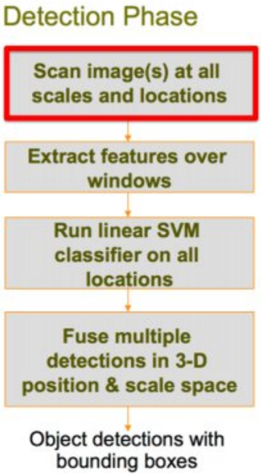
- Since window size is fixed, how can we find people at different sizes?





Objects can be of very different sizes (scales), even in the same image. How do we deal with that?

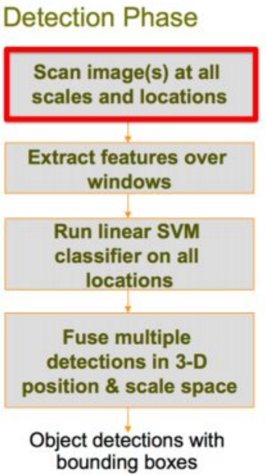
- Shrink (down-scale) the image and slide again



Scale-down the image, and slide the window again (the size of the window is always the same)

The HOG Detector – Sliding the Window

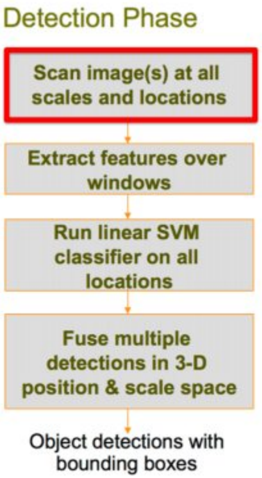
- Keep shrinking and sliding

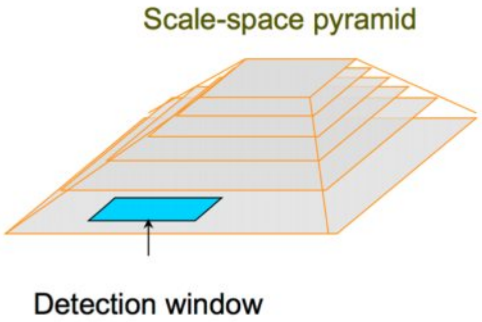


And again...

The HOG Detector – Sliding the Window

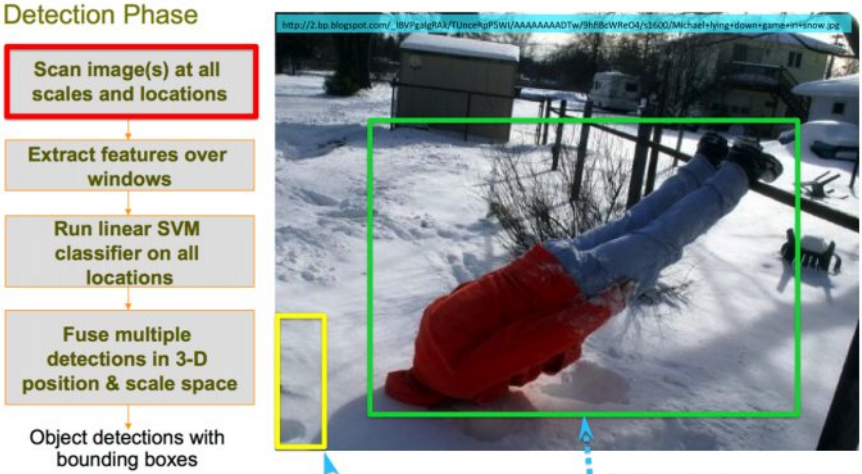
- In fact, do a full image pyramid, and slide your detector at each scale. Make sure the scale differences across levels are small (do lots of re-scaled images)





The HOG Detector – Sliding the Window?

- What if the object is in a weird pose (window is of different aspect ratio)?



How can we deal with this guy?

Our window size

The HOG Detector – Limitations

- Stop thinking too hard. In 2005 people were only in upright position.
- We will re-visit this question a little later (when we talk about DPM)

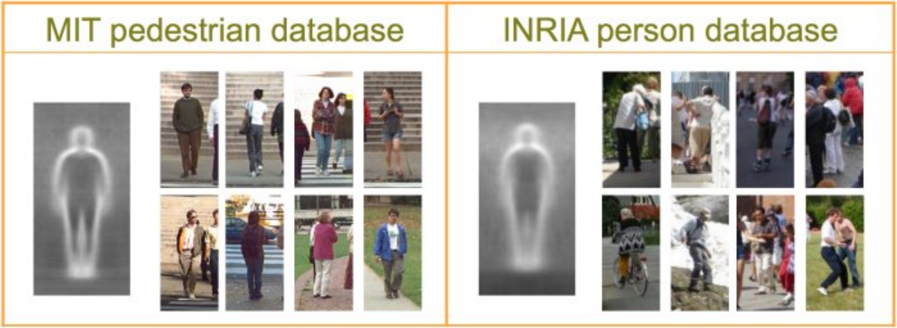
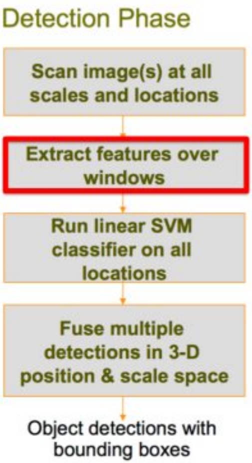
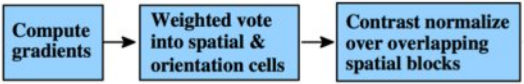


Figure: Main pedestrian detection datasets prior to PASCAL VOC.

The HOG Detector – Features (HOG)

- Famous feature descriptor called HOG that replaced SIFT (at least for object detection). There are three steps to compute it.



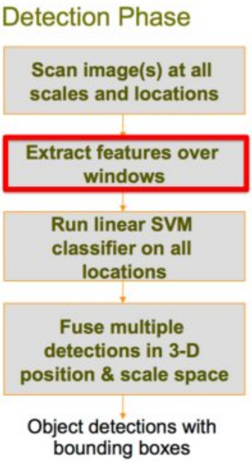


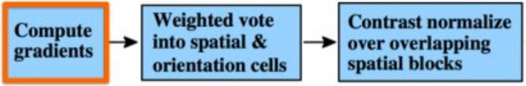
Features:

- Called: **Histograms of Gradients (HOG)**
- Three steps to compute them
- Quite similar to SIFT

The HOG Detector – Features (HOG)

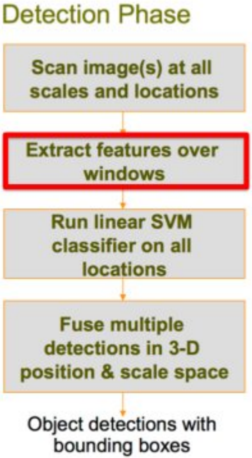
- First compute gradients

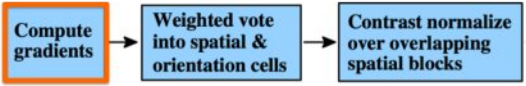




The HOG Detector – Features (HOG)

- There are many ways how to compute the gradients. The HOG detector guys tried a lot of them and picked the best one.





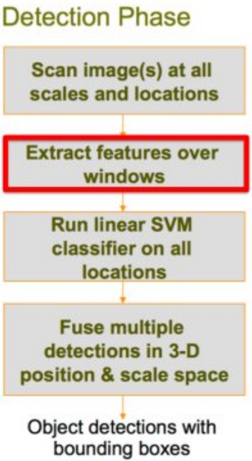
Mask Type	1D centered	1D uncentered	1D cubic-corrected	2x2 diagonal	3x3 Sobel
Operator	[-1, 0, 1]	[-1, 1]	[1, -8, 0, 8, -1]	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Miss rate at 10^{-4} FPPW	11%	12.5%	12%	12.5%	14%

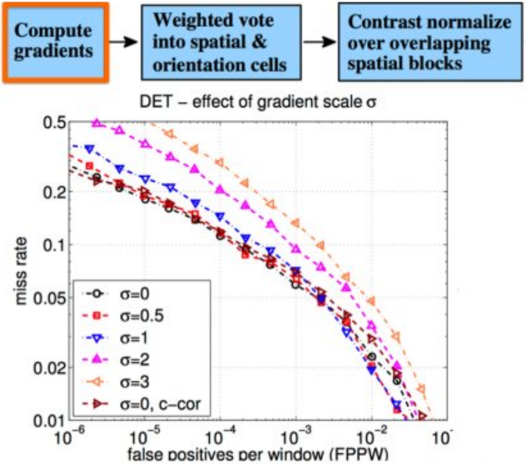
(Miss rate: smaller is better)

This gradient filter gives the best performance

The HOG Detector – Features (HOG)

- One can also smooth image before computing the gradients. The HOG detector guys tested that as well. This is **great** science, **analyze every step**!



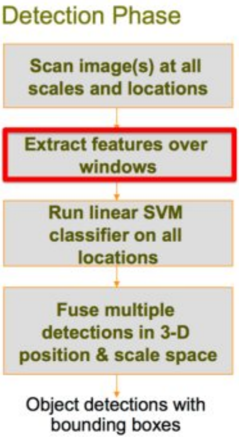


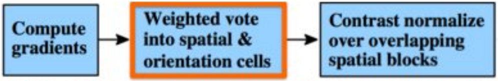
(Miss rate: smaller is better)

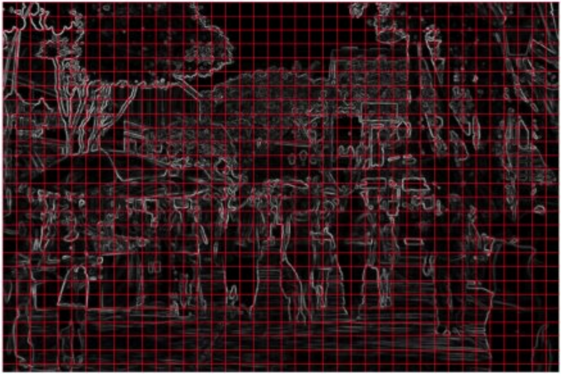
No Gaussian smoothing gives the best performance

The HOG Detector – Features (HOG)

- Divide the image into **cells** of 8×8 pixels.



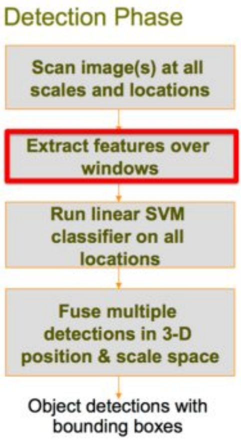


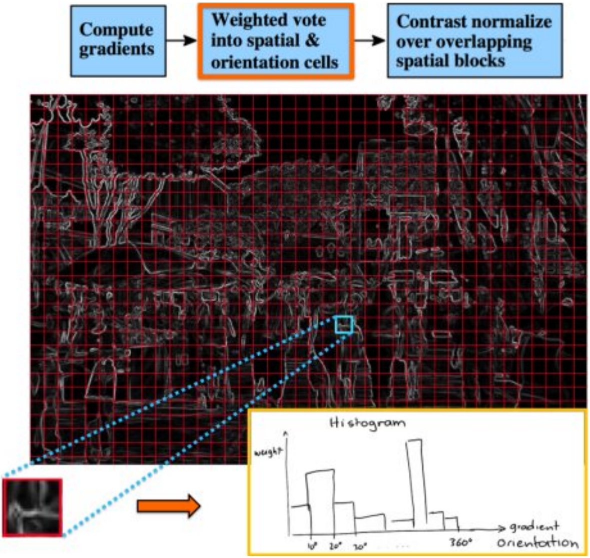


Divide the gradient image into non-overlapping **cells**. Each cell is typically 8x8 pixels.

The HOG Detector – Features (HOG)

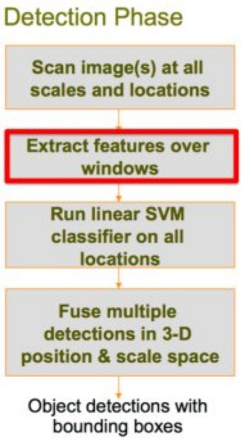
- Compute a histogram of orientations in each cell (similar to SIFT)

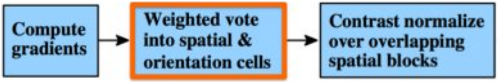


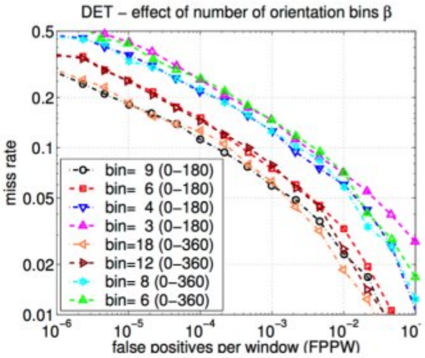


The HOG Detector – Features (HOG)

- Again, check how many bins is best to use. Turns out: 9 with orient 0-180.





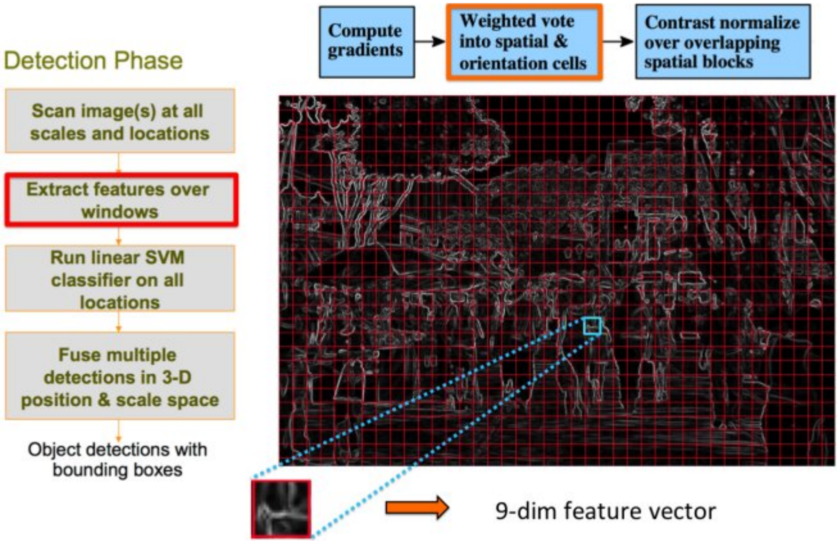


(Miss rate: smaller is better)

9 bins (unsigned orient) is best

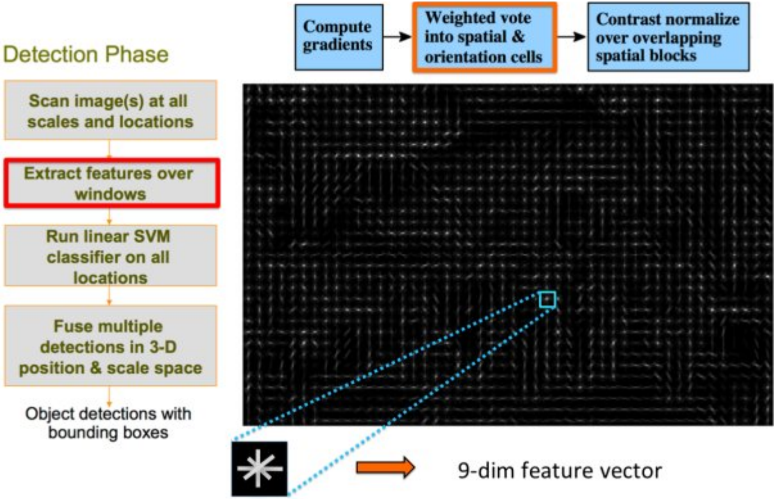
The HOG Detector – Features (HOG)

- So each cell now has a 9-dimensional feature vector



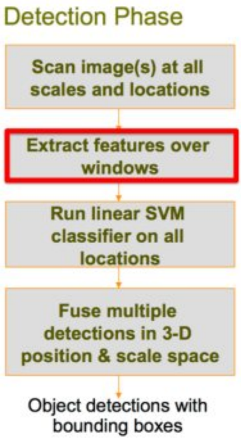
The HOG Detector – Features (HOG)

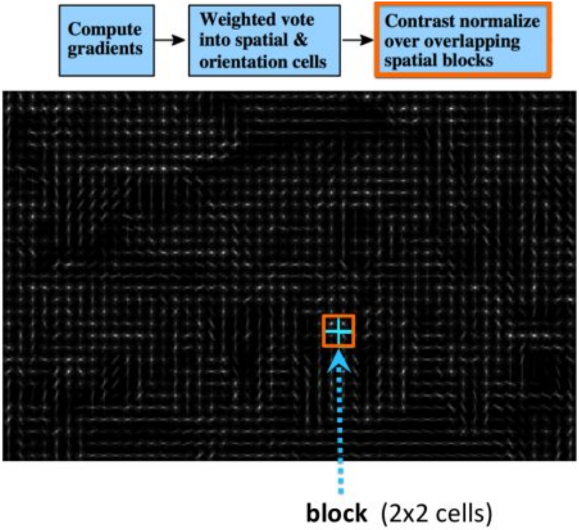
- In literature you will see this kind of **visualization** for HOG. In each cell people plot all the orientations that are present in the cell. Do not confuse this visualization with the actual feature (composed of 9 matrices).



The HOG Detector – Features (HOG)

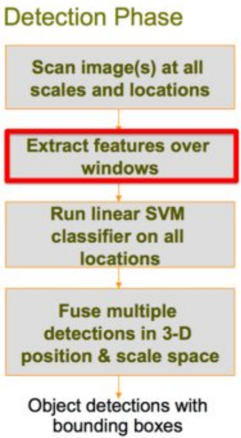
- We're not finished. We now take **blocks**, where each block has 2×2 cells.

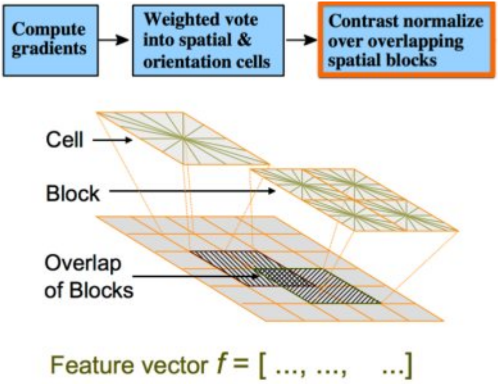




The HOG Detector – Features (HOG)

- We normalize each feature vector, such that each block has unit norm. This step doesn't change the dimension of the feature, just the strength. Why are we doing this?

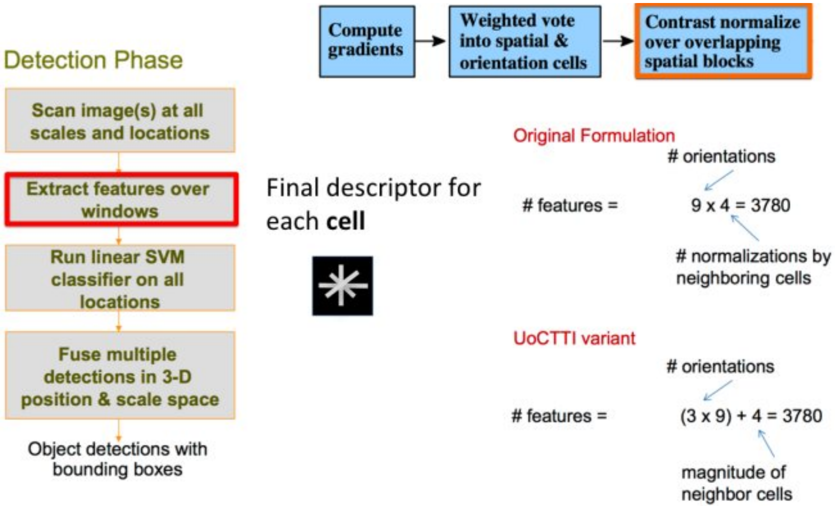




L2 normalization in each block: $$\mathbf{f} = \frac{\mathbf{f}}{\sqrt{||\mathbf{f}||_2^2 + \epsilon^2}}$$

The HOG Detector – Features (HOG)

- Since each cell is in 4 blocks, we have 4 different normalizations, and we make each one into separate features.



The HOG Detector – Features (HOG)

- For person class, window is 15×7 HOG cells (what's the size in pixels?)
- We vectorize each the feature matrix in each window.

Compute gradients → **Weighted vote into spatial & orientation cells** → **Contrast normalize over overlapping spatial blocks**

Detection Phase

Scan image(s) at all scales and locations
↓
Extract features over windows
↓
Run linear SVM classifier on all locations
↓
Fuse multiple detections in 3-D position & scale space
↓
Object detections with bounding boxes

Final descriptor for window (person class in this case)

Original Formulation

orientations

features = 15 x 7 x 9 x 4 = 3780

cells

normalizations by neighboring cells

UoCTTI variant

orientations

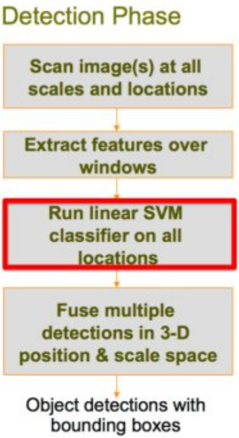
features = 15 x 7 x (3 x 9) + 4 = 3780

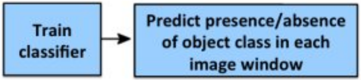
cells

magnitude of neighbor cells

The HOG Detector – Classification

- Features done, we are ready for classification. We first need to **train** our classifier, and only after we can do detection (prediction).



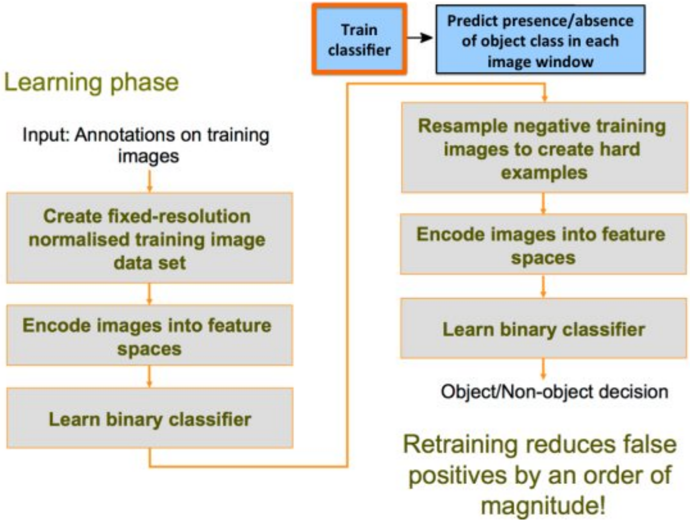


Detection:

- **Train** a window classifier
- Use the trained classifier to **predict** presence/absence of object class in each window in the image

The HOG Detector – Training

- Several simple steps. Plus a few useful additional tricks (remember, hacking is part of the Secret Life of a Vision Researcher).



The HOG Detector – Training

- Take a dataset with annotations. If nothing exists, collect and label yourself.

Train classifier → **Predict presence/absence of object class in each image window**

Learning phase

Input: Annotations on training images

↓

Create fixed-resolution normalised training image data set

↓

Encode images into feature spaces

↓

Learn binary classifier

positive training examples

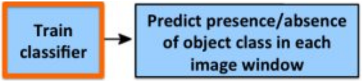
negative training examples

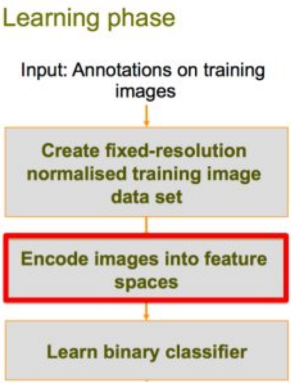
- All image crops **are scaled to the same size** (for this example (15x8) x (7x8) pixels), where 8 is the width/height of each HOG cell in pixels
- **Cool trick**: take a bigger region than each annotated object to also capture **context** (works better!)

Pics: S. Lazebnik

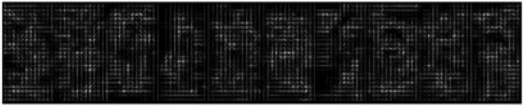
The HOG Detector – Training

- Scale positive and negative examples to the size of detection window. Compute HOG.

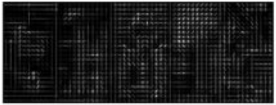




positive training examples



negative training examples

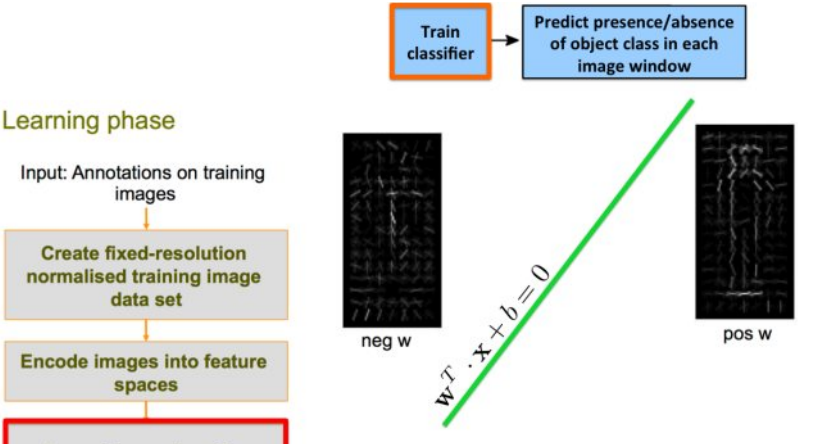


*** These are just feature visualizations. Each \`\`picture'' is really a 15x7x31 feature matrix.

Before training a classifier, we vectorize each of these examples: f=f(:)

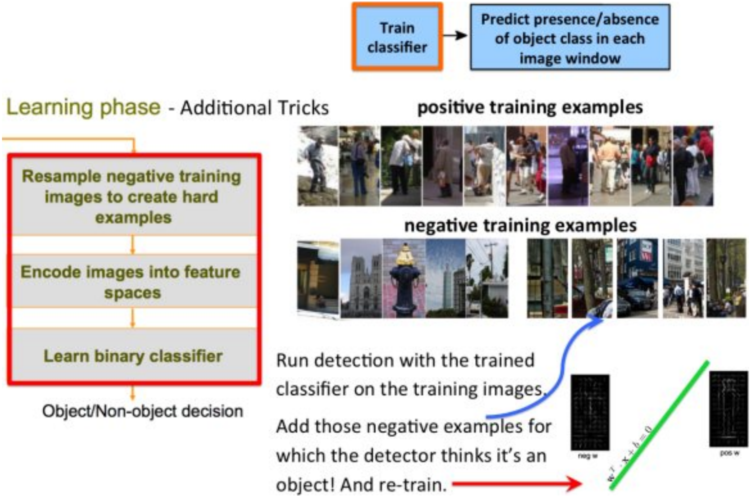
The HOG Detector – Training

- Train a classifier (with e.g. LibSVM).



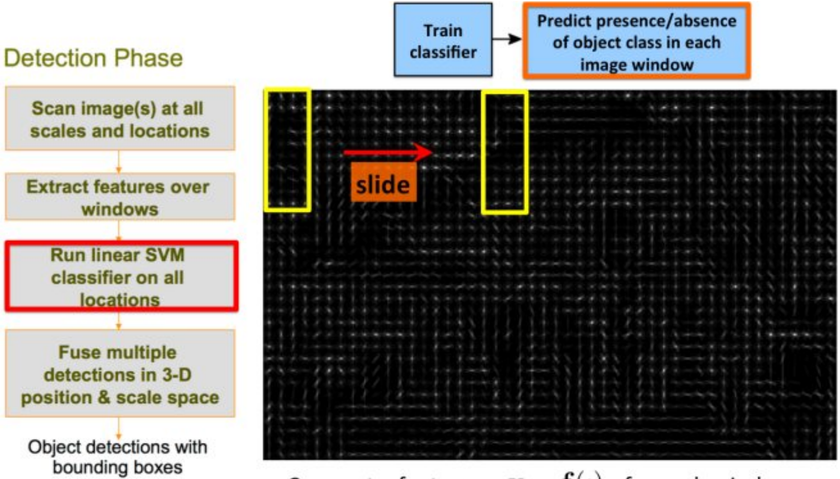
Train classifier. SVM (Support Vector Machines) is typically used.

- Additional tricks: **Bootstrapping**. A fancy name for running your classifier on **training** images (with full detection pipeline), and finding mis-classified windows. Add those to training examples, and re-train classifier.



The HOG Detector – Detection

- Take a window, crop out a feature matrix, vectorize and classify

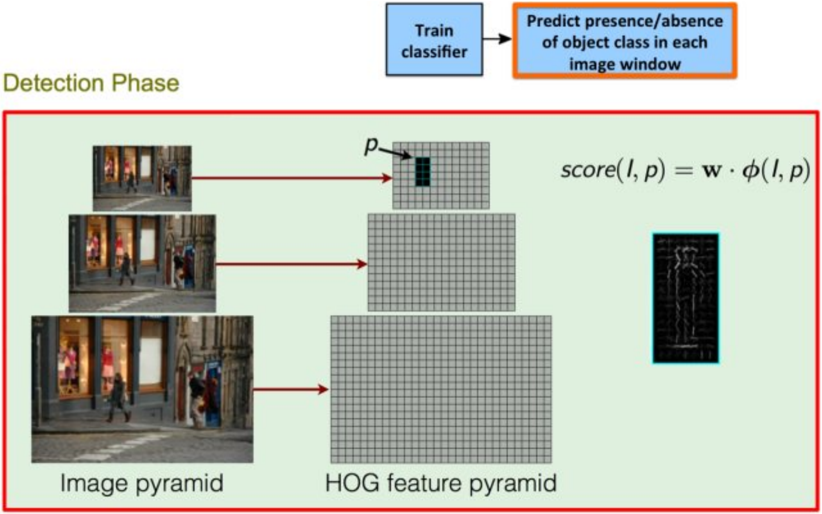


Crop out a feature $\mathbf{x} = \mathbf{f}(:)$ for each window

Compute: $\text{score} = \mathbf{w}^T \cdot \mathbf{x} + b$ (higher better)

The HOG Detector – Detection

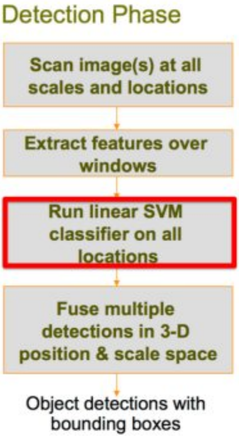
- Computing the score $\mathbf{w}^T \cdot \mathbf{x} + b$ in every location is the same as performing **cross-correlation with template w** (and add b to result).



[Pic from: R. Girshik]

The HOG Detector – Training

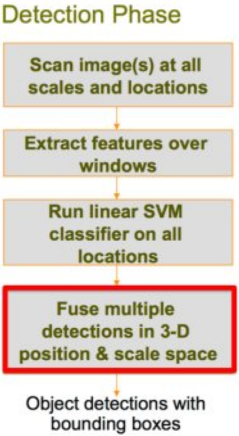
- Threshold the scores (e.g., $\text{score} > -1$)



- Run detector on all scales (image sizes)
- Find scores (and thus boxes) higher than threshold
- You get a soup of overlapping boxes. What can you do to get rid of multiple detections of the same object?

The HOG Detector – Post-processing

- Perform Non-Maxima Supression (NMS)

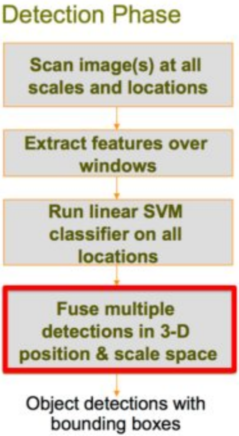


Non-maxima suppression (NMS)

- Greedy algorithm.
- At each iteration pick the highest scoring box.

The HOG Detector – Post-processing

- Perform Non-Maxima Supression (NMS)



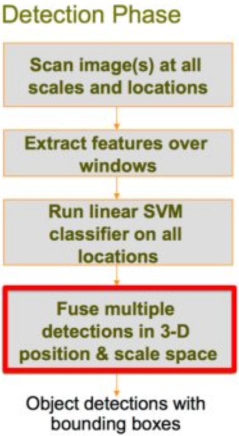
Non-maxima suppression (NMS)

$$\text{overlap} = \frac{\text{area}(box_1 \cup box_2)}{\text{area}(box_1 \cap box_2)} > 0.5 \Rightarrow \boxed{\text{remove } box_2}$$

- Remove all boxes that overlap more than XX (typically 50%) with the chosen box

The HOG Detector – Post-processing

- Perform Non-Maxima Supression (NMS)

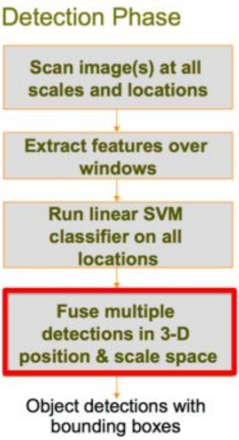


Non-maxima suppression (NMS)

- Greedy algorithm.
- At each iteration pick the highest scoring box.
- Remove all boxes that overlap more than XX (typically 50%) with the chosen box

The HOG Detector – Post-processing

- Perform Non-Maxima Supression (NMS)

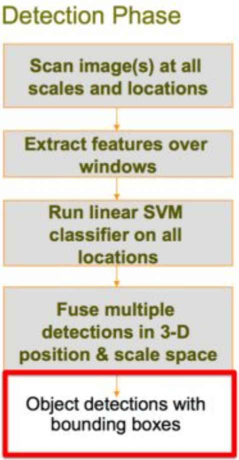


Non-maxima suppression (NMS)

- Greedy algorithm.
- At each iteration pick the highest scoring box.
- Remove all boxes that overlap more than XX (typically 50%) with the chosen box

The HOG Detector – Post-processing

- Done!



Voila!

(Any idea how you would get rid of that tree detection on the upper right?)

Results

- Some results

How Should We Evaluate Object Detection Approaches?

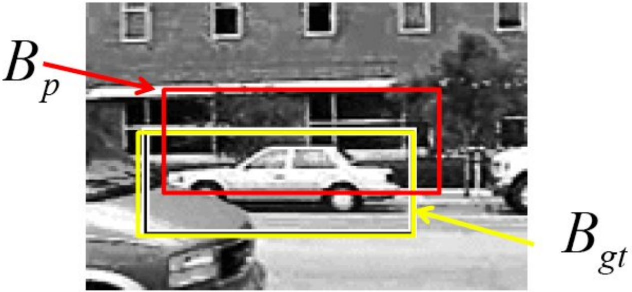
- How can we tell if our approach is doing well?
- What should be our evaluation?

What's a Correct Detection

Evaluation criteria:

- Detection is correct if the intersection of the bounding boxes, divided by their union, is $> 50\%$.

$$a_0 = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})}$$



[Source: K. Grauman, slide credit: R. Urtasun]

Multiple Detections are Considered Wrong

- Below both detections have more than 50% overlap with ground-truth annotation. But only **one** will count as correct, the other(s) will count as **false positive** (wrong).

Precision and Recall

- We sort all the predicted boxes (for all images) according to scores, in descending order
- Then for each k (location) in the list we can compute precision and recall obtained when using top k boxes in the list

Precision and Recall

- We sort all the predicted boxes (for all images) according to scores, in descending order
- Then for each k (location) in the list we can compute precision and recall obtained when using top k boxes in the list
- Recall:

$$\text{recall} = \frac{\#\text{correct boxes}}{\#\text{ground-truth boxes}}$$

- Precision:

$$\text{precision} = \frac{\#\text{correct boxes}}{\#\text{all predicted boxes}}$$

- What's the min/max value of recall/precision?

Precision and Recall

- We sort all the predicted boxes (for all images) according to scores, in descending order
- Then for each k (location) in the list we can compute precision and recall obtained when using top k boxes in the list
- Recall:

$$\text{recall} = \frac{\#\text{correct boxes}}{\#\text{ground-truth boxes}}$$

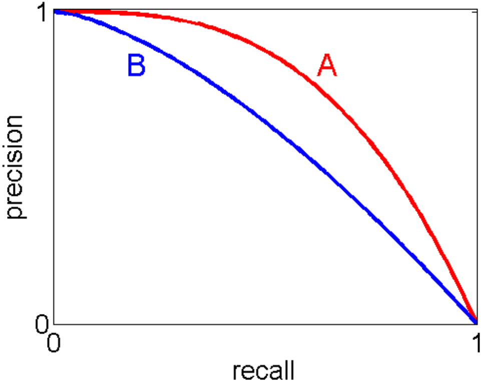
- Precision:

$$\text{precision} = \frac{\#\text{correct boxes}}{\#\text{all predicted boxes}}$$

- What's the min/max value of recall/precision?

Precision and Recall Curve

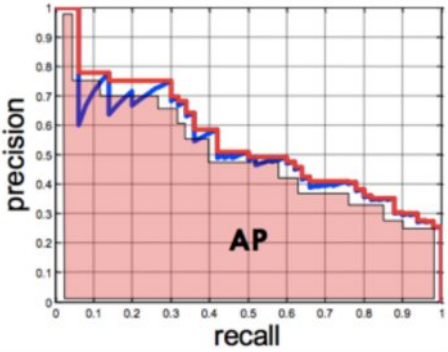
- Then you can plot a precision-recall curve
- Which curve in the plot below is better, A or B?



[Pic: http://pmtk3.googlecode.com/svn-history/r785/trunk/docs/demos/Decision_theory/PRhand_01.png]

Average Precision

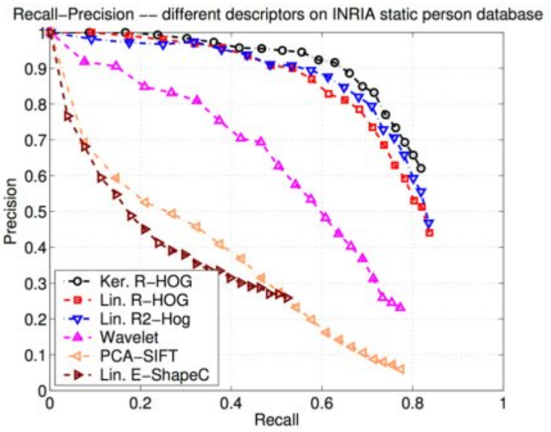
- Average Precision (AP): Compute the area under the precision-recall curve
- What's the best AP one can get? What's the worst?
- AP is the standard measure for evaluating object detection performance
- Sometimes you may encounter notation mAP. This is mean Average Precision, and it's just an average of APs across different classes.



[Pic from: R. Girshik]

Performance of the HOG Detector (back in 2005)

- PR curve for the HOG detector
- Interesting: Look at the curve for PCA-SIFT (improved SIFT). Way down there...



[Pic from: R. Girshik]