The DPM Detector

P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan

Object Detection with Discriminatively Trained Part Based Models

T-PAMI, 2010

Paper: `http://cs.brown.edu/~pff/papers/lsvm-pami.pdf`
Code: `http://www.cs.berkeley.edu/~rbg/latent/`

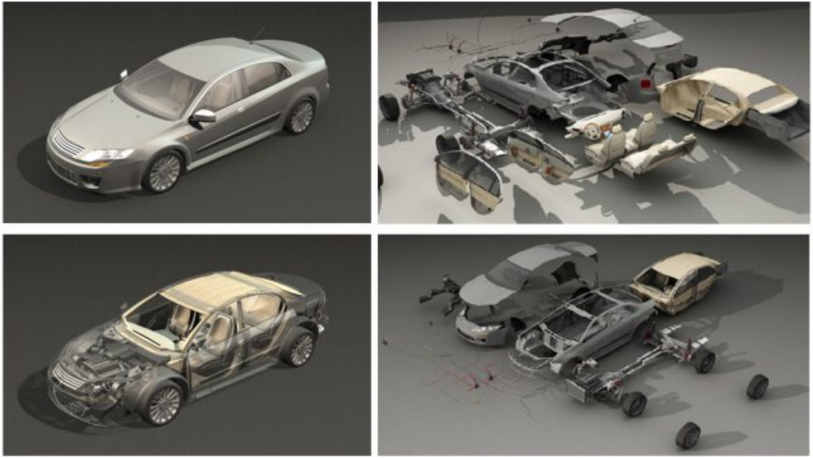
The HOG Detector

- The HOG detector models an object class as a single rigid template

Figure: Single HOG template models people in upright pose.

But Objects Are Composed of Parts

Even Rigid Objects Are Composed of Parts



Objects Are Composed of Deformable Parts

- Revisit the old idea by Fischler & Elschlager 1973
- Objects are composed of parts at specific **relative locations**. Our model should probably also model object parts.
- Different instances of the same object class have parts in slightly different locations. Our object model should thus allow slight **slack** in part position.

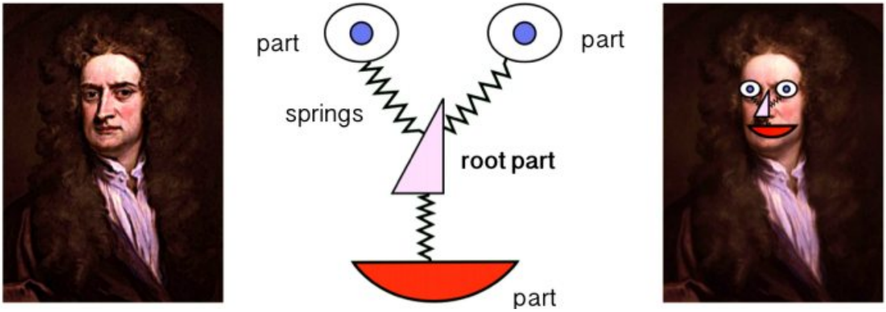
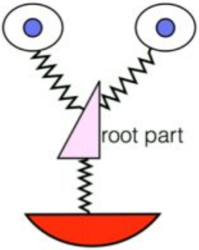


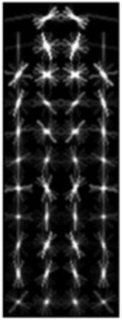
Figure: Objects are a collection of deformable parts

[Pic from: R. Girshik]

The DPM Model

- The DPM model starts by borrowing the idea of the HOG detector. It takes a HOG template for the full object. (If you take something that works, things can only get better, right?)



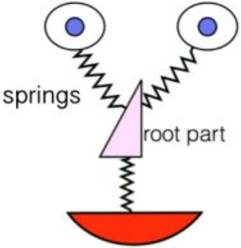


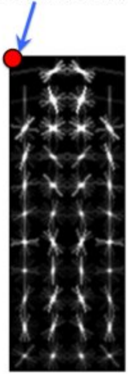
root part (or root filter)

The DPM Model

- DPM now wants to add parts. It wants to add them at locations **relative** to the location of the root filter. Relative makes sense: if we move, we take our parts with us.

We add parts at locations relative to this point
(upper left corner of the root filter)

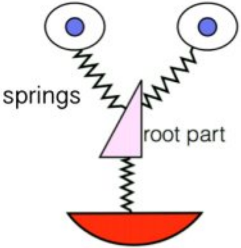


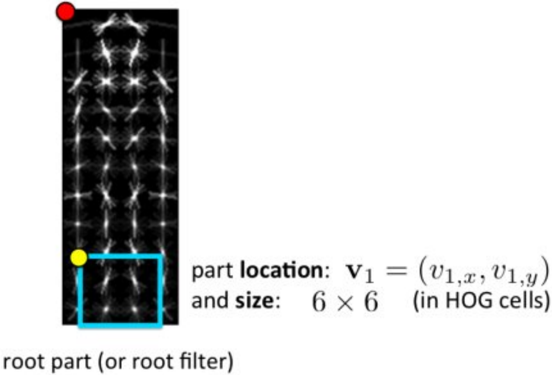


root part (or root filter)

The DPM Model

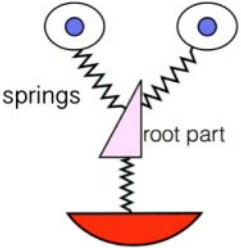
- Add a part at a relative location and scale.

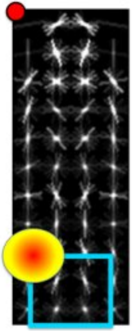




The DPM Model

- Give some slack to the location of the part. Why is this a good idea?



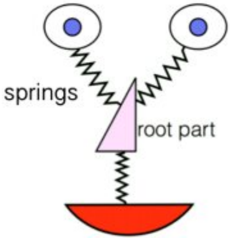


A part also has deformation: it can slightly ``move'' around expected location
This deformation is modeled with a quadratic function

root part (or root filter)

The DPM Model

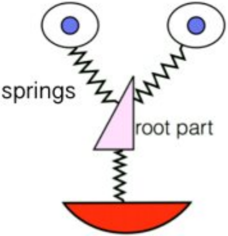
- People are of different heights, thus have feet at different locations relative to the head. And we want to detect all people, not just the average ones.



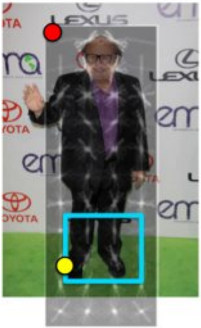
Lebron James: Too big for the box

The DPM Model

- People are of different heights, thus have feet at different locations relative to the head. And we want to detect all people, not just the average ones.

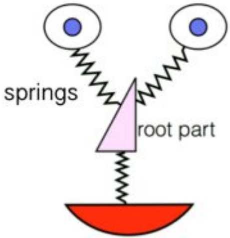


Danny de Vito: Too small for the box



The DPM Model

- People are of different heights, thus have feet at different locations relative to the head. And we want to detect all people, not just the average ones.

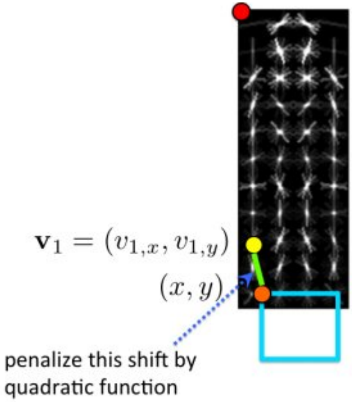


Brad Pitt: Fits perfectly

The DPM Model

- We will, however, trust less detections where parts are not exactly in their expected location. DPM penalizes part shifts with a quadratic function:

$$a(x - v_x)^2 + b(x - v_x) + c(y - v_y)^2 + d(y - v_y)$$

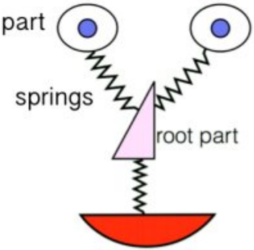


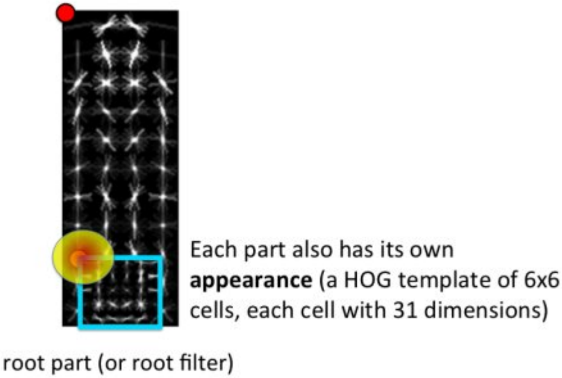
For example, a very tall person may have feet way lower. We want our model to detect also tall people.

But since there are less really tall people, we want to penalize such detections a little bit (we will trust it less – how many images do actually have NBA players, afterall?).

The DPM Model

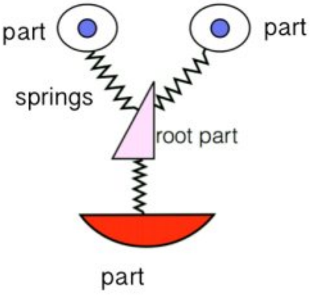
- Each part also has an **appearance**, which is modeled with a HOG template
- Each part's template is at **twice the resolution** as the root filter

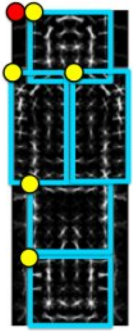




The DPM Model

- And finally, DPM has a few parts. Typically 6 (but it's a parameter you can play with). How many weights does a 6-part DPM model have?
- How shall we score this part-model guy in an image (how to do detection)?





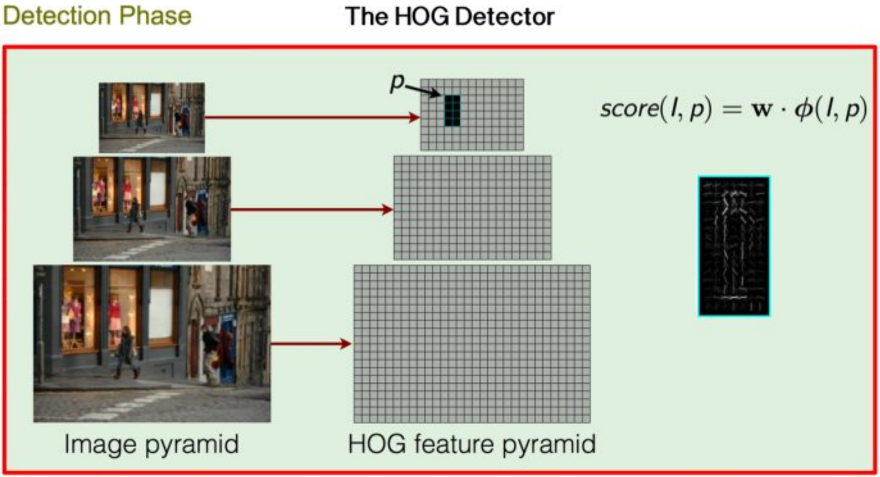
root part (or root filter)

Full model:

- Root filter (HOG template)
- Parts:
  - Location
  - Deformation
  - HOG template

Remember the HOG Detector

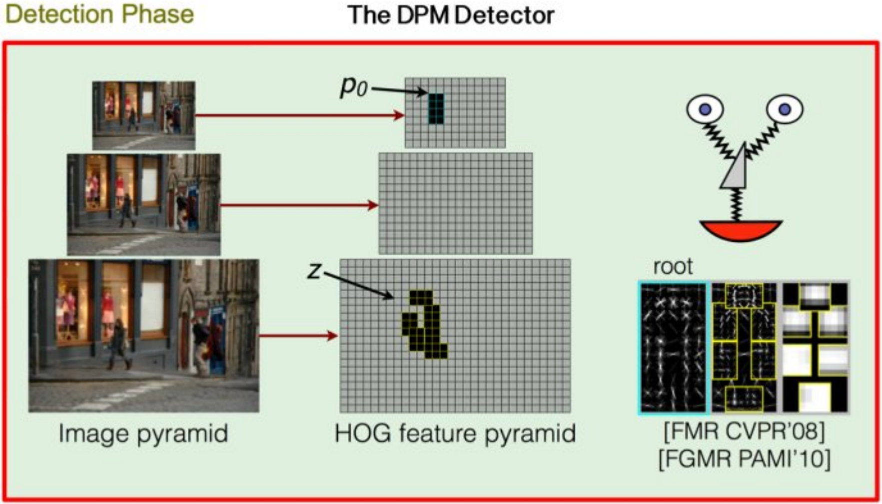
- HOG detector computes image pyramid, HOG features, and scores each window with a learned linear classifier



[Pic from: R. Girshik]

DPM Detector

- For DPM the story is quite similar (pyramid, HOG, score window with a learned linear classifier), but now we also need to score the parts.

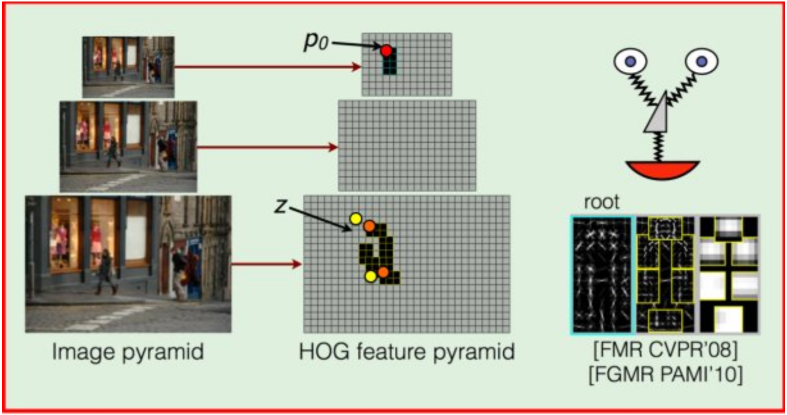


[Pic from: R. Girshik]

Scoring

$$z = (p_1, \ldots, p_n)$$

$$\text{score}(I, p_0) = \max_{p_1,\ldots,p_n} \underbrace{\sum_{i=0}^{n} m_i(I, p_i)}_{\text{Filter scores}} - \underbrace{\sum_{i=1}^{n} d_i(p_0, p_i)}_{\text{Spring costs}}$$



Scoring

- More specifically, we will score a location (window) in the image as follows:

$$\text{score}(l, p_0) = \max_{p_1,\ldots,p_n} \left(\sum_{i=0}^{n} F_i \cdot HOG(l, p_i) - \sum_{i=1}^{n} \mathbf{w_{def}}^i \cdot (dx, dy, dx^2, dy^2) \right)$$

where

- F_0 is the (learned) HOG template for root filter
- F_i is the (learned) HOG template for part i
- $HOG(l, p_i)$ means a HOG feature cropped in window defined by part location p_i at level l of the HOG pyramid
- $\mathbf{w_{def}}^i$ are (learned) weights for the deformation penalty
- (dx, dy, dx^2, dy^2) with $(dx, dy) = (x_i, y_i) - ((x_0, y_0) + \mathbf{v_i})$ tell us how far the part i is from its expected position $(x_0, y_0) + \mathbf{v_i}$

- **Main question:** How shall we compute that nasty $\max_{p_1,\ldots,p_n}$?

Scoring

- More specifically, we will score a location (window) in the image as follows:

$$\mathrm{score}(l, p_0) = \max_{p_1,\dots,p_n} \Big(\sum_{i=0}^{n} F_i \cdot HOG(l, p_i) - \sum_{i=1}^{n} \mathbf{w_{def}}^i \cdot (dx, dy, dx^2, dy^2) \Big)$$

  where

  - F_0 is the (learned) HOG template for root filter
  - F_i is the (learned) HOG template for part i
  - $HOG(l, p_i)$ means a HOG feature cropped in window defined by part location p_i at level l of the HOG pyramid
  - $\mathbf{w_{def}}^i$ are (learned) weights for the deformation penalty
  - (dx, dy, dx^2, dy^2) with $(dx, dy) = (x_i, y_i) - ((x_0, y_0) + \mathbf{v_i})$ tell us how far the part i is from its expected position $(x_0, y_0) + \mathbf{v_i})$

- **Main question:** How shall we compute that nasty $\max_{p_1,\dots,p_n}$?

Scoring

- Push the max inside (why can we do that?):

$$\mathrm{score}(I, p_0) = F_0 \cdot HOG(I, p_0) + \sum_{i=1}^{n} \max_{p_i} \left(F_i \cdot HOG(I, p_i) - \mathbf{w_{def}}^{i} \cdot \phi_{def}(x_i, y_i) \right)$$

Scoring

- Push the max inside:

$$\text{score}(I, p_0) = F_0 \cdot HOG(I, p_0) + \sum_{i=1}^{n} \max_{p_i} \Big(F_i \cdot HOG(I, p_i) - \mathbf{w_{def}}^{i} \cdot \phi_{def}(x_i, y_i) \Big)$$

- We can compute this with **dynamic programming**. Any idea how?

Computing the Score with Dynamic Programming

$$\text{score}(l, p_0) = F_0 \cdot HOG(l, p_0) + \sum_{i=1}^{n} \max_{p_i} \left(F_i \cdot HOG(l, p_i) - \mathbf{w_{def}}^i \cdot \phi_{def}(x_i, y_i) \right)$$

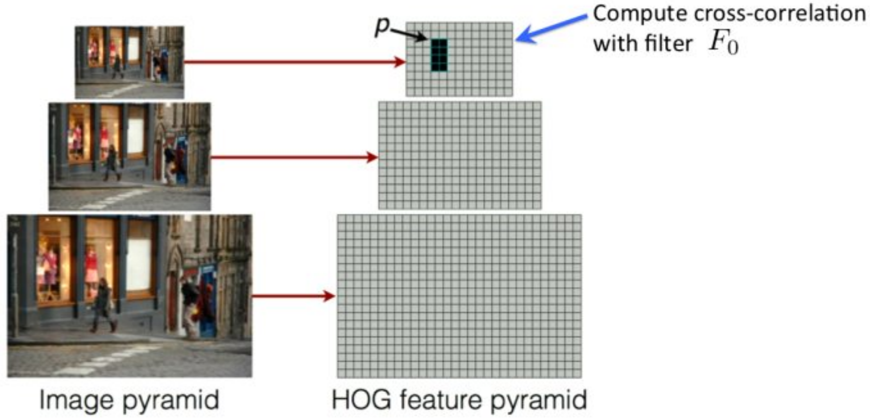
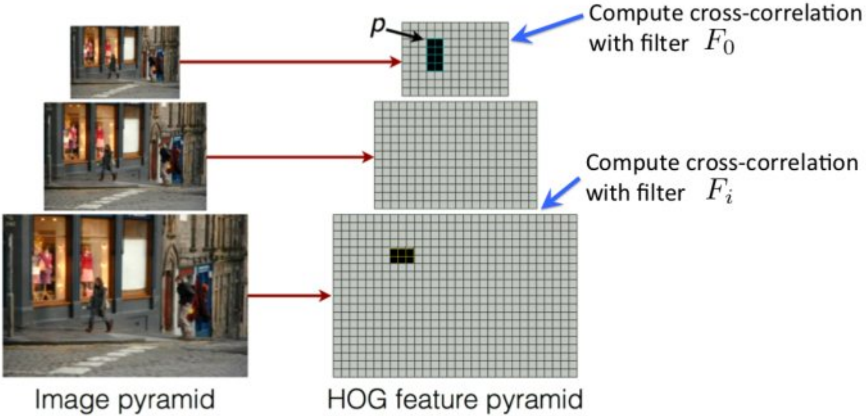


Figure: We can compute $F_i \cdot HOG(l, p_i)$ for the full level l via cross-correlation of the HOG feature matrix at level l with the template (filter) F_i

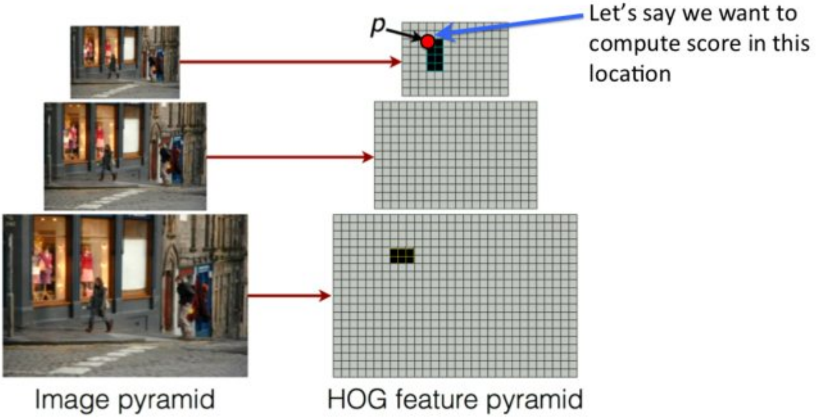
Computing the Score with Dynamic Programming

$$\text{score}(l, p_0) = F_0 \cdot HOG(l, p_0) + \sum_{i=1}^{n} \max_{p_i} \Big(F_i \cdot HOG(l, p_i) - \mathbf{w_{def}}^i \cdot \phi_{def}(x_i, y_i) \Big)$$



Computing the Score with Dynamic Programming

$$\text{score}(l, p_0) = F_0 \cdot HOG(l, p_0) + \sum_{i=1}^{n} \max_{p_i} \Big(F_i \cdot HOG(l, p_i) - \mathbf{w_{def}}^i \cdot \phi_{def}(x_i, y_i) \Big)$$



Computing the Score with Dynamic Programming

$$\text{score}(l, p_0) = F_0 \cdot HOG(l, p_0) + \sum_{i=1}^{n} \max_{p_i} \Big(F_i \cdot HOG(l, p_i) - \mathbf{w_{def}}^i \cdot \phi_{def}(x_i, y_i) \Big)$$

This is 0 in yellow point, because

$$(dx, dy, dx^2, dy^2) = (0, 0, 0, 0)$$

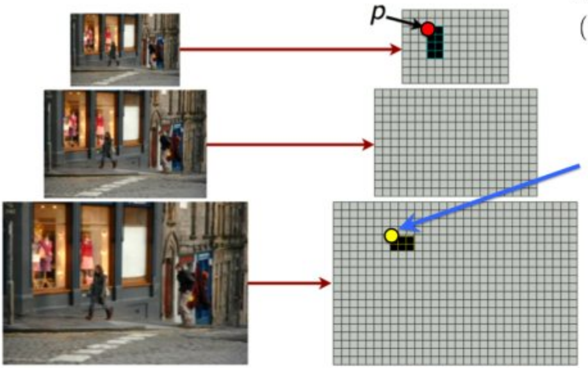


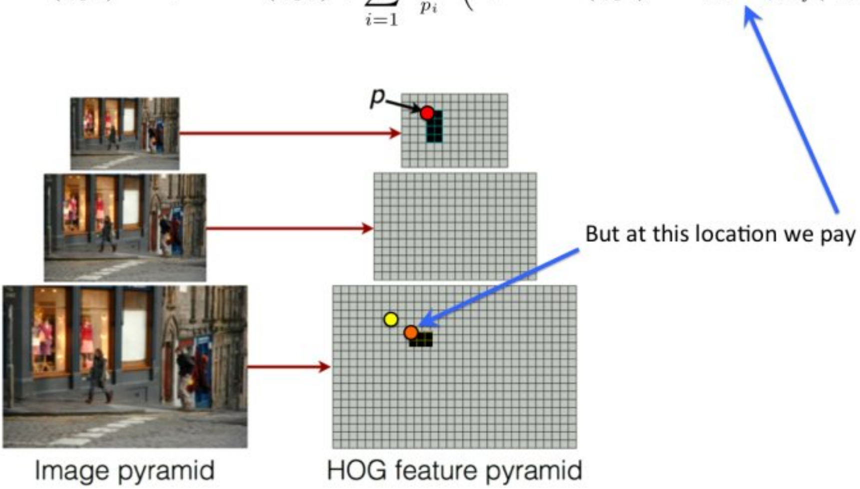
Image pyramid

HOG feature pyramid

There is no penalty for placing the part in the yellow location (the part is at expected location relative to the location of the root filter)

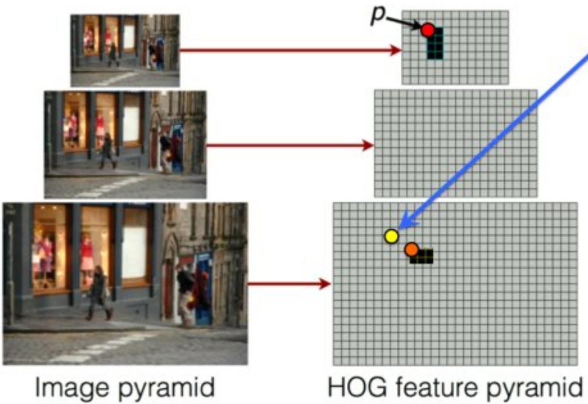
Computing the Score with Dynamic Programming

$$\text{score}(l, p_0) = F_0 \cdot HOG(l, p_0) + \sum_{i=1}^{n} \max_{p_i} \left(F_i \cdot HOG(l, p_i) - \mathbf{w_{def}}^i \cdot \phi_{def}(x_i, y_i) \right)$$



Computing the Score with Dynamic Programming

We are computing this: $$\max_{p_i} \left(F_i \cdot HOG(l, p_i) - \mathbf{w_{def}}^i \cdot \phi_{def}(x_i, y_i) \right)$$



We need to loop over all possible placements of the part. For each placement we need to:

- Compute deformation cost
- Read out the correlation value
- Subtract deformation from corr value

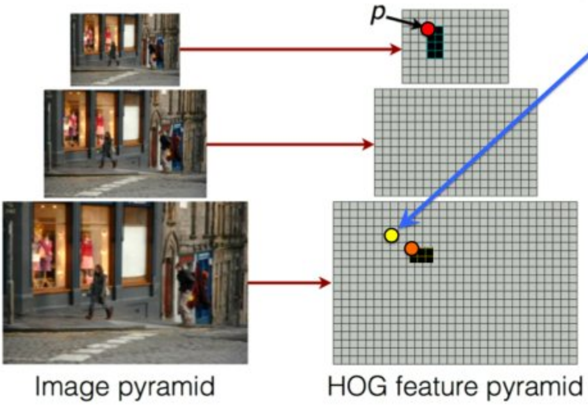
Find the max of these scores across all placements. Store the max in the yellow spot.

Figure: We can compute these scores efficiently with something called **distance transforms** (this is exact). But works equally well: Simply limit the scope of where each part could be to a small area, e.g., a few HOG cells up,down,left,right relative to yellow spot (this is approx).

Computing the Score with Dynamic Programming

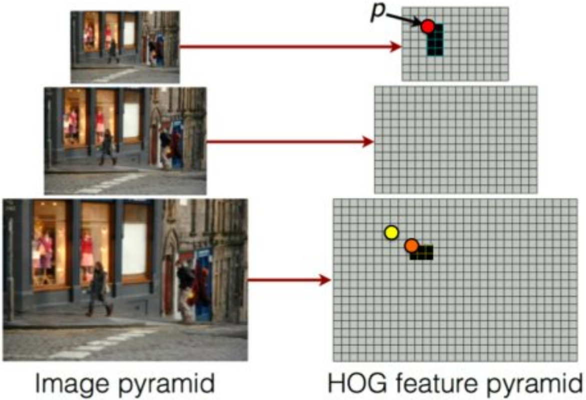
$$\sum_{i=1}^{n} \max_{p_i} \left(F_i \cdot HOG(l, p_i) - \mathbf{w_{def}}^{i} \cdot \phi_{def}(x_i, y_i) \right)$$

Do this for each part. Sum all the max part scores in the yellow spot



Computing the Score with Dynamic Programming

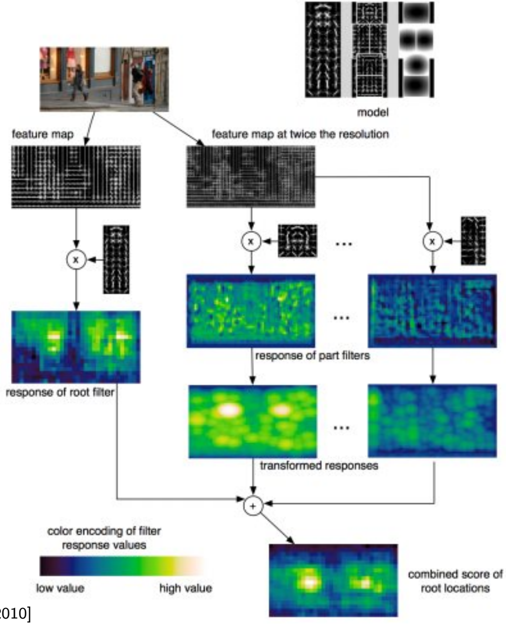
$$\text{score}(l, p_0) = F_0 \cdot HOG(l, p_0) + \sum_{i=1}^{n} \max_{p_i} \Big(F_i \cdot HOG(l, p_i) - \mathbf{w_{def}}^i \cdot \phi_{def}(x_i, y_i) \Big)$$



Add the value in the yellow location to the value in the red location.

Done!

Detection



[Pic from: Felzenswalb et al., 2010]

Training

- You can't train this model as simple as the HOG detector, via SVM. For those taking CSC411: Why not?

Training

- You can't train this model as simple as the HOG detector, via SVM. For those taking CSC411: Why not?
- Because the part positions are not annotated (we don't have ground-truth, and SVM needs ground-truth). We say that the parts are **latent**.
- You can train the model with something called **latent SVM**. For ML buffs:
    - Check the Felzenswalb paper
    - For those with even stronger ML stomach: Yu, Joachims, Learning Structural SVMs with Latent Variables, ICML'09.

Results

Figure: Performance of the HOG detector on person class on PASCAL VOC

[Pic from: R. Girshik]

Results

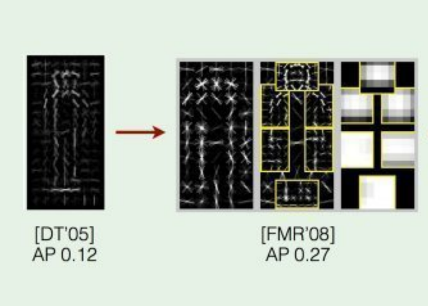


Figure: DPM version 1: adds the parts

[Pic from: R. Girshik]

Results

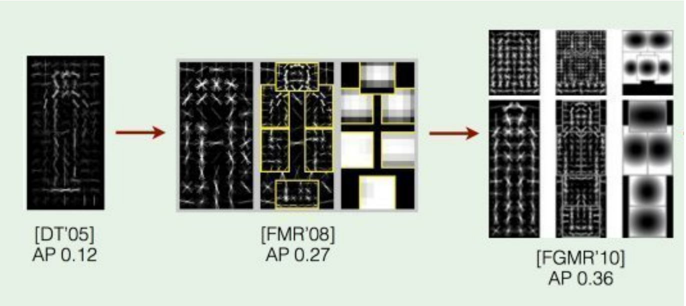


Figure: DPM version 2: adds another template (called mixture or component). Supposed to detect also people sitting down (e.g., occluded by desk).

[Pic from: R. Girshik]

Results

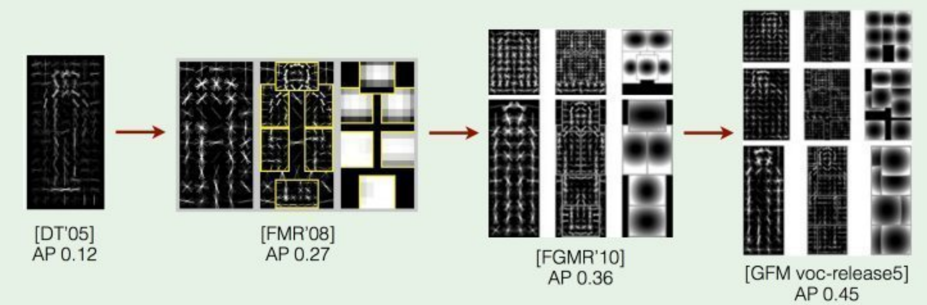
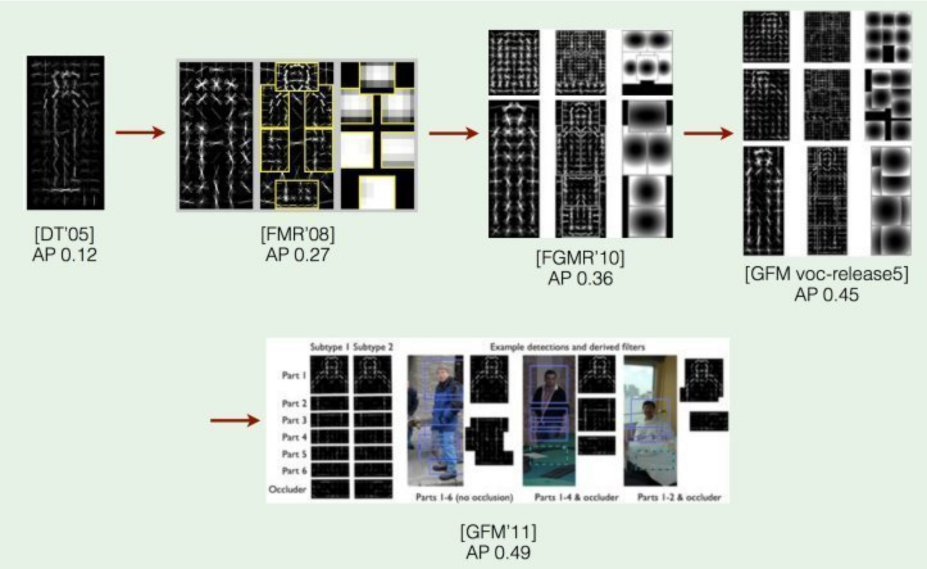


Figure: DPM version 3: adds multiple mixtures (components)

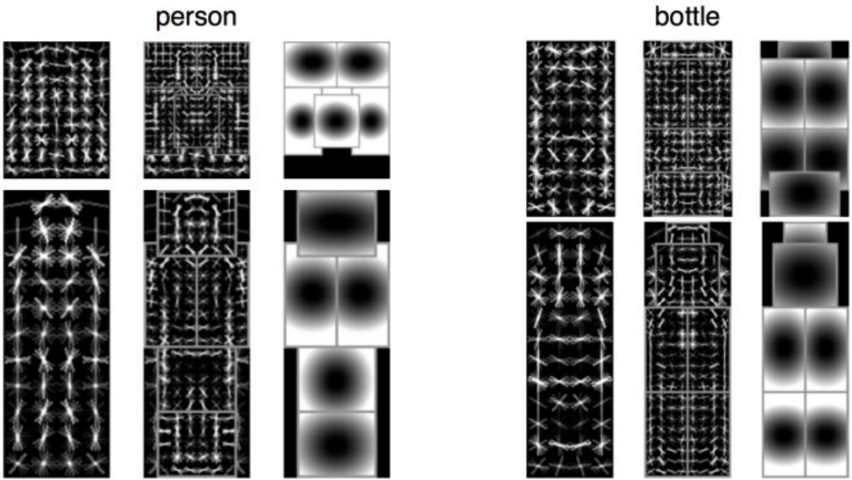
[Pic from: R. Girshik]

Results



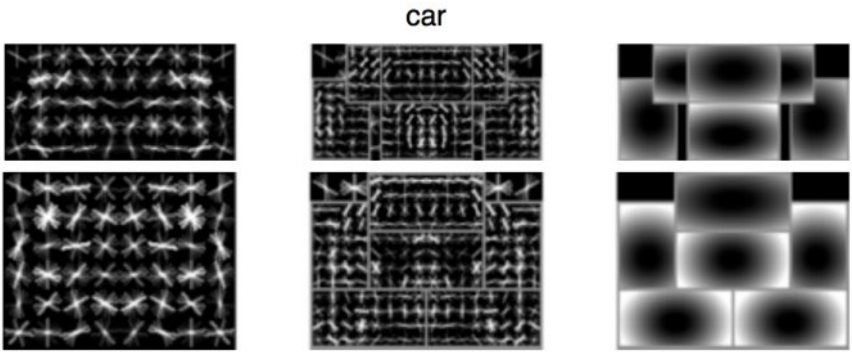
[Pic from: R. Girshik]

Learned Models



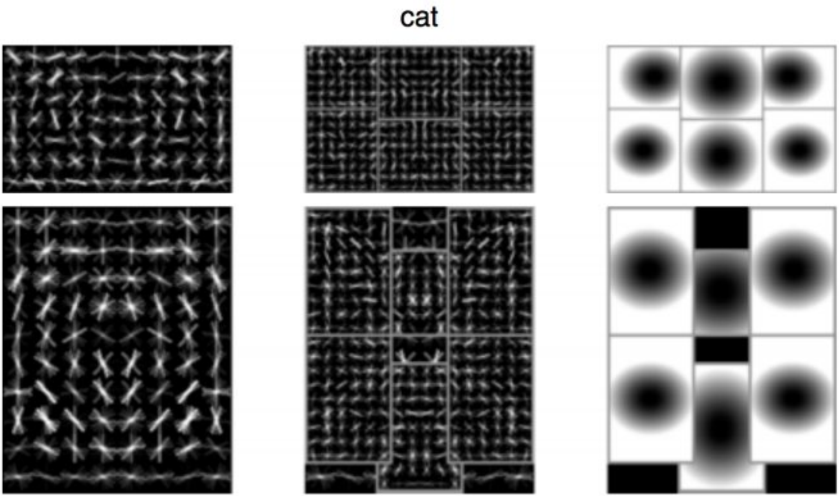
[Pic from: Felzenswalb et al., 2010]

Learned Models



[Pic from: Felzenswalb et al., 2010]

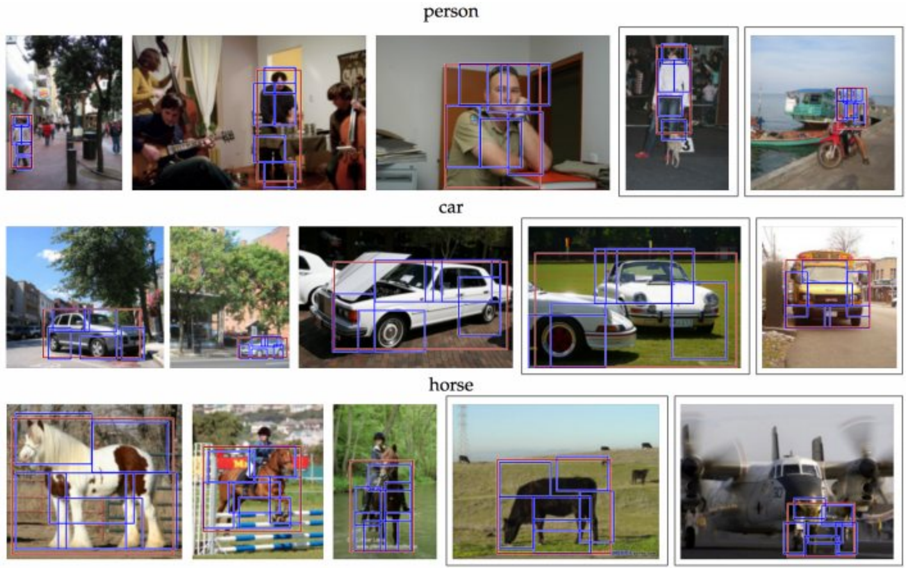
Learned Models



(Takes some imagination to see a cat...)

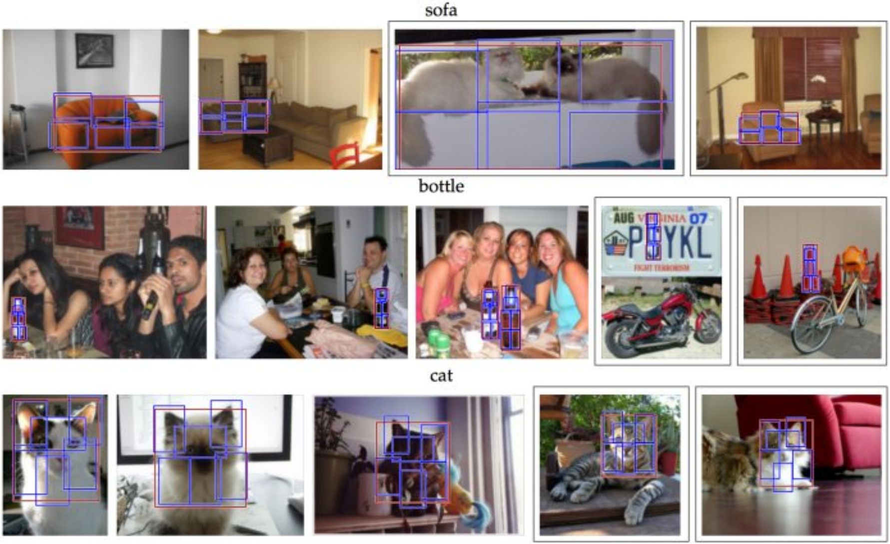
[Pic from: Felzenswalb et al., 2010]

Results



[Pic from: Felzenswalb et al., 2010]

Results



[Pic from: Felzenswalb et al., 2010]

DPM

- As you already know, the code is available:

  `http://www.cs.berkeley.edu/~rbg/latent/`

- Trivia:
  - Takes about 20-30 seconds per image per class. Speed-ups exist.
  - Depending on the size of the dataset, training takes around 12 hours (for most PASCAL classes).
  - Has some cool post-processing tricks: bounding box prediction and context re-scoring. Each typically results in around 2% improvement in AP.
  - In the code, if you switch off the parts, you get the Dalal & Triggs' HOG detector.