Simple Detection

D.A. Forsyth CS598 MAAV

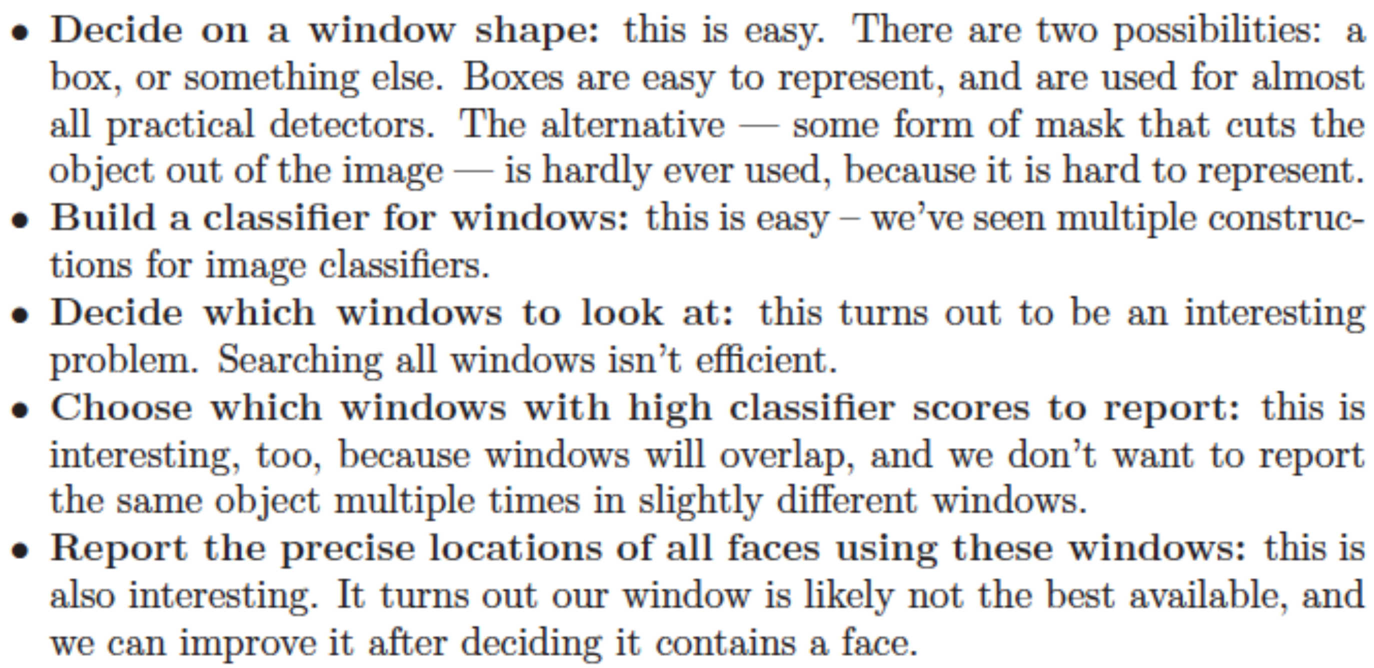
Classification vs detection

- Classification:
  - there is an X in this image
    - what
- Detection:
  - there is an X HERE in this image
    - what AND where

- Key issues
  - how to specify where
  - relationship between what and where
    - efficiency, etc
  - evaluation
    - surprisingly fiddly

Start simple

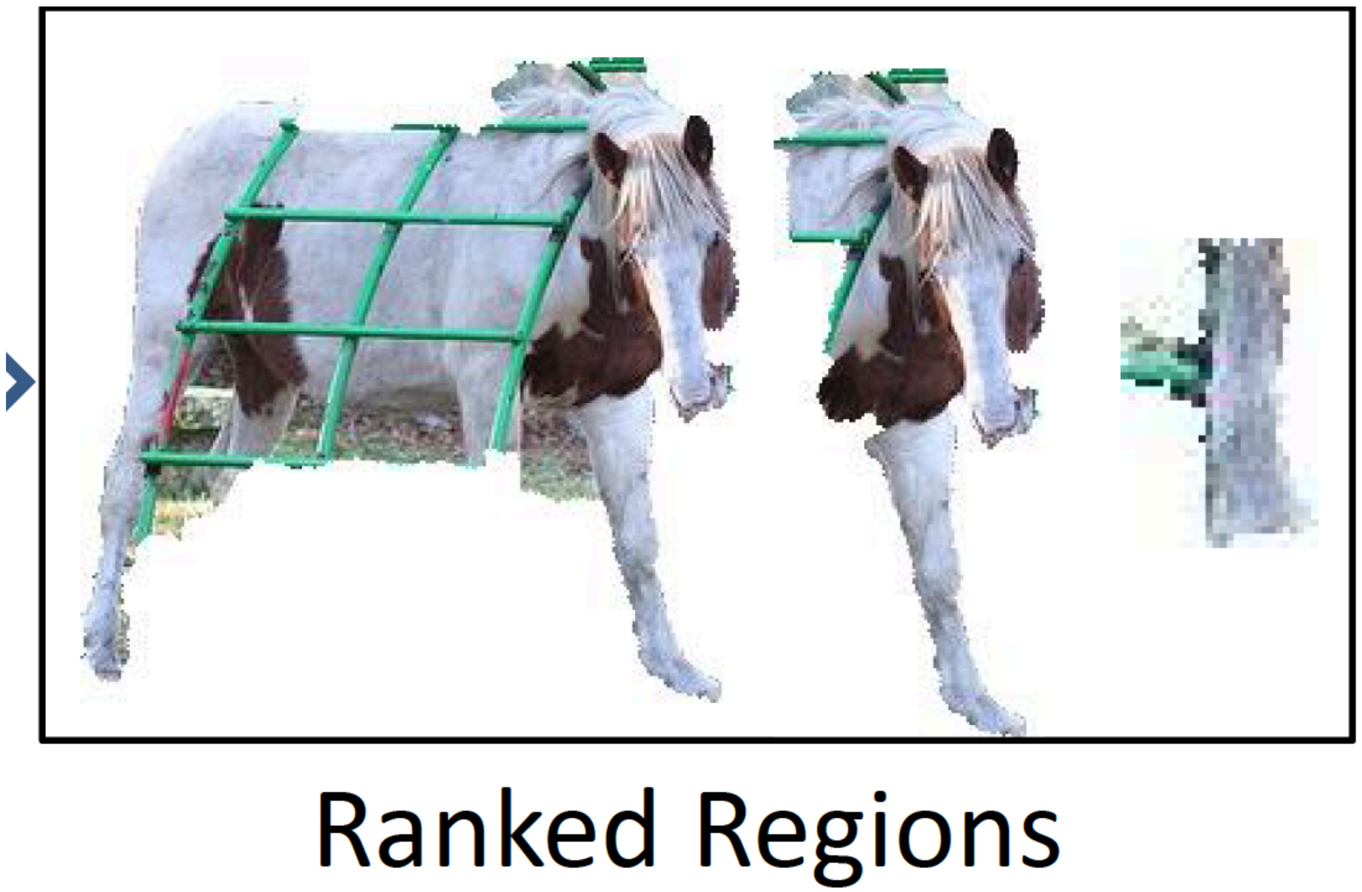
- Where = axis aligned box



- **Decide on a window shape:** this is easy. There are two possibilities: a box, or something else. Boxes are easy to represent, and are used for almost all practical detectors. The alternative — some form of mask that cuts the object out of the image — is hardly ever used, because it is hard to represent.
- **Build a classifier for windows:** this is easy – we've seen multiple constructions for image classifiers.
- **Decide which windows to look at:** this turns out to be an interesting problem. Searching all windows isn't efficient.
- **Choose which windows with high classifier scores to report:** this is interesting, too, because windows will overlap, and we don't want to report the same object multiple times in slightly different windows.
- **Report the precise locations of all faces using these windows:** this is also interesting. It turns out our window is likely not the best available, and we can improve it after deciding it contains a face.

Which window

- Surprising fact
  - Easy to tell whether a region is likely to be an object
    - even if you don’t know what object (Endres+Hoiem, 10; Uijlings et al 12)
    - if it’s an object
      - there’s contrast with surroundings in texture, etc
    - if not
      - often neighbor region is similar



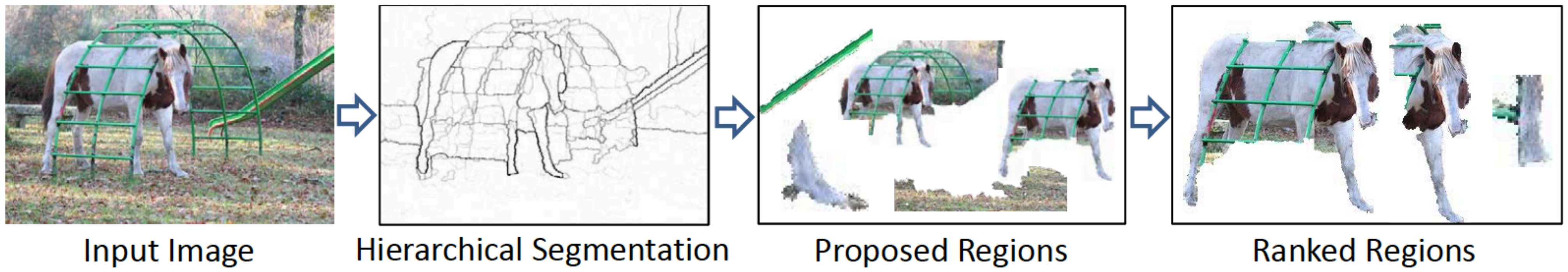


Fig. 1: Our pipeline: compute a hierarchical segmentation, generate proposals, and rank proposed regions. At each stage, we train classifiers to focus on likely object regions and encourage diversity among the proposals, enabling the system to localize many types of objects. See section 3 for a more detailed overview.

General strategy

- Construct hierarchy of image regions
    - using a hierarchical segmenter
- Rank regions using a learned score
- Make boxes out of high-ranking regions

- Selective search

Selective search pipeline

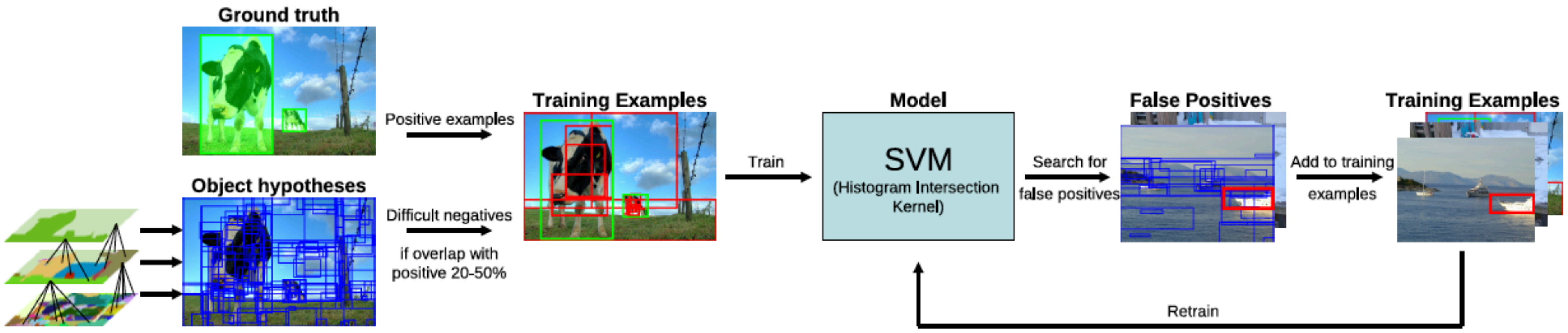
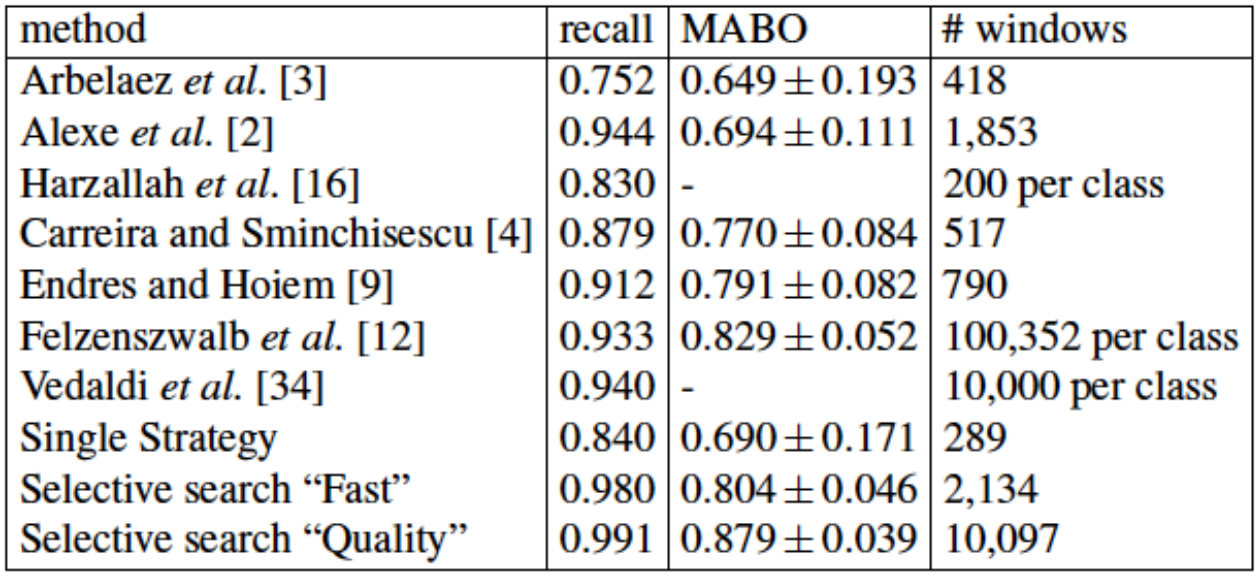


Figure 3: The training procedure of our object recognition pipeline. As positive learning examples we use the ground truth. As negatives we use examples that have a 20-50% overlap with the positive examples. We iteratively add hard negatives using a retraining phase.

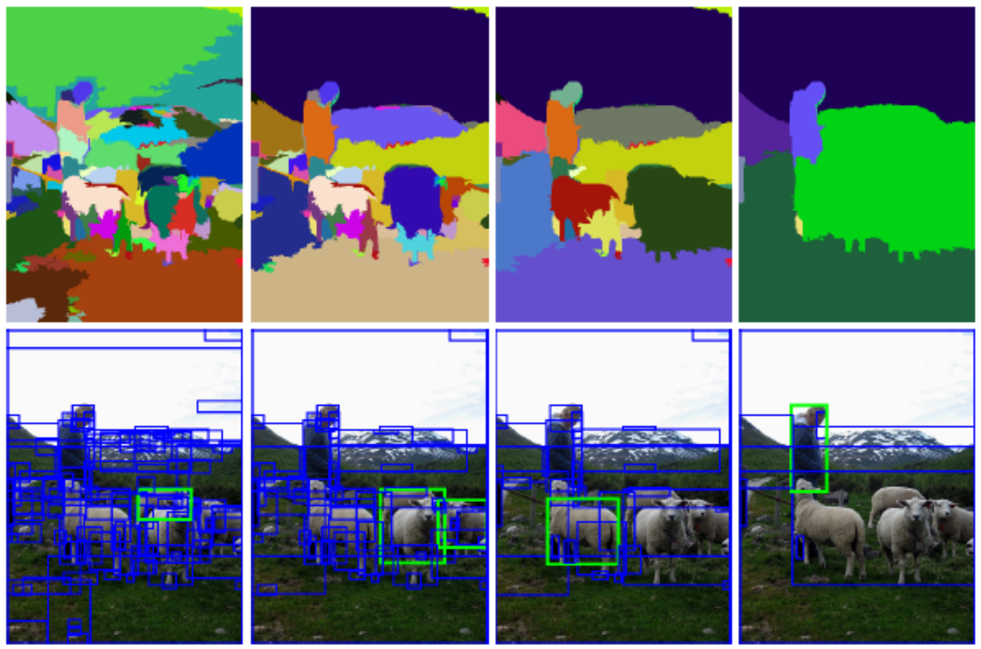
This sort of thing works well



method	recall	MABO	# windows
Arbelaez *et al.* [3]	0.752	0.649 ± 0.193	418
Alexe *et al.* [2]	0.944	0.694 ± 0.111	1,853
Harzallah *et al.* [16]	0.830	-	200 per class
Carreira and Sminchisescu [4]	0.879	0.770 ± 0.084	517
Endres and Hoiem [9]	0.912	0.791 ± 0.082	790
Felzenszwalb *et al.* [12]	0.933	0.829 ± 0.052	100,352 per class
Vedaldi *et al.* [34]	0.940	-	10,000 per class
Single Strategy	0.840	0.690 ± 0.171	289
Selective search "Fast"	0.980	0.804 ± 0.046	2,134
Selective search "Quality"	0.991	0.879 ± 0.039	10,097

Table 5: Comparison of recall, Mean Average Best Overlap (MABO) and number of window locations for a variety of methods on the Pascal 2007 TEST set.

You need to search at multiple scales



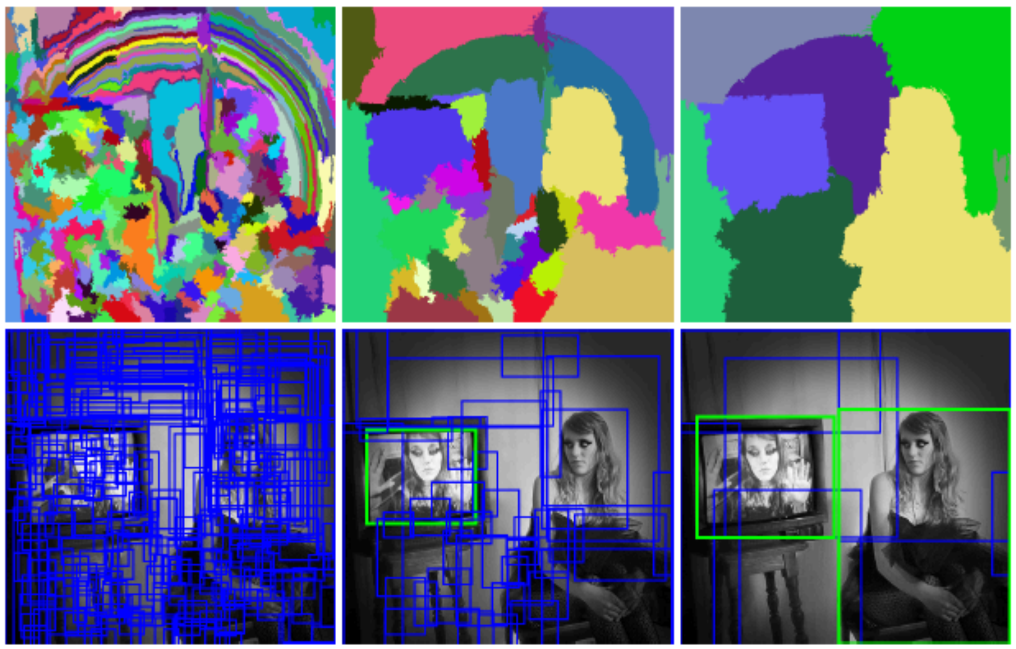


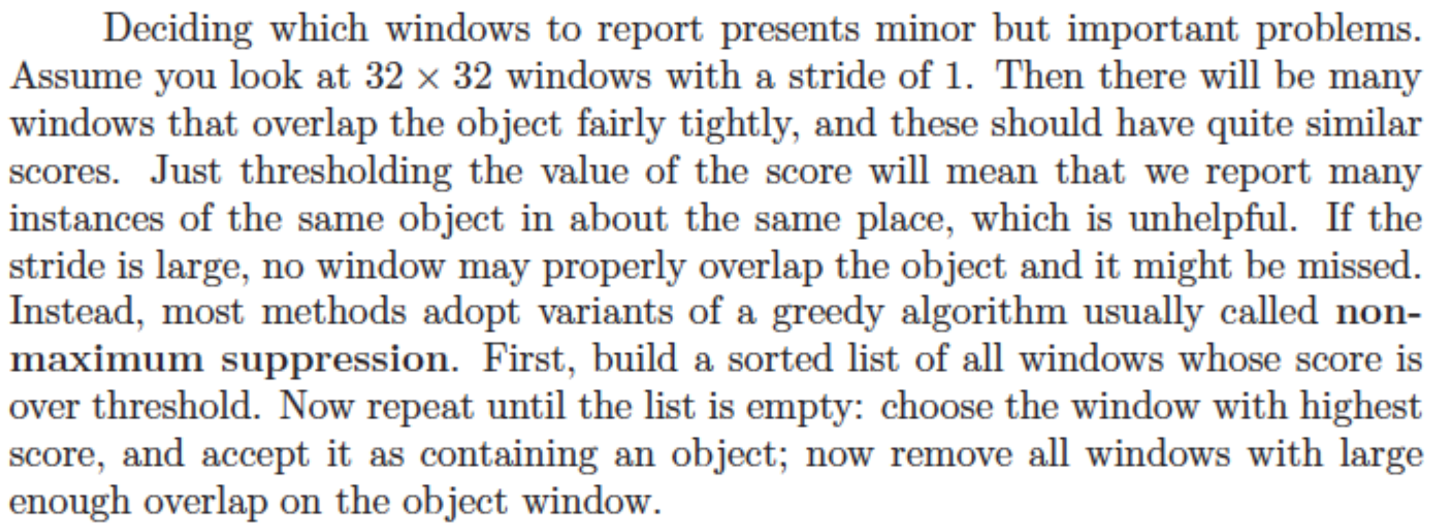
Figure 2: Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales. On the right we necessarily find the objects at different scales as the girl is contained by the tv.

Simplest detector

- Use selective search to propose boxes
- Check with classifier

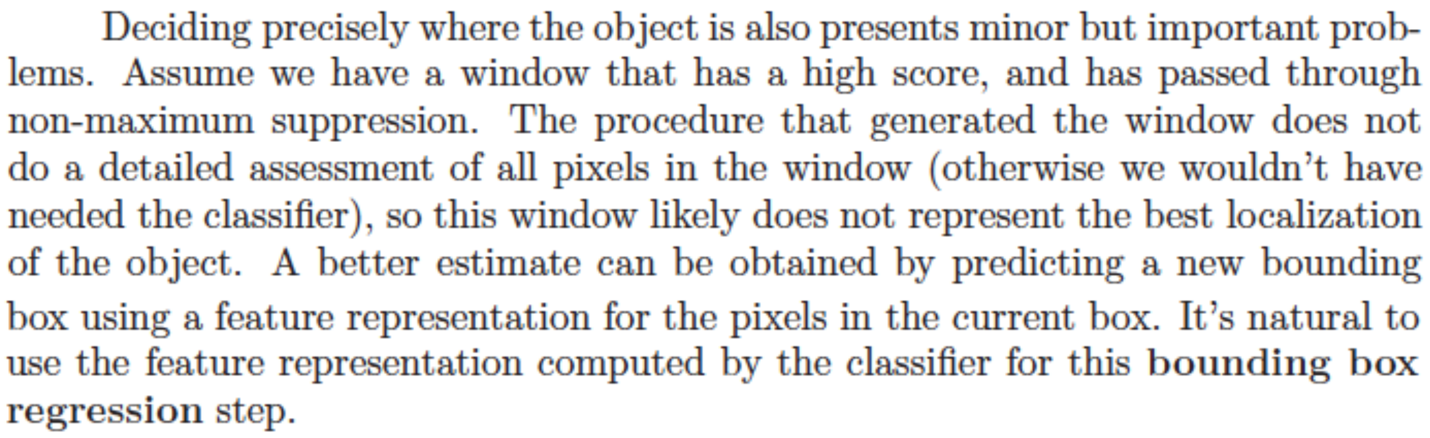
- BUT
  - boxes likely overlap - non-maximum suppression
  - boxes likely in poor location - bounding box regression

Non maximum suppression



Deciding which windows to report presents minor but important problems. Assume you look at 32×32 windows with a stride of 1. Then there will be many windows that overlap the object fairly tightly, and these should have quite similar scores. Just thresholding the value of the score will mean that we report many instances of the same object in about the same place, which is unhelpful. If the stride is large, no window may properly overlap the object and it might be missed. Instead, most methods adopt variants of a greedy algorithm usually called **non-maximum suppression**. First, build a sorted list of all windows whose score is over threshold. Now repeat until the list is empty: choose the window with highest score, and accept it as containing an object; now remove all windows with large enough overlap on the object window.

Bounding box regression



Deciding precisely where the object is also presents minor but important problems. Assume we have a window that has a high score, and has passed through non-maximum suppression. The procedure that generated the window does not do a detailed assessment of all pixels in the window (otherwise we wouldn't have needed the classifier), so this window likely does not represent the best localization of the object. A better estimate can be obtained by predicting a new bounding box using a feature representation for the pixels in the current box. It's natural to use the feature representation computed by the classifier for this **bounding box regression** step.

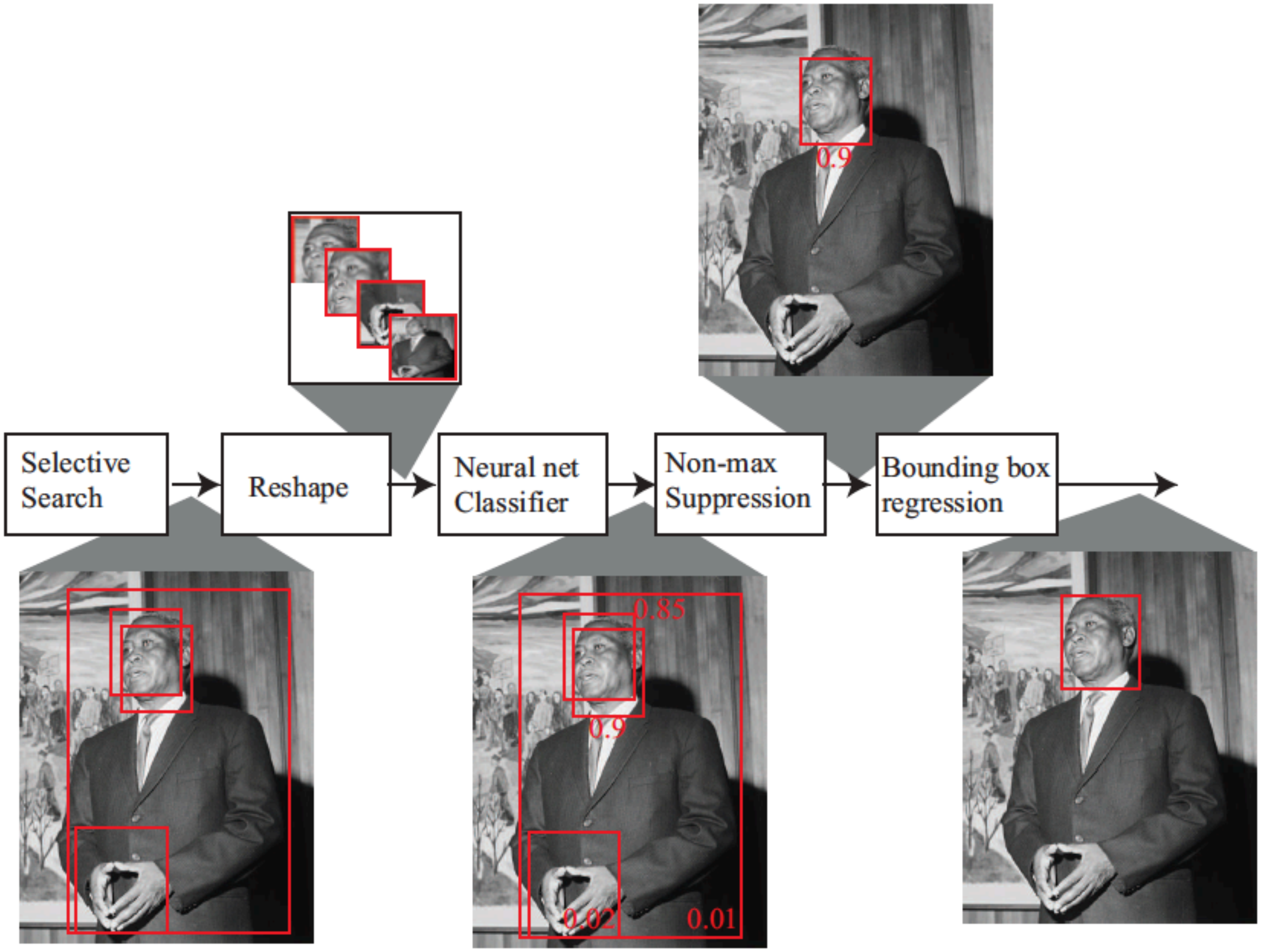


FIGURE 18.6: *A schematic picture of how R-CNN works. A picture of Inkosi Albert Luthuli is fed in to selective search, which proposes possible boxes; these are cut out of the image, and reshaped to fixed size; the boxes are classified (scores next to each box); non-maximum suppression finds high scoring boxes and suppresses nearby high scoring boxes (so his face isn't found twice); and finally bounding box regression adjusts the corners of the box to get the best fit using the features inside the box.*

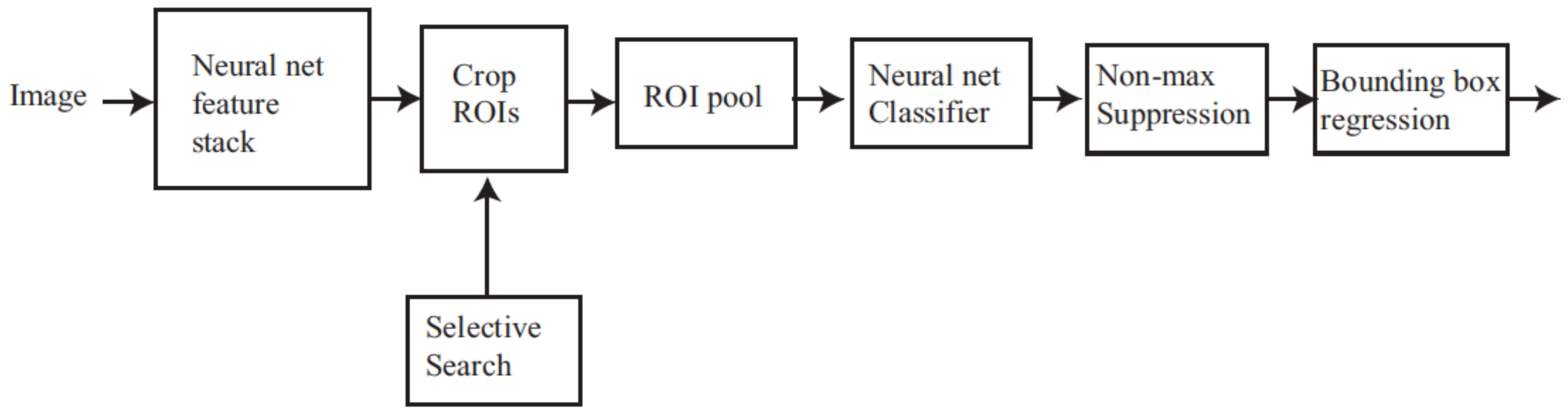
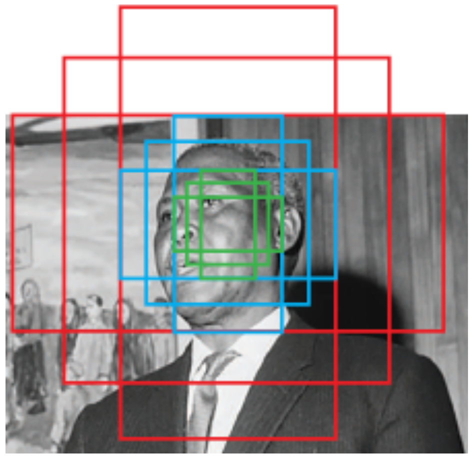


FIGURE 18.7: *Fast R-CNN is much more efficient than R-CNN, because it computes a single feature map from the image, then uses the boxes proposed by selective search to cut regions of interest (ROI's) from it. These are mapped to a standard size by a ROI pooling layer, then presented to a classifier. The rest should be familiar.*

Configuration spaces

- You should think of a box as a point in a 4D space
  - configuration space of the boxes
- Selective search is weird
  - networks don't do lists much
- Alternative
  - sample the configuration space on some form of grid
    - eg three aspect ratios, three scales, grid of locations
    - important: many possible sampling schemes
  - check each sample with rank score

Anchor boxes



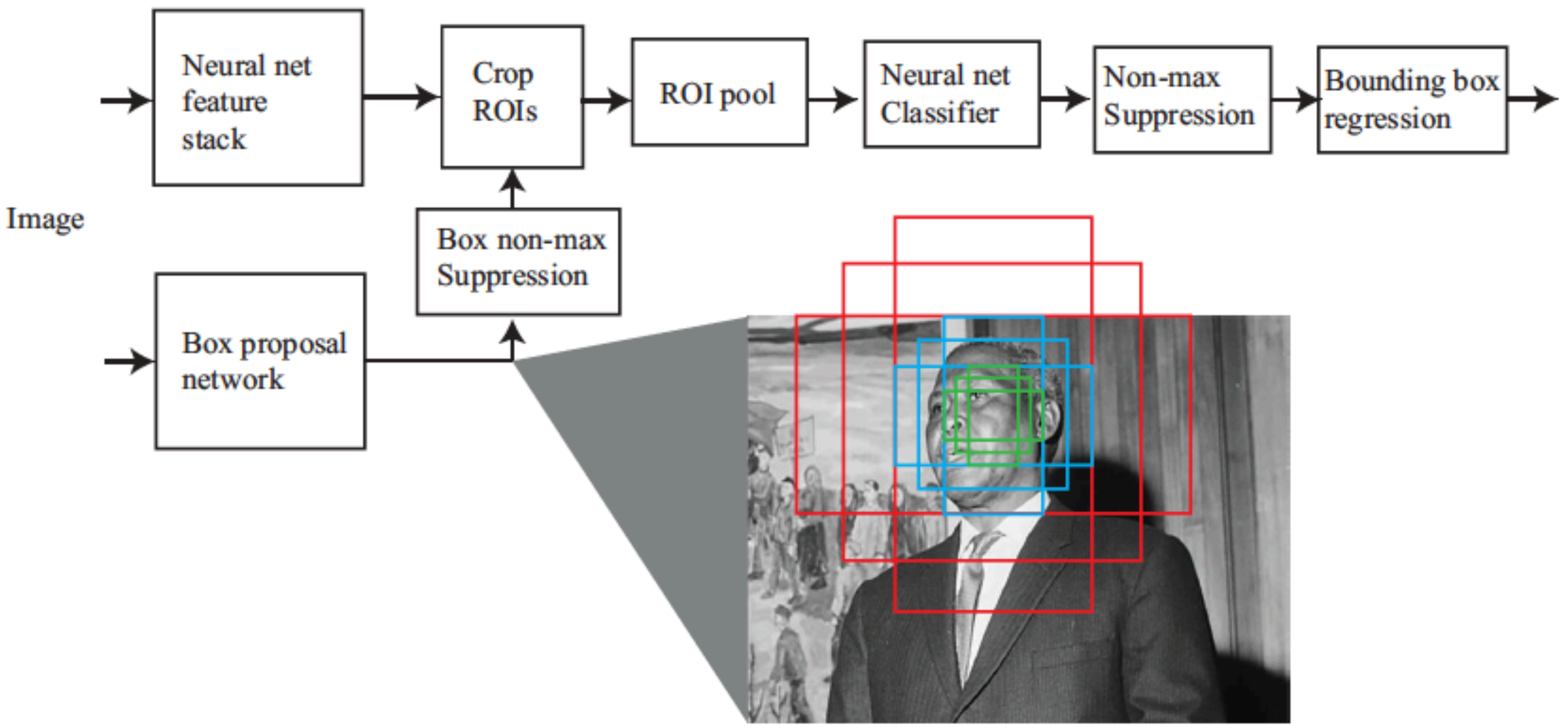
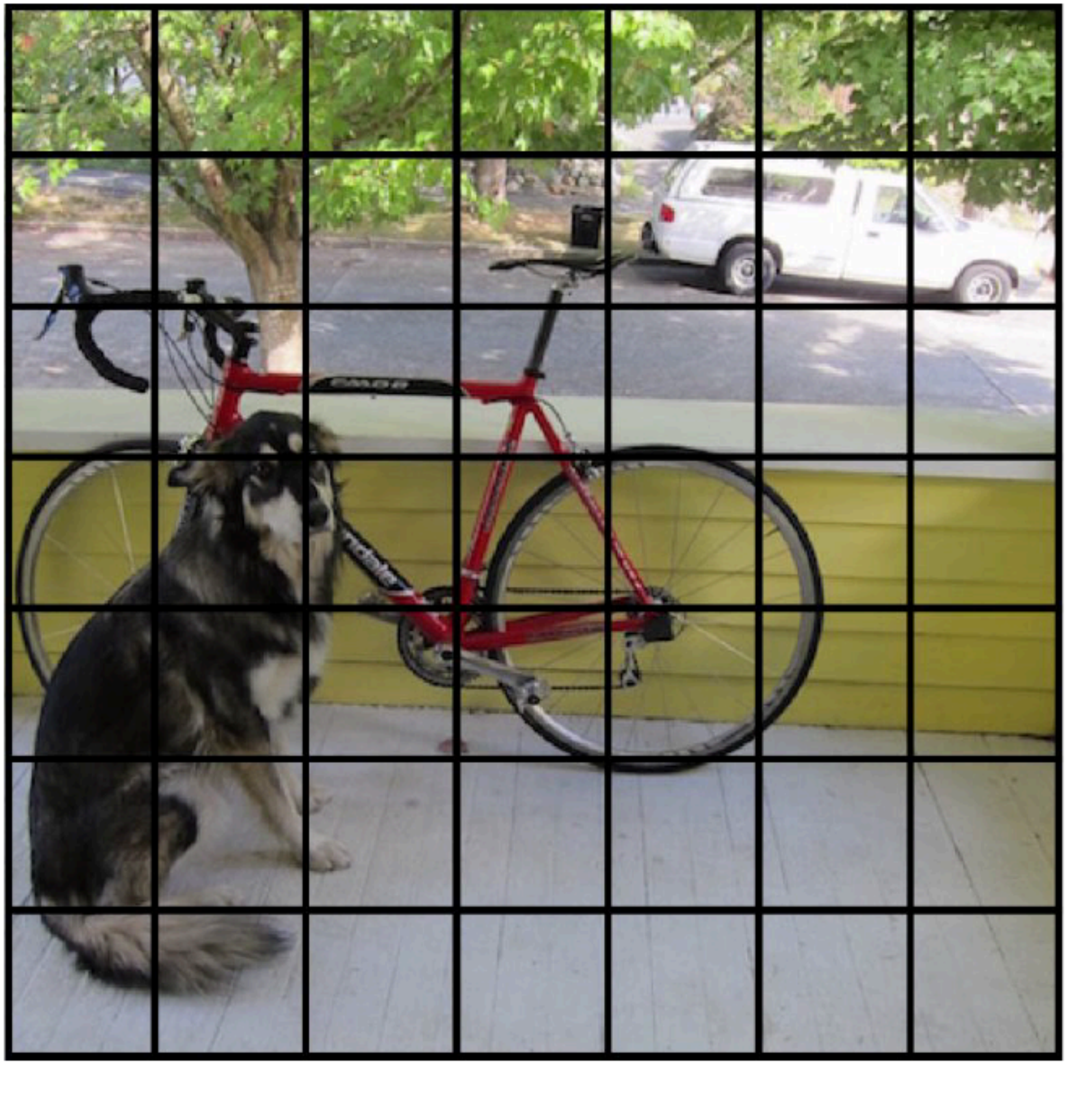


FIGURE 18.8: *Faster RCNN uses two networks. One uses the image to compute "objectness" scores for a sampling of possible image boxes. The samples (called "anchor boxes") are each centered at a grid point. At each grid point, there are nine boxes (three scales, three aspect ratios). The second is a feature stack that computes a representation of the image suitable for classification. The boxes with highest objectness score are then cut from the feature map, standardized with ROI pooling, then passed to a classifier. Bounding box regression means that the relatively coarse sampling of locations, scales and aspect ratios does not weaken accuracy.*

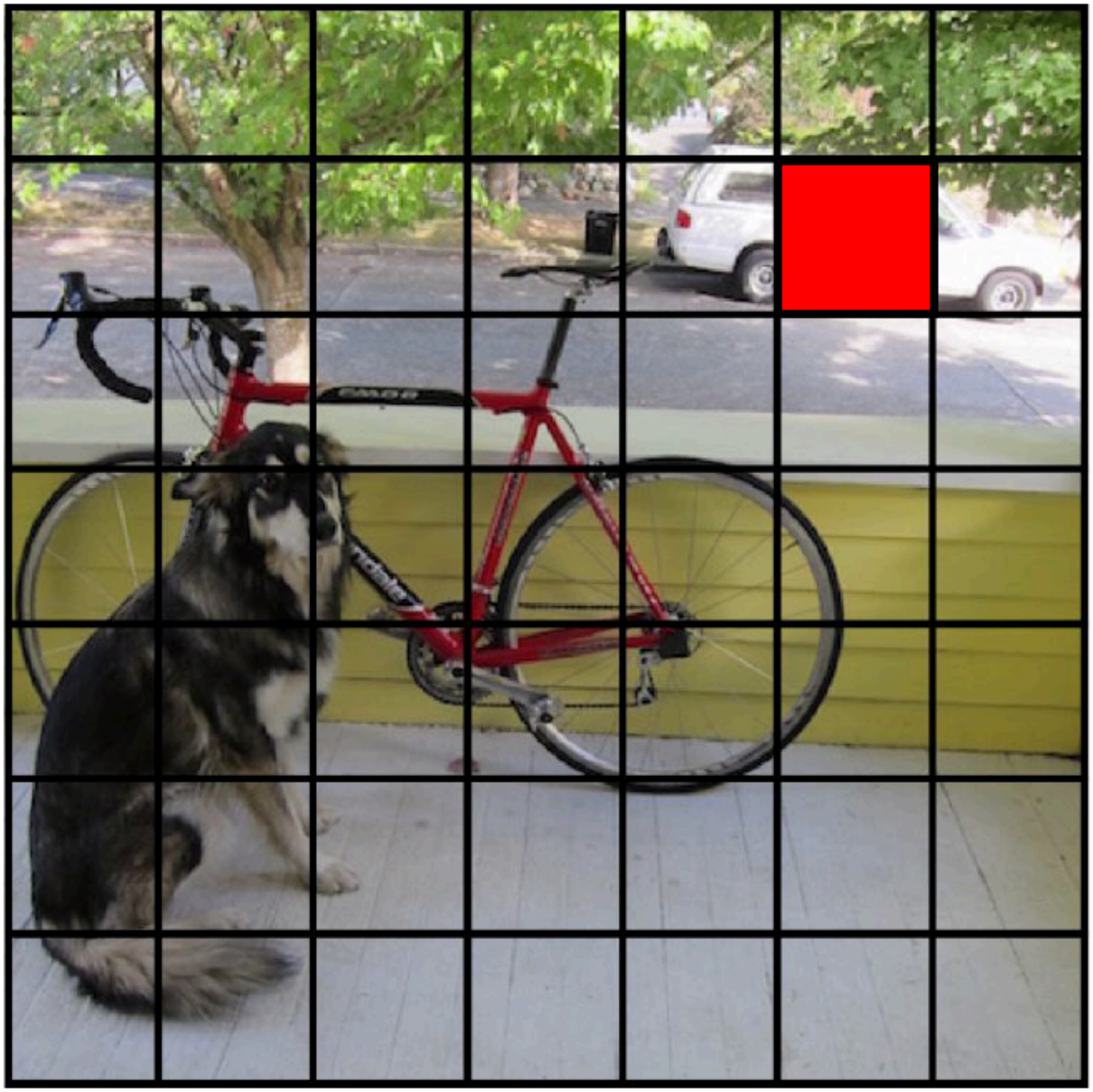
YOLO

- YOLO v3 is about as fast and accurate as you can get
- link on webpage
- key idea
    - look at box scores, label values independently

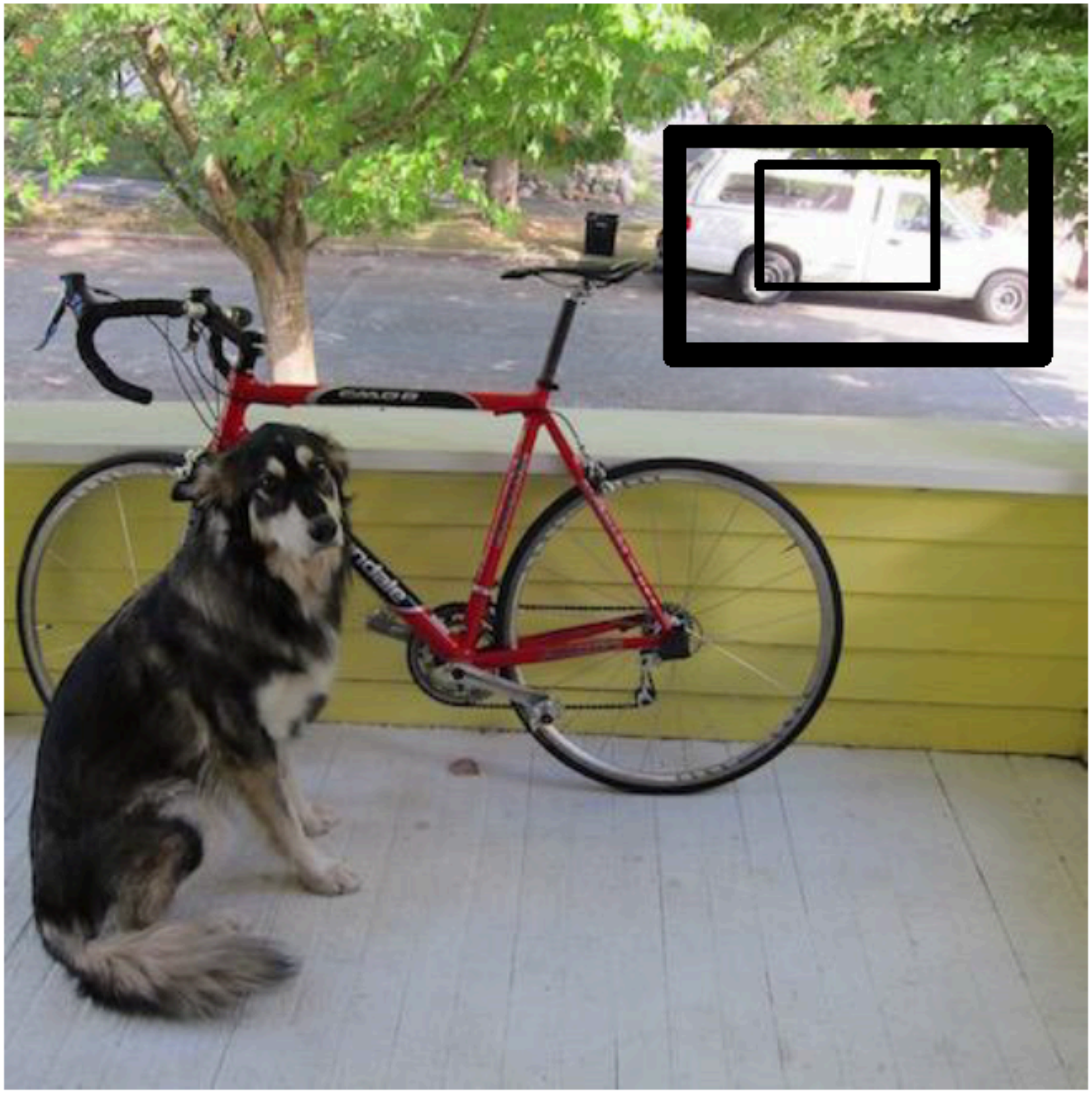
We split the image into a grid



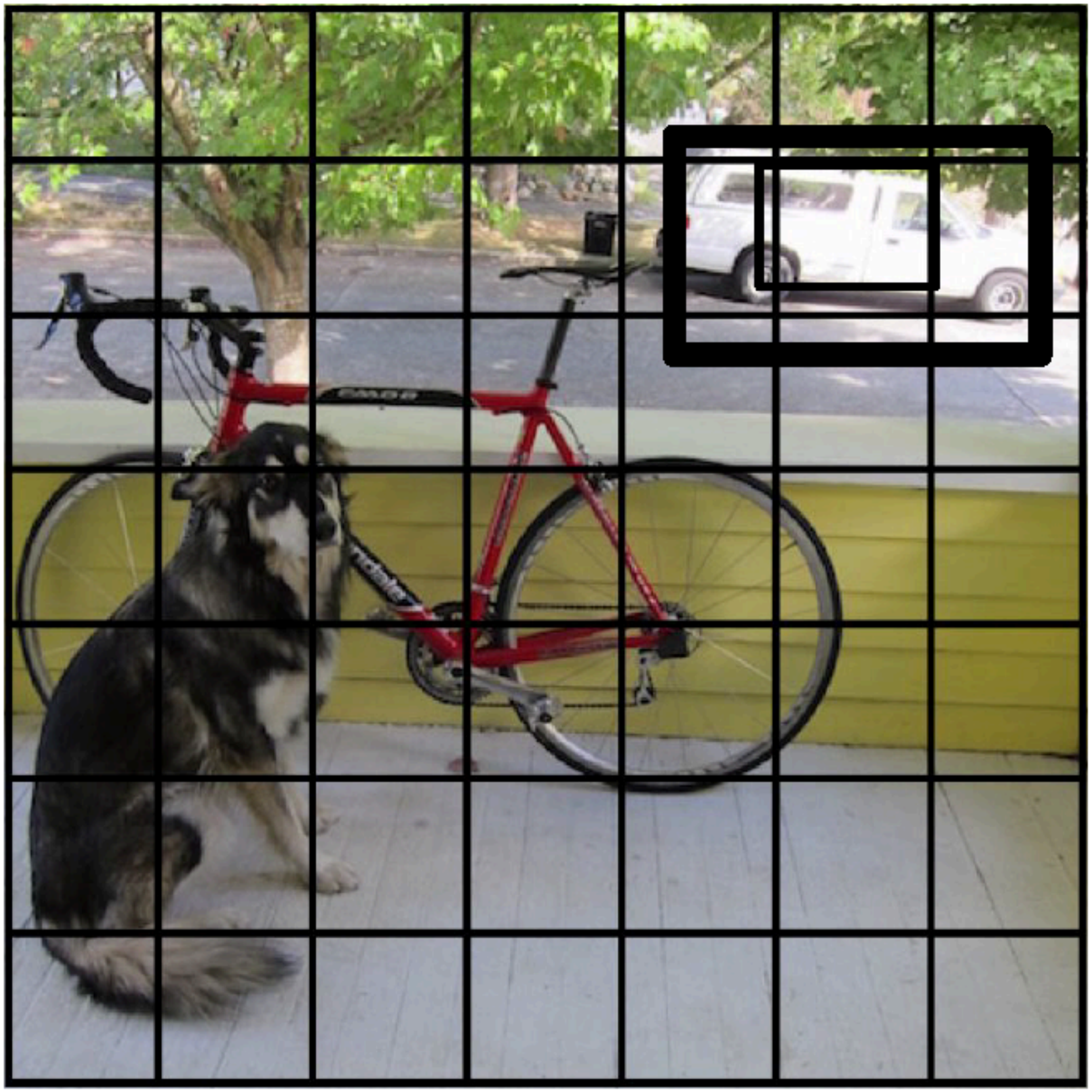
Each cell predicts boxes and confidences: P(Object)



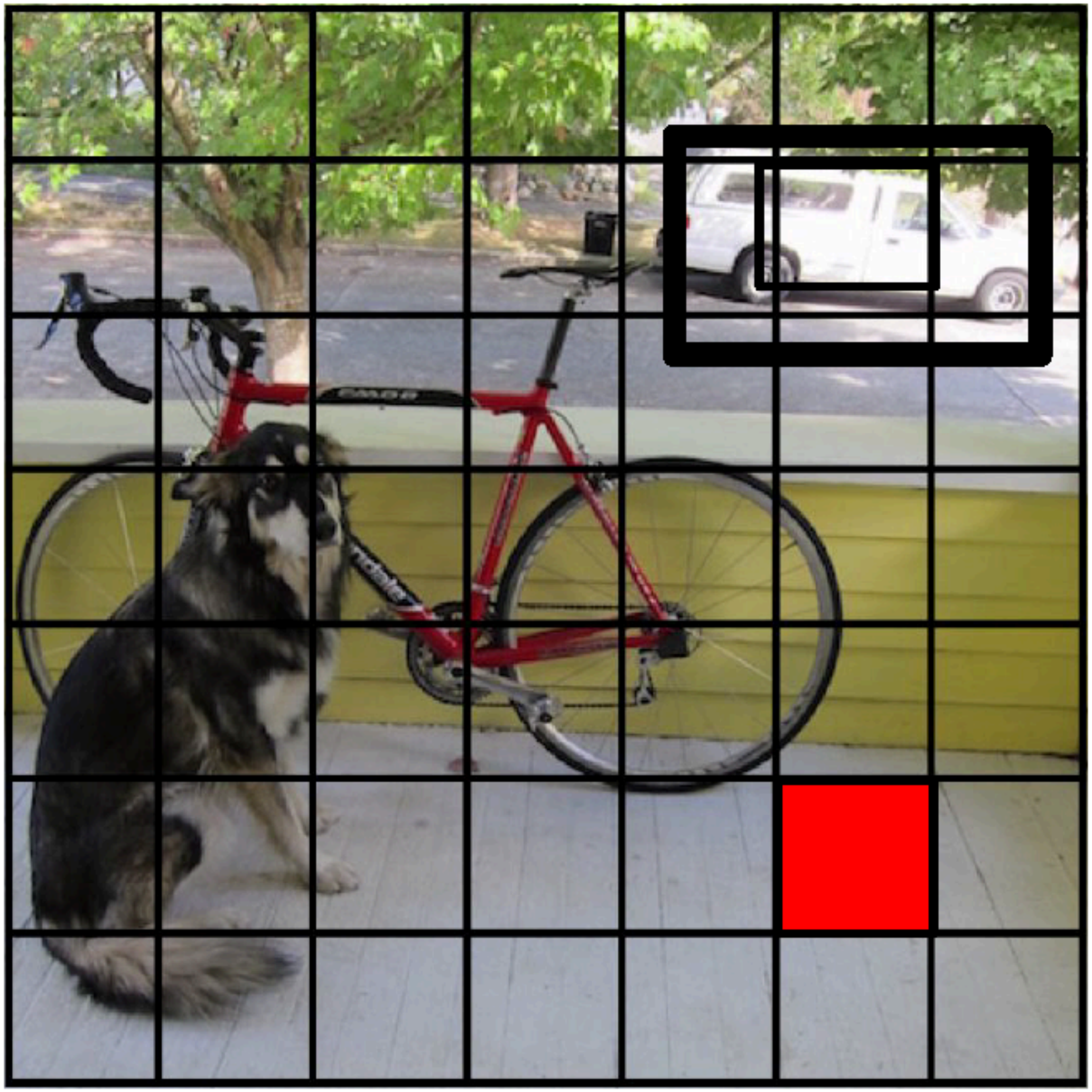
Each cell predicts boxes and confidences: P(Object)



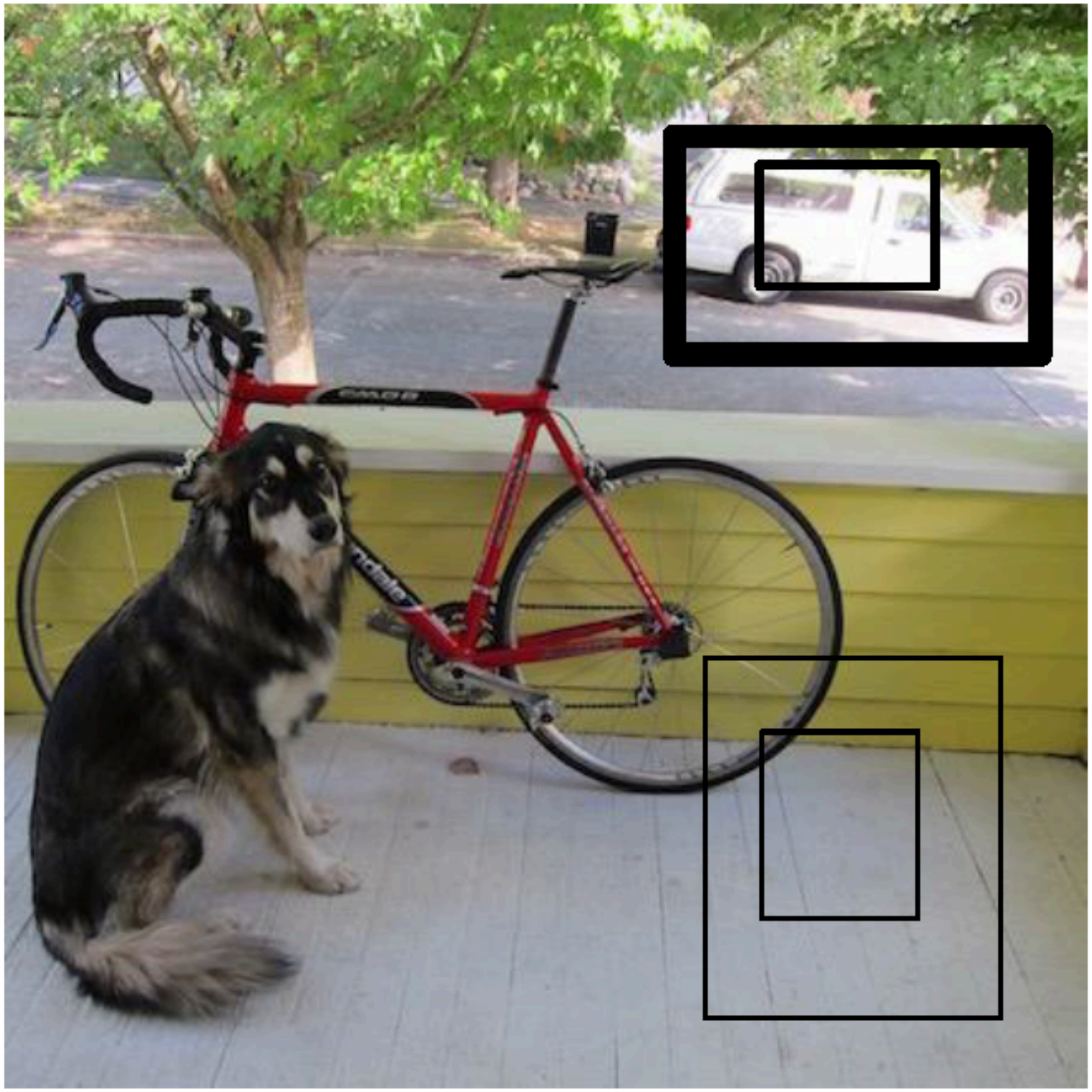
Each cell predicts boxes and confidences: P(Object)



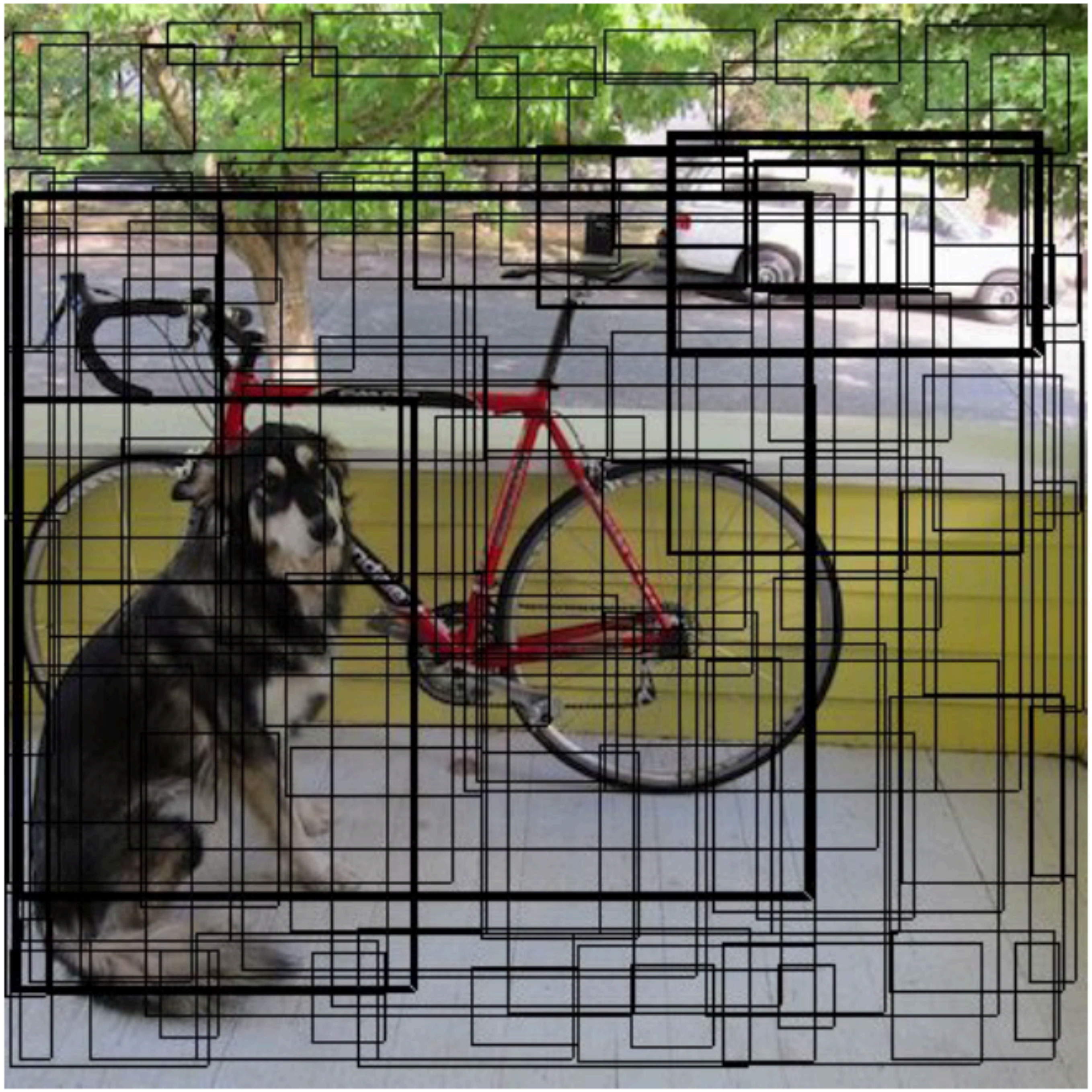
Each cell predicts boxes and confidences: P(Object)



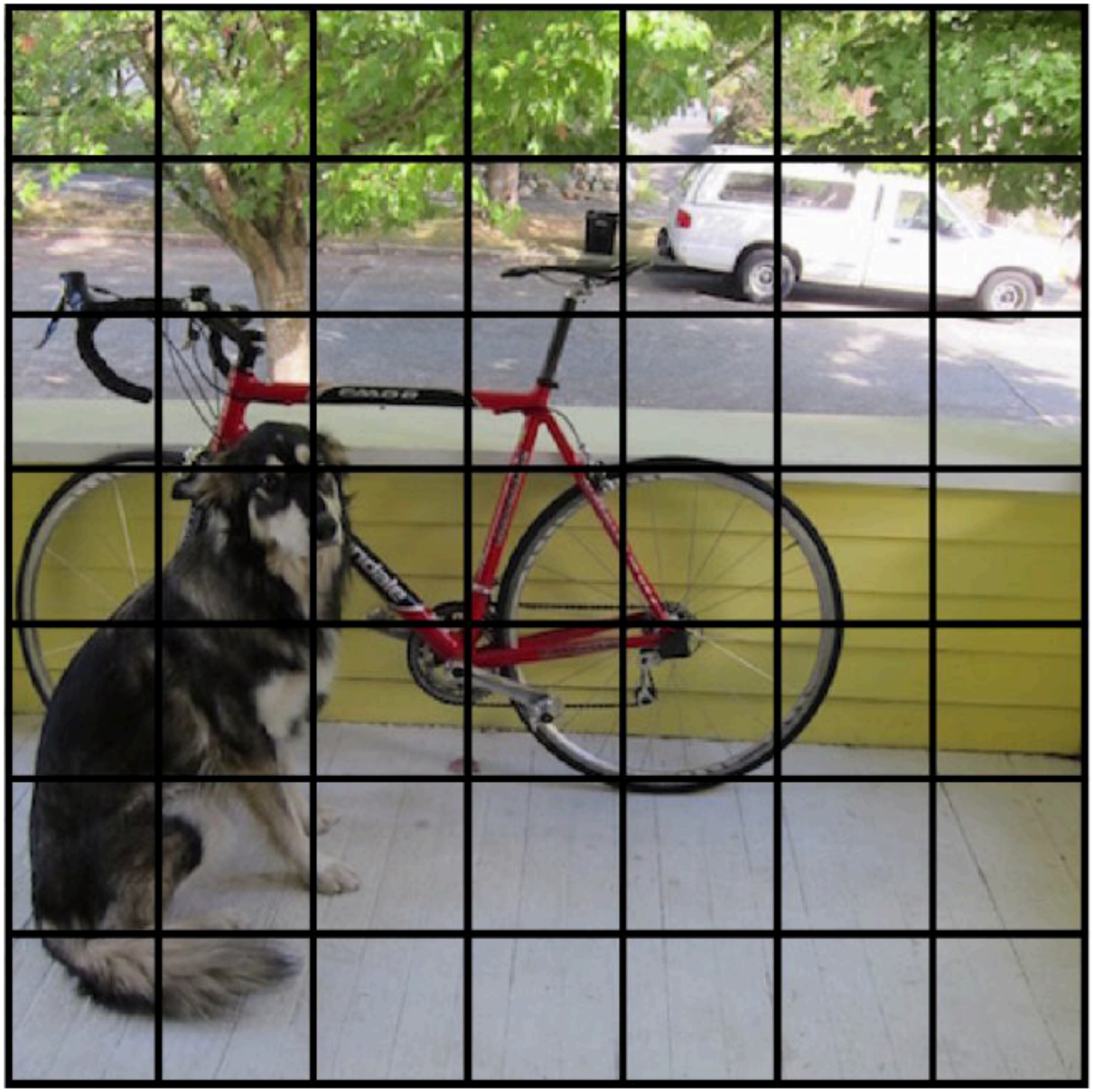
Each cell predicts boxes and confidences: P(Object)



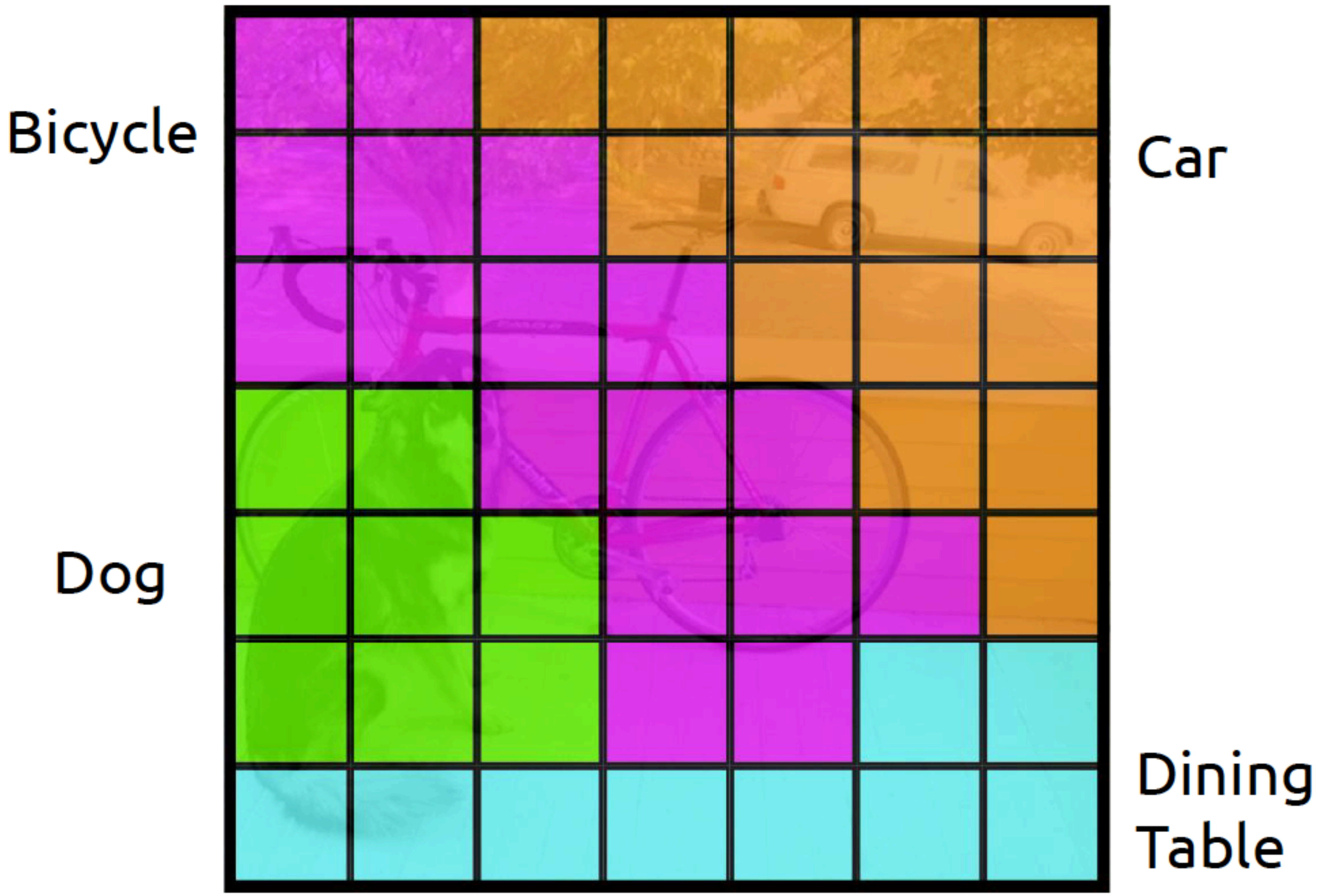
Each cell predicts boxes and confidences: P(Object)



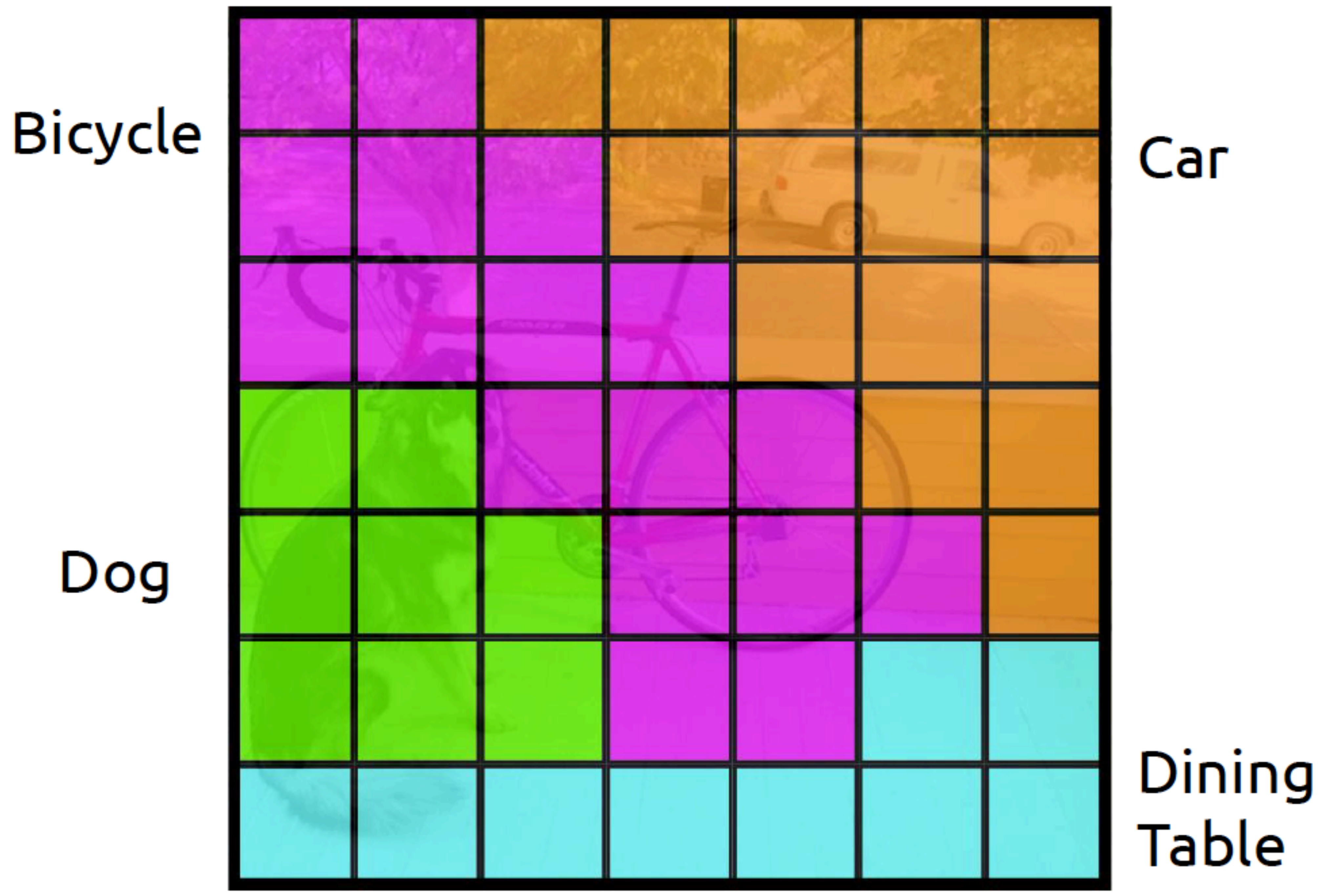
Each cell also predicts a class probability.



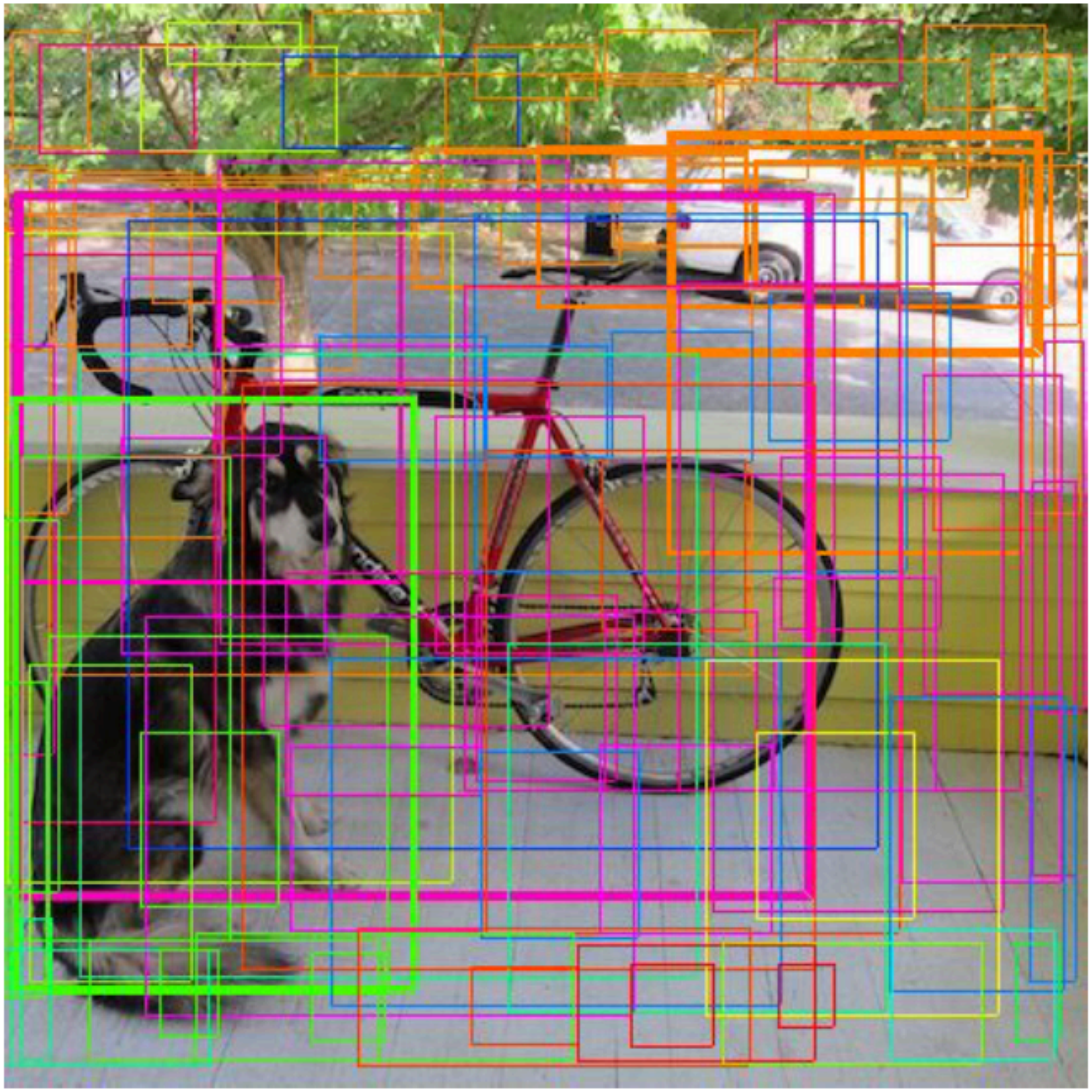
Each cell also predicts a class probability.



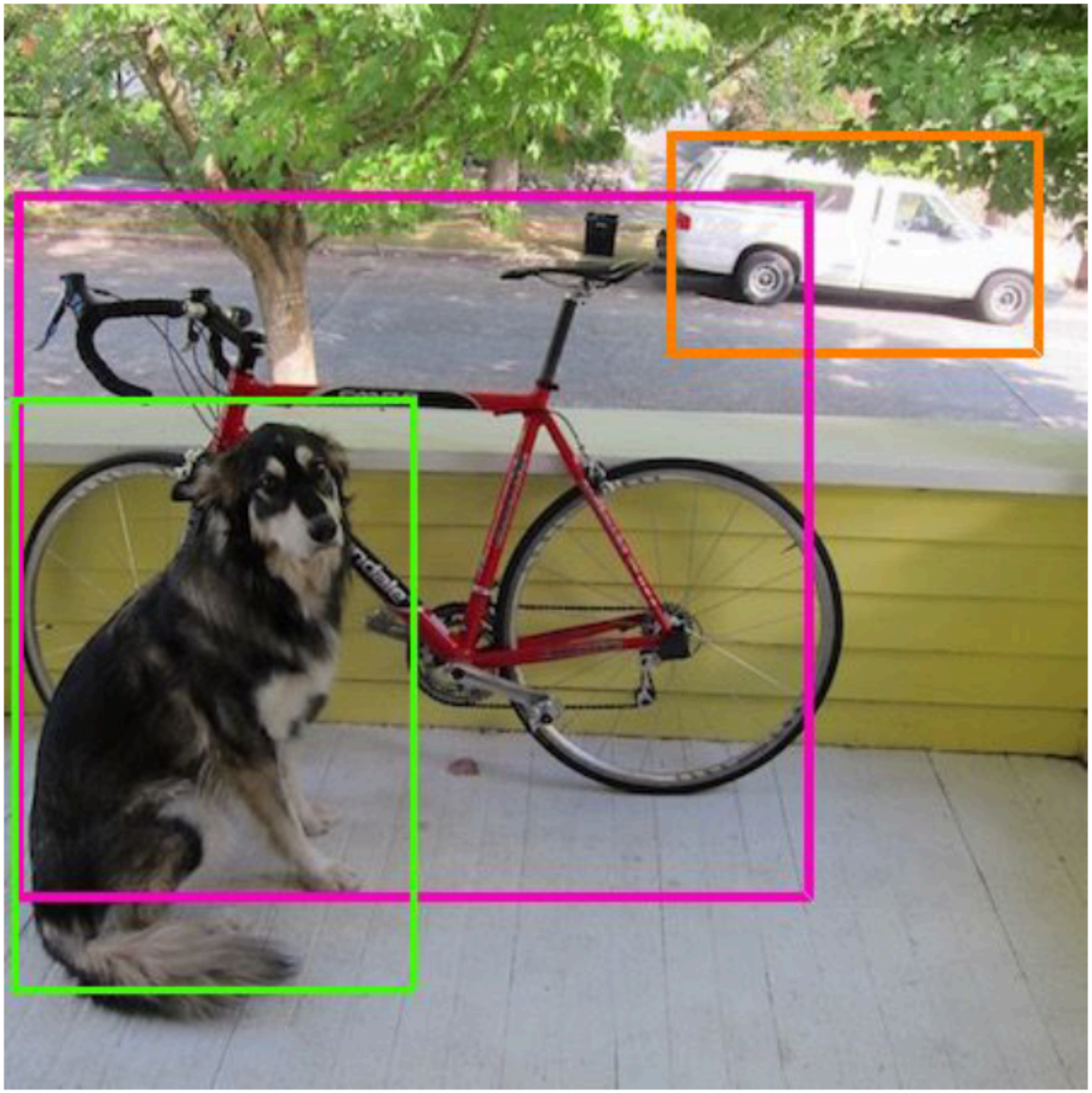
Conditioned on object: P(Car | Object)



Then we combine the box and class predictions.



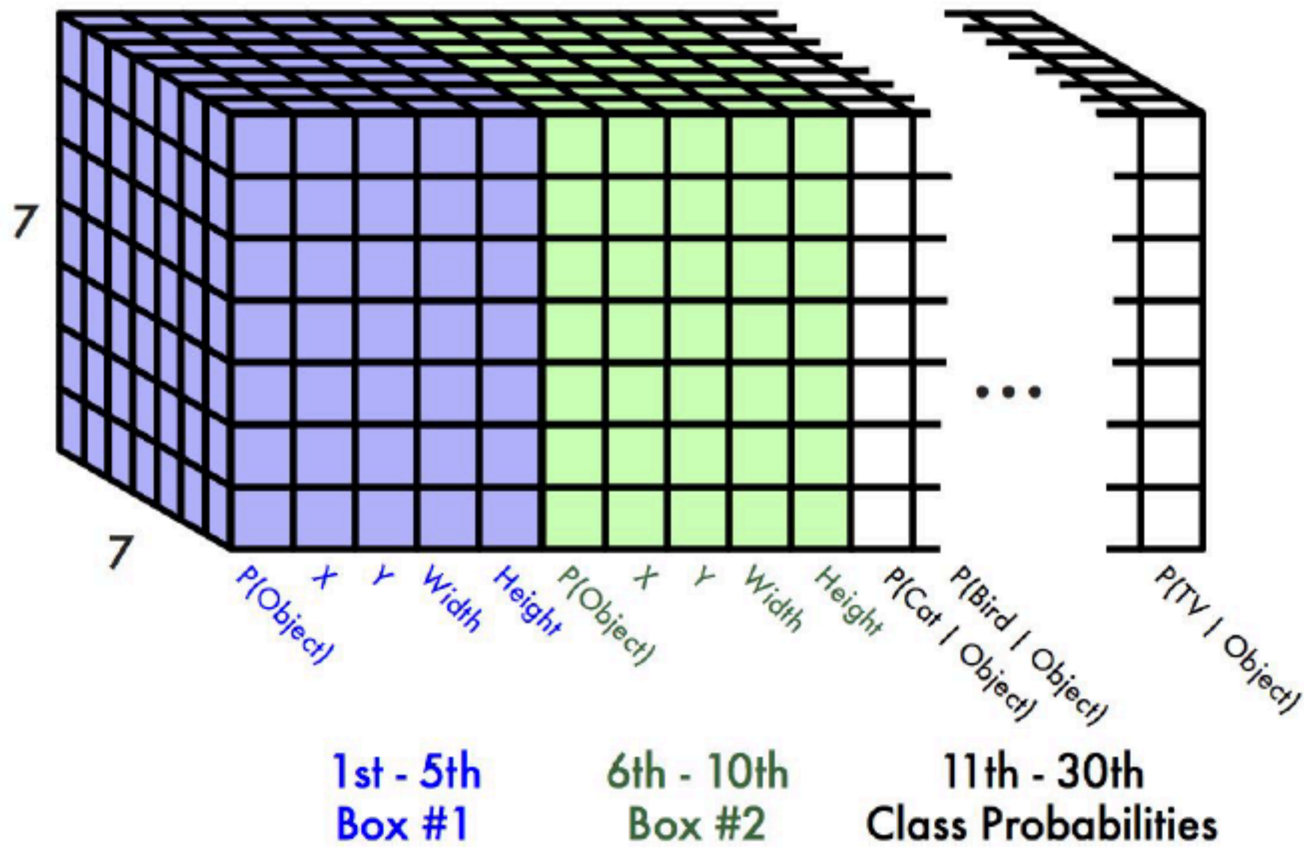
Finally we do NMS and threshold detections



This parameterization fixes the output size

Each cell predicts:

- For each bounding box:
  - 4 coordinates (x, y, w, h)
  - 1 confidence value
- Some number of class probabilities

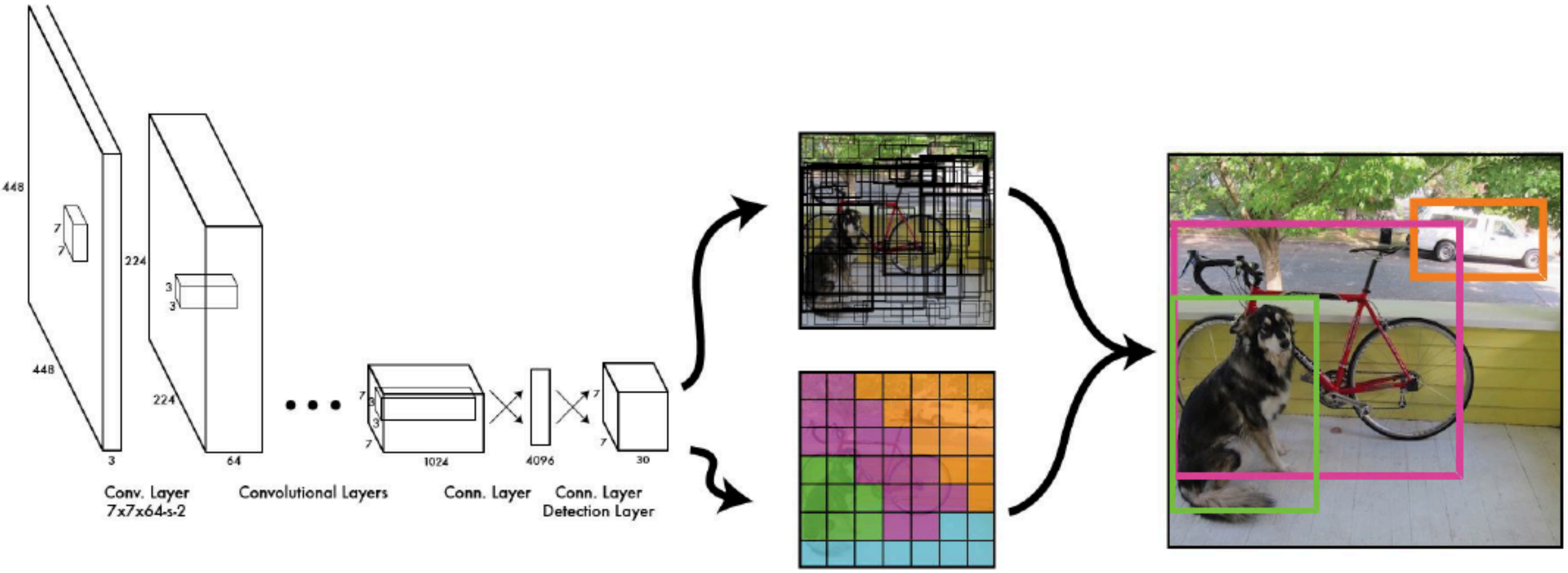


For Pascal VOC:

- 7x7 grid
- 2 bounding boxes / cell
- 20 classes

7 x 7 x (2 x 5 + 20) = 7 x 7 x 30 tensor = **1470 outputs**

Thus we can train one neural network to be a whole detection pipeline



	Pascal 2007 mAP	**Speed**	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	~~63.4~~ 69.0	45 FPS	22 ms/img

Evaluating detectors

- Compare detected boxes w ground truth boxes
- Favor
    - right number of boxes with right label in right place
- Penalize
    - awful lot of boxes
    - multiple detections of the same thing
- Strategy
    - Detector makes a ranked list of boxes
    - GT is a list of boxes
    - Mark detector boxes with relevant/irrelevant
    - summarize lists

IOU

The boxes that the detector predicts are unlikely to match ground truth exactly, and we need some way of telling whether the boxes are good enough. The standard method for doing this is to test the **IoU** (Intersection over Union). Write B_g for the ground truth box and B_p for the predicted box. The IoU is

$$\mathrm{IoU}(B_p, B_g) = \frac{\mathrm{Area}(B_g \cap B_p)}{\mathrm{Area}(B_g \cup B_p)}.$$

Choose some threshold t. If $\mathrm{IoU}(B_p, B_g) > t$, then B_p could match the ground truth box B_g.

Usually, t=0.5; higher t on occasion, but this sets a quite demanding standard for localization

Preventing double dipping

The detector should be credited for producing a box that has a high score and matches a ground truth box. But the detetector should not be able to improve its score by predicting many boxes on top of a ground truth box. The standard way to handle the problem is to mark the overlapping box with highest score `relevant`. The procedure is:

- Choose a threshold t.
- Order $\mathcal{D}$ by the score of each box, and mark every element of $\mathcal{D}$ with `irrelevant`. Choose a threshold t.
- For each element of $\mathcal{D}$ in order of score, compare that box against all ground truth boxes. If any ground truth box has IoU $> t$, mark the detector box `relevant` and remove that ground truth box from $\mathcal{G}$. Proceed until there are no more ground truth boxes.

Now every box in $\mathcal{D}$ is tagged either `relevant` or `irrelevant`.

Recall and Precision

There are standard evaluations for search results like those produced by our detector. The first step is to merge the lists for each evaluation image into a single list of results. The **precision** of a set of search results $\mathcal{S}$ is given by

$$\mathbf{P}(\mathcal{S}) = \frac{\text{number of relevant search results}}{\text{total number of search results}}.$$

The **recall** is given by

$$\mathbf{R}(\mathcal{S}) = \frac{\text{number of relevant search results}}{\text{total number of relevant items in collection}}.$$

As you move down the list $\mathcal{D}$ in order of score, you get a new set of search results. The recall never decreases as the set gets larger, and so you could plot the precision as a function of recall (write $\mathbf{P}(\mathbf{R})$). These plots have a characteristic saw-tooth structure (Figure 18.9). If you add a single irrelevant item to the set of results, the precision will fall; if you then add a relevant item, it jumps up. The sawtooth

Interpolated precision

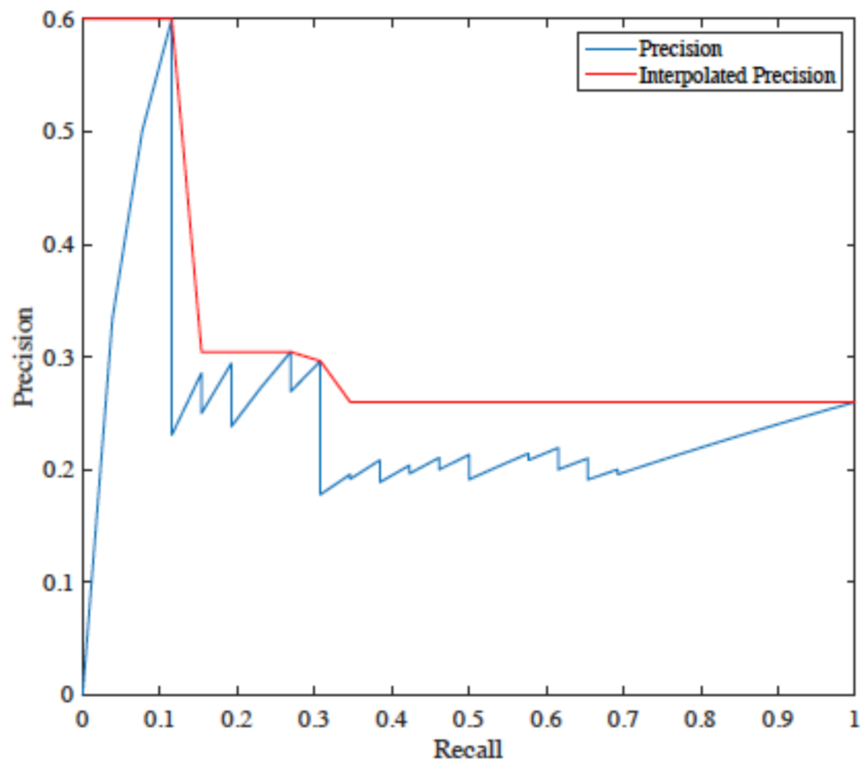


FIGURE 18.9: *Two plots for an imaginary search process. The precision plotted against recall shows a characteristic sawtooth shape. Interpolated precision measures the best precision you can get by increasing the recall, and so smoothes the plot. Interpolated precision is also a more natural representation of what one wants from search results – most people would be willing to add items to get higher precision. Interpolated precision is used to evaluate detectors.*

Interpolated precision

the precision will fall; if you then add a relevant item, it jumps up. The sawtooth doesn't really reflect how useful the set of results is — people are usually willing to add several items to a set of search results to improve the precision — and so it is better to use **interpolated precision.** The interpolated precision at some recall value R_0 is given by

$$\hat{\mathcal{P}}(R_0) = \max_{R \geq R_0} \mathbf{P}(R)$$

mAP

(Figure 18.9). By convention, the **average precision** is computed as

$$\frac{1}{11}\sum_{i=0}^{10}\hat{\mathcal{P}}(\frac{i}{10}).$$

This value summarizes the recall-precision curve. Notice this averages in interpolated precision at high recall. Doing so means a detector cannot get a high score by producing only very few, very accurate boxes — to do well, a detector should have high precision even when it is forced to predict every box.

Average precision evaluates detection for one category of object. The **mean average precision** (mAP) is the mean of the average precision for each category. The value depends on the IoU threshold chosen. One convention is to report mAP at $IoU = 0.5$. Another is to compute mAP at a set of 10 IoU values ($0.45 + i \times 0.05$ for $i \in 1 \ldots 10$), then average the mAP's. These evaluations produce numbers that tend to be bigger for better detectors, but it takes some practice to have a clear sense of what an improvement in mAP actually means.