Pedestrian Detection

Carlo Tomasi

September 27, 2017

A program that detects people in images has a multitude of potential applications, including tracking for biomedical applications or surveillance, activity recognition for person-device interfaces (device control, video games), organizing personal picture collections, and much more. However, detecting people is difficult, as the appearance of a person can vary enormously because of changes in viewpoint or lighting, clothing style, body pose, individual traits, occlusion, and more. It then makes sense that the first people detectors were really detectors of pedestrians, that is, people walking at a measured pace on a sidewalk, and viewed from a fixed camera. Pedestrians are nearly always upright, their arms are mostly held along the body, and proper camera placement relative to pedestrian traffic can virtually ensure a view from the front or from behind (Figure 1). These factors reduce variation of appearance, although clothing, illumination, background, occlusions, and somewhat limited variations of pose still present very significant challenges.

Figure 1: Images of pedestrians [2].

The prototypical pedestrian detector places a standard-sized window at all positions and at all levels of a Gaussian pyramid constructed on the input image, computes some feature vector in the window, and runs a classifier on the vector. In one instance proposed by Dalal and Triggs [2], the window is 64×128 (in portrait mode), the features are Histograms of Oriented Gradients (HOG), and the classifier is a Support Vector Machine (SVM) [1]. This note examines a few technical aspects and recent extensions of pedestrian detection.

1 Non-Maximum Suppression

If window W is centered on a pedestrian and the classifier returns the correct answer, then windows that significantly overlap with W are also likely to be classified as pedestrians. *Non-Maximum Suppression (NMS)* eliminates all but one of these detections. In its simplest form, NMS picks the window with the highest score $p(y|\mathbf{x})$ in the image as a true detection, then eliminates all the windows that overlap with the winner. The procedure is then repeated on the remaining windows until no windows remain unexamined.

If two people are so close to each other in the image that their windows overlap, this greedy procedure will miss to report one of them. Because of this, one often introduces a measure of difference between windows—for instance, the Euclidean distance between their two feature vectors—and suppresses overlapping windows only if the difference is large enough. More principled and global NMS methods have been proposed [7].

2 Training and Performance

The Dalal and Triggs detector [2] uses a Support Vector Machine (SVM) to distinguish pedestrians from non-pedestrians. What classifier is being used is relatively unimportant, and you can easily think of using a random forest of decision trees instead, for comparable performance. Dalal and Triggs's SVM is trained on a set of 1239 manually cropped and labeled images of pedestrians—together with their left-right reflections—as positive examples.

For negative examples, images of other than people are of course very easy to obtain. However, some of them are so obviously not people that they are only moderately useful for classification. To understand this, think of a binary classifier abstractly as a hyper-surface in feature space: any feature on one side of the surface is categorized as a positive feature, that is, a feature from the set to be recognized (pedestrian), and any feature on the other side is classified as negative. A good training set will have examples that are very close to the hyper-surface, so that they determine it tightly. Random patches out of person-free images are generally not very likely to be close to the surface: To be so, they would have to look very much like people but not be people.

To address this problem, negative examples are initially chosen as a set of 12180 patches sampled at random from 1218 person-free photographs, and a preliminary detector is trained on these patches together with the 1239 positive ones. That detector is then run on all windows in Gaussian pyramids built on top of each of the person-free photos. The false positives found in this run are considered to be "hard examples" and are added to the training set as negative training samples. The detector is then retrained on the augmented set to produce the final classifier. This data mining technique has been shown to improve performance significantly [2].

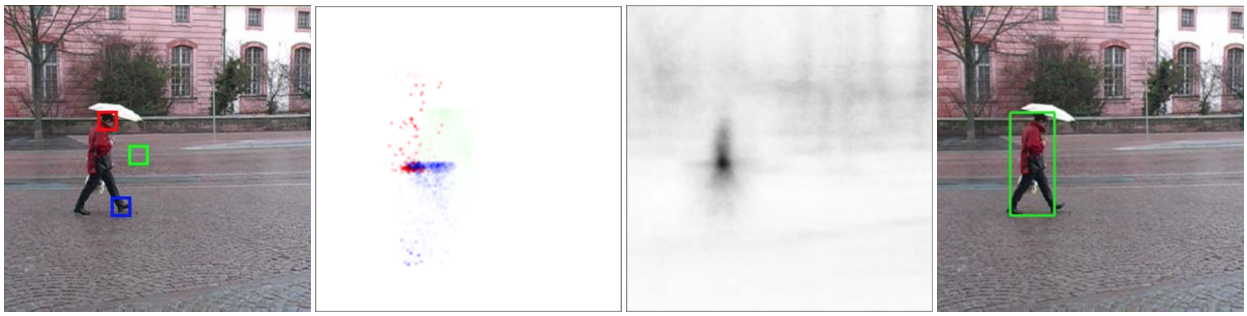
Overall, the authors report an 11 percent miss rate (false negatives) for a false positive rate per window of 10^{-4} on a small test set of 640×480 images. To understand what these figures mean, consider that a Gaussian pyramid with a sampling factor of $s = 1.2$ has $L = 8$ levels greater than 128×64 pixels, and therefore has a total of

$$\frac{1 - s^{-2L}}{1 - s^{-2}} \approx 3.1$$

times as many pixels as there are in the original image. So there are about $(640 - 64 + 1) \times (480 - 128 + 1) \times 3.1/8^2 \approx 9860$ possible window positions in the pyramid if a window stride of 8 pixels is used. A false positive rate of 10^{-4} per window then means that about one false positive occurs on average in each image. In other words, a reasonably small false-positive rate per image requires a very small false-positive rate per window. If that rate is achieved, then about 11 percent of the pedestrians in the performance evaluation database are not detected.

3 Hough Forests

The Dalal and Triggs detector evaluates the appearance within a window as a whole: Either the whole window looks like a pedestrian centered in it, or it does not. A part-based method, on the other hand, learns



(a) – Original image with three sample patches emphasized

(b) – Votes from the forest leaves visited by the three patches in (a)

(c) – Votes from the leaves visited by *all* the patches from (a)

(d) – The detection hypothesis corresponding to the peak in *(c)*

Figure 2: The three patches emphasized in *(a)* contribute votes about the possible location of a pedestrian-window centroid to a number of leaves in the Hough forest. Figure *(b)* shows the votes from these leaves (each color channel corresponds to leaves associated to each of the three patches). These votes derive from all patches in the training set that were classified into those leaves. Note the weakness of the vote from the patch in the background (green). After the votes from the leaves visited by all the patches in the image (not just the three that are emphasized in (a)) are aggregated into a Hough image *(c)*, the pedestrian can be detected *(d)* as a peak in this image.
[Figure from Gall and Lempitsky [5]. Caption adapted from the same article.]

to recognize *parts* of a pedestrian (head, shoulders, legs, feet, ...) separately, and infers the presence of a pedestrian as the result on some consensus among part detectors. A part-based detector has the potential to be less sensitive to occlusions, changes in body configuration, and variations due to viewpoint, if the way to reach consensus among detectors is flexible enough.

Hough forests are one such part-based detection scheme [5, 6]. Just as before, the *result* of detection is a set of positions for fixed-size windows within an image pyramid that are deemed to contain a pedestrian[1]. However, these window positions are determined by sliding a smaller (16×16 pixels) *patch* over the pyramid, and some feature $\mathbf{x}$ is computed for each patch. This feature could be a single histogram of oriented gradients, or other information such as color, texture, or even the raw image intensities. A patch classifier is trained to detect parts of the object of interest and to vote for the centroid of a window that might contain the whole object. At test time, image locations that receive many votes are returned. Figure 2 illustrates the idea. This idea is now fleshed out by describing how the Hough forest is trained and tested.

3.1 Training Hough Forests

During training, a binary random-forest classifier with M trees ($M = 15$) is learned to classify patch features as positive if they come from a window containing an instance of the object of interest, and negative if they do not. So "recognizing a part" means just this. The classifier does not distinguish between a head and a shoulder, or any other body part. A node in the tree is a leaf when its depth is 15 or when it has information from fewer than 20 training patches. As usual, each leaf λ contains the probability $p_\lambda = p(1|\lambda)$ that a patch that lands on λ has a positive label. Obviously, $p(0|\lambda) = 1 - p_\lambda$, and the single number p_λ encodes a Bernoulli distribution $p(y|\lambda)$ that is empirically estimated on the training data. For instance, $p_\lambda = 1$ means that all the training samples that landed on λ belonged to some window containing the object of interest. In addition, in a Hough forest, the leaf λ also contains a list D_λ of two-dimensional *displacement vectors*, one for each positive patch in the training set that is classified into leaf λ. Specifically, if a training patch was

[1] Or some other object. The original paper shows results for pedestrians, cars, and horses.

found at image position $\mathbf{p}$ within a window centered at (known) image position $\mathbf{w}$, then the displacement vector

$$\mathbf{d} = \mathbf{p} - \mathbf{w}$$

is stored in D_λ. If all patches in λ are negative, the list D_λ is empty. As an example, the red dots in Figure 2 (b) are the centroid vectors $\mathbf{w}$ found in the forest leaves into which the patch in the red rectangle in Figure 2 (a) was classified. The next section traces the origin of these red dots in some detail, and the section thereafter resumes with a description of more training details. You may want to open a copy of this document, so you can look at the relevant figures while you read the next section.

3.1.1 Life of Pi

Each training image contributes a large number of patches to the training set T. Each patch in T also comes with a label, which is equal to 1 if the patch overlaps a window that contains a pedestrian[2], and 0 otherwise. When T is built, the centroid $\mathbf{w}$ of the pedestrian window that overlaps a patch at $\mathbf{p}$ is known ($\mathbf{w}$ is around the person's belly button), and so is, of course, the centroid $\mathbf{p}$ of the patch. The displacement vector $\mathbf{d} = \mathbf{p} - \mathbf{w}$ for that patch/window pairing is then entered into the set D_λ for each of the 15 leaves (one per tree) that the patch falls into.

Figure 2 (b) was created after the Hough forest was trained (more on training later). To make that figure, the authors selected the red patch π (and two more) from the test image in Figure 2 (a) as examples. When this image is run through the pedestrian detector, all its patches are also run through each of the 15 trees in the random forest. In particular, patch π also ends up into 15 leaves, one per tree. Each leaf contains a number of displacements, each of which comes from some patch in some training image that overlaps some window with a pedestrian in it. Each of the red dots in Figure 2 (b) is drawn at position

$$\mathbf{w} = \mathbf{p}_\pi - \mathbf{d}$$

where $\mathbf{p}_\pi$ is the centroid of patch π and $\mathbf{d}$ varies over all displacements from the 15 leaves π ended up in.

Nobody knows exactly what training image, training patch, or pedestrian window those red dots come from, because that information is not stored in the random forest. However, we can try and guess as follows.

The training algorithm does not "know" that π is on top of a person's head. However, π does look like a head, and there are likely many pedestrians in some of the training images whose head looks somewhat similar to π. At training time, the patches from these heads are likely to have fallen in at least some of the 15 leaves that saw π. Because these patches are head patches, their displacement vector points to someone's belly-button $\mathbf{w}$, which for typical people is about three head sizes below the head. Therefore, the displacement vectors for all these people are not too different from each other. In addition, since π is actually on top of a head, the dot positions $\mathbf{w}$ cluster around the person's belly button in Figure 2 (a). Hence the fuzz of red dots at navel height in Figure 2 (b).

Let us reflect on the rather clever idea that this cloud of red dots comes from: The training algorithm cannot distinguish heads from other body parts, and yet patches on heads "know" where to find the person's belly button, so they can agree on where to put their votes. The reason why they agree is that (i) different bodies have similar shapes and sizes (so this idea would not work for, say, dancers, who can assume very diverse body shapes) and (ii) patches and person detectors have fixed sizes and are found in image pyramids. As a consequence of (ii), two "head" patches look similar to each other only when they are on two pyramid levels such that the two heads fit is a similar way in the two patches. Thus, pyramid processing accounts for

[2]Recall that these windows are determined manually by a person when the training set is built.

correct scaling, and the position of the belly button is then in a consistent place *relative* to the size of the patch.

There is a second set of red dots in Figure 2 (b) that require explanation. These are the sparsely distributed dots in the upper half of the figure, which all voted for "the wrong position." These dots likely come from patches in some training image that overlap some pedestrian window (so they have a label of 1). These patches look enough like heads that they fall in one of the 15 leaves for π, *but they are not heads*. While they do contribute displacements to those leaves, their displacements point to the centroid of their overlapping window, and that displacement is not a head-to-belly-button displacement. These "wrong" displacements are random and uncorrelated, and the cloud of the resulting red dots is therefore sparse.

3.1.2 More About Training

Hough forests use random decision forests in a rather unconventional way, because of their inclusion of patch-to-window displacements. These forests are only incidentally classifiers, in that they classify patches into whether they do or do not overlap a pedestrian window. The main role of the forest is instead to create the images of votes for pedestrian centroids, and how these votes are then used will be explained in the next section. Still, the method used for training the forest is rather conventional, with a few twists explained next.

The split rule at each node τ in the random forest picks two pixels at predetermined positions in the patch and compares the difference in their values to a threshold:

$$\begin{cases} \tau.L & \text{if } I(\tau.\mathbf{a}) - I(\tau.\mathbf{b}) < \tau.t \\ \tau.R & \text{otherwise} \end{cases} .$$

More specifically, the five scalar parameters in $\tau.\mathbf{a}$, $\tau.\mathbf{b}$, $\tau.t$ are chosen as follows. A list of 20,000 random pairs of points are chosen ahead of training, with coordinates drawn uniformly at random from the set of valid patch coordinates. During training, a point pair is chosen at random out of this list for each node, and the threshold $\tau.t$ is computed that leads to the greatest reduction of impurity, defined as either class-label uncertainty

$$kH(c) \quad \text{where} \quad H(c) = -c \log c - (1-c) \log(1-c)$$

or offset uncertainty

$$\sum_{j:y_j=1} (\mathbf{d}_j - \mathbf{m}_\mathbf{d})^T (\mathbf{d}_j - \mathbf{m}_\mathbf{d}) \; .$$

In these expressions, k is the number of patches at the node, the constant

$$c = \mathbb{E}[y|\lambda] = p_\lambda$$

is the mean label value among all patches at the node (label 1 means "positive" and label 0 means "negative"), and $\mathbf{m}_\mathbf{d}$ is the mean displacement for the patches at the node. Thus, class-label uncertainty grows with the number of patches and the entropy $H(c)$ (see Figure 3) of the label distribution: There is no uncertainty when $c = 0$ or $c = 1$, and the maximum uncertainty is when $c = 1/2$. The notation "$j : y_j = 1$" means "sum over displacements for patches with positive label," so that the offset uncertainty is proportional to the squared spread of the displacements around their mean.

A random coin flip determines which measure of impurity to use for each node, with the exception that offset uncertainty is always used when the fraction of positive patches at the node is at least $c = 0.95$. Interleaving impurity measures leads to relatively pure label distributions and a low variance in the displacements at each leaf. Using only offset uncertainty for nearly-pure positive nodes tightens the spread of their displacements, when reducing class label uncertainty would have little effect.

3.2 Hough-Forest Object Detection

The procedure for detecting object window candidates at test time can be described as follows. A patch is slid over every image in a Gaussian pyramid with scale factor ϕ (with $0 < \phi < 1$). A *vote pyramid* $\{V_\ell(\mathbf{x}) \mid \ell = 1, \ldots, L\}$ of the same size as the input pyramid is initialized to zero, and accumulates votes cast by every patch. Specifically, suppose that a patch centered at $\mathbf{p}$ at level ℓ of the pyramid is classified into leaf λ_m in tree number m, for $m = 1, \ldots, M$. The probability that the patch at $\mathbf{p}$ has a positive label given that it lands in leaf λ_m is p_{λ_m}, and the leaf contains a list

$$D_{\lambda_m} = \{\mathbf{d}_{m1}, \ldots, \mathbf{d}_{m|D_{\lambda_m}|}\}$$

of displacements. The leaf casts a vote p_{λ_m} which is equally spread into $|D_{\lambda_m}|$ displacement votes, one vote per vector in D_{λ_m}. As a result, displacement $\mathbf{d}_{mj}$ casts a vote

$$\frac{p_{\lambda_m}}{|D_{\lambda_m}|} \quad \text{for position} \quad \mathbf{p} - \mathbf{d}_{mj} \quad \text{of the vote image} \quad V_\ell(\mathbf{x}) \ .$$

The resulting votes suffer from quantization problems similar to the bin quantization problems we encountered in connection with HOG features: Small changes in either $\mathbf{p}$ or $\mathbf{d}_{mj}$ can cause a vote to land in a different pixel in the vote image V_ℓ. To address this problem, each vote image is smoothed by convolution with a Gaussian of fixed factor σ. This solution is in concept similar to voting by bilinear interpolation, but different in the shape of the smoothing kernel. The original paper [5] uses $\sigma = 3$ pixels.

The result of this computation is a pyramid of vote images $V_\ell(\mathbf{x})$. One can view these images as samples of a vote function $\mathcal{V}(\mathbf{y}, s)$ that is continuous in both position $\mathbf{y}$ (measured in the original, full-resolution image coordinates) and scale s. A mode-seeking algorithm called *mean shift* [4], described in Appendix A, can be used to find the local maxima of $\mathcal{V}$. While mean-shift was first developed for probability distributions, it can also be used with other nonnegative functions. The resulting triples of the form $(\mathbf{y}^*, s^*, \mathcal{V}(\mathbf{y}^*, s^*))$ indicate where in space ($\mathbf{y}^*$) and scale (s^*) the maxima occur and their confidence (that is, supporting vote) $\mathcal{V}(\mathbf{y}^*, s^*)$. A confidence threshold $\mathcal{V}_0$ tuned by cross-validation can be used to select the windows to return as the result of detection. There is no need for non-maximum suppression, because the mean-shift algorithm returns one point per local maximum.

The notion of parts voting for whole objects is called *Hough voting*, by reference to an analogous technique originally used to find lines and circles in images [3]. The original papers on Hough forests [5, 6] derive the quantitative aspects of Hough voting from probabilistic considerations. However, these are based on arguable assumptions, and it is not clear that the probabilistic viewpoint adds useful insights.

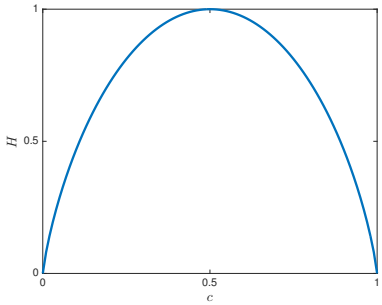


Figure 3: The entropy function $H(c) = -c \log c - (1 - c) \log(1 - c)$.

3.3 Performance of Hough-Forest Object Detection

Object detection with random forests was tested on several benchmark databases and resulted in Recall-Precision Equal-Error Rates (RPEER) between 94.4 and 98.6 percent [5, 6]. The significance of this performance measure is as follows. Let *precision* be defined as the fraction of the reported detections that are correct, and *recall* as the fraction of all true detections that are reported. Then, low values of the confidence threshold $\mathcal{V}_0$ lead to many positives, both true and false, resulting in low precision and high recall. As the threshold increases, recall decreases because some of the true positives go undetected, and precision increases as some of the false positives disappear. For some value $\mathcal{V}_0^*$, precision and recall are equal to each other. The RPEER is the common value that precision and recall achieve at $\mathcal{V}_0^*$. In other words, if precision and recall are considered equally important and therefore required to be equal to each other, the best value that can be achieved for them is between 94.4 and 98.6 percent on the benchmarks that were used in the experiments. More details on various performance measures for retrieval and detection systems are given in Appendix B.

References

[1] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines*. Cambridge University Press, Cambridge, UK, 2000.

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, June 2005.

[3] R. O. Duda and P. E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15:11–15, 1972.

[4] K. Fukunaga and L. D.Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

[5] J. Gall and V. Lempitsky. Class-specific Hough forests for object detection. In *IEEE Conference on Computer vision and Pattern Recognition*, pages 143–157, 2009.

[6] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky. Hough forests for object detection, tracking, and action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(11):2188–2202, 2011.

[7] R. Rothe, M. Guillaumin, and L. van Gool. Non-maximum suppression for object detection by passing messages between windows. In *Asian Conference on Computer Vision*, pages 290–306, 2014.

Appendices

A The Mean Shift Algorithm

The mean shift algorithm [4] finds the mode of a probability function $p(\mathbf{z})$ given a set $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ of data points in $\mathbb{R}^d$ that are assumed to be drawn from $p(\mathbf{z})$. Weights can also be associated to the points, with the understanding that a point $\mathbf{z}_n$ with weight w_n represents a probability density[3] w_n at $\mathbf{z}_n$.

Let $K_h(\mathbf{z})$ be a Gaussian *kernel* defined as follows:

$$K_h(\mathbf{z}) = e^{-\left(\frac{\|\mathbf{z}\|}{h}\right)^2}$$

where $h > 0$ is called the *bandwidth* of the kernel ($K_h(\mathbf{z})$ is not a probability density, as it is not normalized to integrate to one). The quantity

$$\phi_h(\mathbf{z}) = \sum_{n=1}^{N} w_n K_h(\mathbf{z} - \mathbf{z}_n) \ , \tag{1}$$

is a measure of the local density of the data in a neighborhood of $\mathbf{z}$: If there are many data points $\mathbf{z}_n$ near $\mathbf{z}$, then the value of $\phi_h(\mathbf{z})$ is large. The vector

$$\boldsymbol{\mu}_h(\mathbf{z}) = \frac{\sum_{n=1}^{N} \mathbf{z}_n w_n K_h(\mathbf{z} - \mathbf{z}_n)}{\sum_{n=1}^{N} w_n K_h(\mathbf{z} - \mathbf{z}_n)} \tag{2}$$

is an average of the data $\mathbf{z}_n$ weighted by both the weights w_n and a decreasing function of their distance from $\mathbf{z}$, and can therefore be interpreted as the local centroid (or mean) of the data around $\mathbf{z}$. If the data is locally Gaussian, the density at the centroid $\boldsymbol{\mu}_h(\mathbf{z})$ is no less than the density at $\mathbf{z}$. Figure 4 illustrates this point.

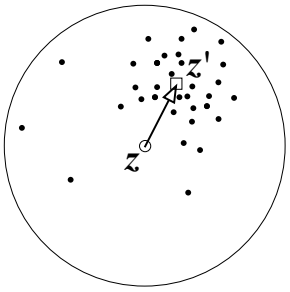


Figure 4: This drawing assumes $w_i = 1$ for simplicity. The large circle suggests a Gaussian function centered at $\mathbf{z}$ (small hollow circle). The distribution of points (black dots) under this Gaussian is lopsided, and the weighted mean $\mathbf{z}'$ (small hollow square) of the data does not coincide with $\mathbf{z}$. If $\mathbf{z}$ is moved (arrow) to point $\mathbf{z}'$, then the local density increases.

Equation (1) gives a way to measure the data density around any point $\mathbf{z} \in \mathbb{R}^d$ and equation (2) computes a new point $\mathbf{z}' = \boldsymbol{\mu}_h(\mathbf{z})$ where the density is equal to or greater than that at $\mathbf{z}$. These observations yield an

[3] Equivalently, but somewhat loosely, this means that if M points are drawn out of $p(\mathbf{z})$, then about $M w_n / \sum_k w_k$ land very close to $\mathbf{z}_n$.

astonishingly simple algorithm that seeks the mode of the density of the data in Z: Start anywhere (at some initial point $\mathbf{z}_{\text{start}}$) and keep shifting from $\mathbf{z}$ to the local mean $\boldsymbol{\mu}_h(\mathbf{z})$ (which becomes the new $\mathbf{z}$), until $\mathbf{z}$ and $\boldsymbol{\mu}_h(\mathbf{z})$ coincide. Of course, this algorithm is local, and which mode it finds depends on $\mathbf{z}_{\text{start}}$. To find all the modes, it is customary to run this search with

$$\mathbf{z}_{\text{start}} = \mathbf{z}_n$$

in turn for each n. Several starting points may lead to convergence to the same end point. A list of the distinct points these runs converge to is then returned. Algorithm 1 summarizes this procedure. In addition, what "local" means depends on the value of the bandwidth parameter h, which is therefore usually tuned by cross-validation.

When applied to the Hough vote image $\mathcal{V}(\mathbf{y}, s)$, one can think of each position $\mathbf{x}$ at level ℓ in the pyramid as a point $\mathbf{z}_n$ with weight

$$w_n = \mathcal{V}(\mathbf{x}, \ell) \ .$$

Algorithm 1. The mean shift algorithm

Input: $\mathbf{z}_1, \ldots, \mathbf{z}_N, h > 0, \epsilon > 0$ ▷ ϵ is a termination threshold

  for $k = 1, \ldots, N$ **do**

    $\mathbf{z}' \leftarrow \mathbf{z}_k$

    repeat

      $\mathbf{z} \leftarrow \mathbf{z}'$

      $\mathbf{z}' \leftarrow \frac{\sum_{n=1}^{N} \mathbf{z}_n w_n K_h(\mathbf{z}-\mathbf{z}_n)}{\sum_{n=1}^{N} w_n K_h(\mathbf{z}-\mathbf{z}_n)}$

    until $\|\mathbf{z} - \mathbf{z}'\| \leq \epsilon$ ▷ Stop if there is not enough improvement

    $\mathbf{y}_k \leftarrow \mathbf{z}'$

  end for

  return $\text{unique}(\{\mathbf{y}_1, \ldots, \mathbf{y}_N\})$ ▷ List of distinct elements in $\{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$

Output: $\mathbf{y}$

B Performance Measures for Retrieval and Detection Systems

Several different measures are used in the literature to quantify the performance of a retrieval or detection system. This Appendix introduces some of these measures in a unified way with the intent of emphasizing the connections between them.

Given a collection $\mathbb{C}$ of C items of information (web pages, images, video clips, documents, or other) and a predicate $\mathcal{R}(i)$ that takes an item $i \in \mathbb{C}$ as input and is true when the item i is *relevant*, a *retrieval system* returns all items $i \in \mathbb{C}$ for which it *estimates* $\mathcal{R}(i)$ to be true.

Given an image, a collection $\mathbb{C}$ of C windows in the image, and a predicate $\mathcal{R}(i)$ that takes a window $i \in \mathbb{C}$ as input and is true when the window i is *relevant*, a *detector* returns all windows $i \in \mathbb{C}$ for which it *estimates* $\mathcal{R}(i)$ to be true.

Thus, at this level of abstraction, detectors and retrieval systems are the same, and the term *system* will henceforth refer to either.

B.1 The Four Basic Sets of Items

Whether item i is really relevant or the system deems it to be so is of course a different matter. We define

$$\mathbb{R} = \{i \in \mathbb{C} \mid \mathcal{R}(i) \text{ is true}\} \quad \text{and} \quad R = |\mathbb{R}|$$

to be the set and number of (really) *relevant* items. We also define

$$\mathbb{P} = \{i \in \mathbb{C} \mid \mathcal{P}(i) \text{ is true}\} \quad \text{and} \quad P = |\mathbb{P}|$$

as the *positive* set (and number) of items that are returned or detected. Thus, $\mathcal{P}(i)$ means that the system estimates $\mathcal{R}(i)$ to be true, correctly or otherwise. The complements of these sets, $\neg\mathbb{R}$ and $\neg\mathbb{P}$, are the sets of *irrelevant* and *negative* items, with sizes

$$I = |\neg\mathbb{R}| \quad \text{and} \quad N = |\neg\mathbb{P}| \ .$$

The following four basic sets and sizes can be defined from $\mathbb{R}$ and $\mathbb{P}$ by set intersection '$A \cap B$' and set complement:

$$\begin{array}{rcrcl} \text{True positives} & : & \mathbb{R} \cap \ \mathbb{P} & \text{of size} & TP \\ \text{False negatives} & : & \mathbb{R} \cap \neg\mathbb{P} & \text{of size} & FN \\ \text{False positives} & : & \neg\mathbb{R} \cap \ \mathbb{P} & \text{of size} & FP \\ \text{True negatives} & : & \neg\mathbb{R} \cap \neg\mathbb{P} & \text{of size} & TN \ . \end{array}$$

In words, true positives are relevant items that are returned; false positives are irrelevant items that are returned; false negatives are relevant items that are not returned; and true negatives are irrelevant items that are not returned.

From these definitions, we have

$$R = TP + FN \quad , \quad I = FP + TN \quad , \quad P = TP + FP \quad , \quad N = FN + TN \ ,$$

and

$$R + I = P + N = C \quad \text{where} \quad C = |\mathbb{C}| \ .$$

The following table summarizes the sizes of these sets:

$\cap$	$\mathbb{R}$	$\neg\mathbb{R}$	
$\mathbb{P}$	TP	FP	P
$\neg\mathbb{P}$	FN	TN	N
	R	I	C

Values in the margins of this table are sums of the numbers in the respective rows and columns.

B.2 System Performance Measures

The performance of a system for a particular predicate is quantified by two numbers that describe the items in $\mathbb{P}$ that are either in excess or in defect relative to those in $\mathbb{R}$. The literature uses different pairs of numbers depending on context and tradition: (recall, precision), (false positive rate, true positive rate), or (specificity, sensitivity).

Recall, *true positive rate*, and *sensitivity* are different names for the same quantity

$$\rho = \frac{TP}{R} \ ,$$

equal to the fraction of relevant items that the system returns. A high value for this measure is desirable, as it indicates that the system misses few of the relevant items.

Precision measures the fraction of returned items that are relevant:

$$\pi = \frac{TP}{P} \ .$$

High precision is also desirable, as it entails a low fraction of junk, or irrelevant items, in the set the system returns. So does a high value of *specificity*, that is, of the fraction of irrelevant items that the system does not return:

$$\sigma = \frac{TN}{I} \ .$$

The *false positive rate* measures the fraction of irrelevant items that the system does return, that is, it measures the flip side of what precision and specificity measure:

$$\phi = \frac{FP}{I} = 1 - \sigma \ .$$

While specificity and false positive rate are exactly complementary to each other, precision and false positive rate are merely decreasing functions of each other.

B.3 Relationships between Different Measures

To determine the relationships between ρ, π, σ, and ϕ, we first write expressions for TP, TN, FP, and FN as functions of ρ, π, σ, ϕ, R, and I. Straightforward manipulation yields the following equalities:

$$\begin{aligned} TP &= \rho R \\ FN &= R(1-\rho) \\ FP &= \phi I = (1-\sigma)I = \rho R \frac{1-\pi}{\pi} \\ TN &= (1-\phi)I = \sigma I = I - \rho R(1-\pi) \ . \end{aligned}$$

Using the appropriate equalities in this group in the definitions of π, σ, and ϕ yields the desired relationships:

$$\begin{aligned} \pi &= \frac{TP}{TP+FP} = \frac{\rho}{\rho+(1-\sigma)\nu} = \frac{\rho}{\rho+\phi\nu} \\ \sigma &= \frac{TN}{TN+FP} = 1-\frac{\rho}{\nu}\frac{1-\pi}{\pi} = 1-\phi \\ \phi &= \frac{FP}{FP+TN} = \frac{\rho}{\nu}\frac{1-\pi}{\pi} = 1-\sigma\ , \end{aligned} \tag{3}$$

where

$$\nu = \frac{I}{R}$$

is the ratio of the number of irrelevant over relevant items in the collection $\mathbb{C}$.

The precision π is bounded from below once recall ρ and the collection parameter ν are given. This is because a certain level of recall requires a sufficiently large number TP of true positives, and precision increases with TP as well. The resulting bound on π can be obtained from the constraint that TN is a nonnegative number:

$$0 \le TN = I - FP \quad \Rightarrow \quad 0 \le \frac{I}{R} - \frac{FP}{R} = \nu - \frac{\rho R\frac{1-\pi}{\pi}}{R} = \nu - \rho\frac{1-\pi}{\pi}$$

and solving for π yields

$$\pi \ge \frac{\rho}{\rho+\nu}\ . \tag{4}$$

No analogous bounds are needed for either σ or ϕ, because these quantities depend on negative returns, which can be set independently of the positive returns.

Figure 5 shows plots of the relationships among π, σ, and ϕ for specific values of ρ and ν. Non-identical relationships that involve π are nonlinear.

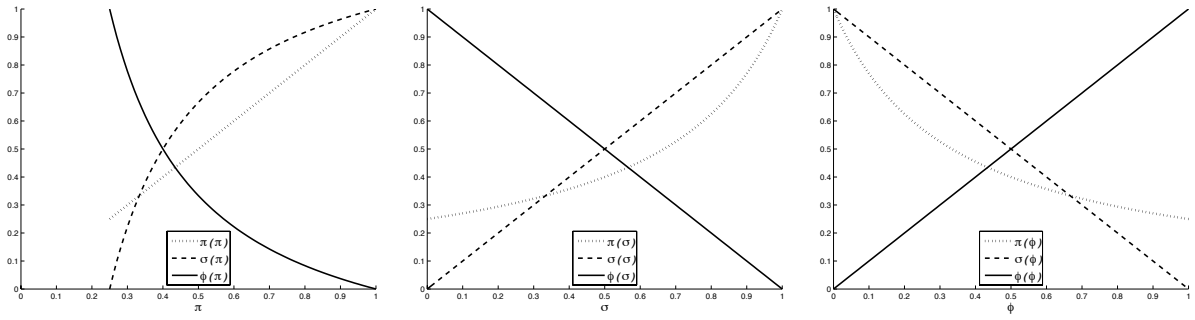


Figure 5: Plots of precision π, specificity σ, and false-positive rate ϕ versus each other for $\nu = 0.9$ and $\rho = 0.3$. The bound (4) causes the plots on the left to be undefined for $\pi < 0.25$. For values of precision lower than this, a recall $\rho = 0.3$ cannot be achieved when $\nu = 0.9$.

B.4 Trade-Off Curves

As mentioned in the previous Section, the performance of a retrieval or detection system is often evaluated in the literature by one of the following pairs:

- $(\rho, \pi) = (\frac{TP}{R}, \frac{TP}{P}) =$ (recall, precision)
- $(\phi, \rho) = (\frac{FP}{I}, \frac{TP}{R}) =$ (false positive rate, true positive rate)
- $(\sigma, \rho) = (\frac{TN}{I}, \frac{TP}{R}) =$ (sensitivity, specificity)

Given a collection $\mathbb{C}$ and a predicate $\mathcal{P}$ (the system's estimate of the true predicate $\mathcal{R}$), there is a trade-off between the two quantities in each pair above. For instance, recall can be improved by increasing P, the number of returned items. However, this improvement comes usually at the price of more irrelevant items being returned as well, resulting in a lower precision. This trade-off can be tuned by varying some parameter, for instance, a threshold t that determines whether an item does or does not satisfy $\mathcal{P}(i)$. Perhaps a low value of t corresponds to a liberal threshold, and causes many items to be returned, thereby yielding high recall and low precision. As t is increased, recall decreases and precision increases.

Three parametric curves can therefore be constructed:

$$(\rho(t), \pi(t)) \quad , \quad (\phi(t), \rho(t)) \quad , \quad (\sigma(t), \rho(t)) \ ,$$

with the first value forming the abscissa and the second the ordinate. These are called the *precision-recall* curve, the *Receiver-Operating-Characteristic* (or *ROC*) curve, and the *specificity-sensitivity* curve, respectively (the name of the ordinate appears first in these names, where applicable). Because of the relationships derived in the previous Section, these curves convey mutually equivalent information.

A different curve could be constructed for each predicate $\mathcal{P}$. Typically, however, a single curve is traced through statistical aggregates (means or medians) of the performance values over a large set of predicates, one aggregate pair per value of the parameter t.

For perfect systems, there is at least one value of t for which

$$(\rho(t), \pi(t)) = (1, 1) \quad , \quad (\phi(t), \rho(t)) = (0, 1) \quad , \quad (\sigma(t), \rho(t)) = (1, 1) \ .$$

Away from these points, it is most desirable for an ideal system to have an optimal value for either measure regardless of the value of the other. In other words, the perfect trade-off curve is a pair of straight segments—one horizontal and one vertical—as shown by the dashed lines in Figure 6.

For a system that draws items at random out of $\mathbb{C}$—thereby using no information about the data—the ratio TP/FP of relevant to irrelevant items in the returned set $\mathbb{P}$ is on average equal to the ratio R/I of relevant to irrelevant items in the whole collection $\mathbb{C}$. Because of this, $\rho = TP/R$ and $\phi = FP/I$ are equal for a random system, whose average performance is therefore on the identity line

$$\rho = \phi$$

in ROC space. Which point of the identity is achieved depends on the ratio between the size P of $\mathbb{P}$ and the size C of $\mathbb{C}$. The average recall value ρ is the ratio between number of relevant items in the returned set $\mathbb{P}$ and number of relevant items in the collection $\mathbb{C}$. For a random system, this ratio is on average equal to the ratio between P and C, the sizes of $\mathbb{P}$ and $\mathbb{C}$:

$$\rho = \frac{P}{C} \ .$$

Since $\sigma = 1 - \phi$, the same system has a specificity-sensitivity curve of the form

$$\rho = 1 - \sigma \ .$$

Setting $\rho = \phi$ in equation (3) shows that a random system has precision

$$\pi = \frac{1}{1+\nu} = \frac{R}{C} \,,$$

a value that is independent of the recall value ρ. This reflects the fact that the precision π is the fraction of relevant items in the returned set $\mathbb{P}$, and for a random system this fraction is on average equal to the fraction R/C of relevant items in the whole collection $\mathbb{C}$. For a fixed collection $\mathbb{C}$, the parameter t affects only P, the size of the set $\mathbb{P}$ of returned items, and therefore leaves recall unaffected. Because of this, the precision-recall curve for a random system is a horizontal line segment with ordinate R/C.

Figure 6 plots the three curves for an ideal and a random system.

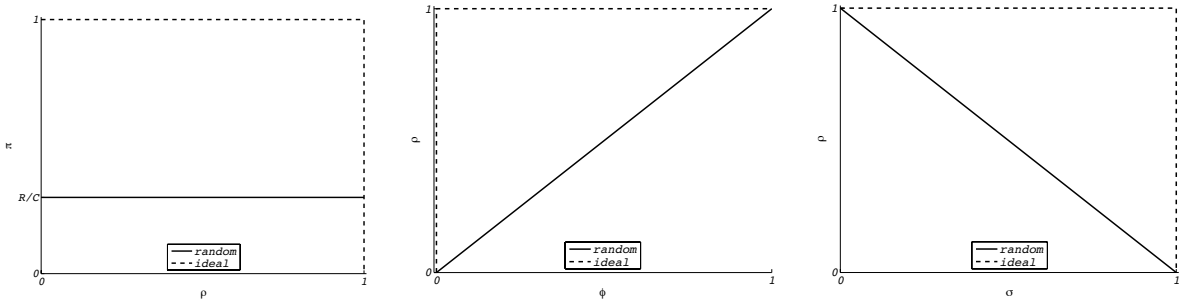


Figure 6: Precision-recall (left), ROC (center), and specificity-sensitivity (right) curves for an ideal (dashed) and a random (solid) system.

The area under the graph of any of the three curves for a perfect system is 1. Curves for actual systems are inside the convex hulls of the ideal curves, and their *Area Under the Curve* (AUC) is therefore less than 1. The value of this area is often taken as a measure of the quality of a retrieval or detection system that can be tuned through a threshold t. A random system has an AUC of 1/2 for ROC and specificity-sensitivity, and of R/C for precision-recall.

Another scalar measure of quality of a retrieval or detection system is the *Equal-Error Rate* (EER), defined as for the common value of the two quantities on the axes when they are constrained to be equally good. Thus, the EER for precision-recall and for specificity-sensitivity is the abscissa or ordinate (they are equal to each other) of the intersection of the curve with the identity line. For ROC, a good false-positive rate ϕ is a low rate, so the ROC-EER is the abscissa or ordinate of the intersection of the ROC curve with the line $\rho + \phi = 1$.