Fast multiple-part based object detection using KD-Ferns

Dan Levi Shai Silberstein Aharon Bar-Hillel
General Motors R&D, Advanced Technical Center - Israel
dan.levi@gm.com shai4998@netvision.net.il aharon.barhillel@gmail.com

Abstract

In this work we present a new part-based object detection algorithm with hundreds of parts performing real-time detection. Part-based models are currently state-of-the-art for object detection due to their ability to represent large appearance variations. However, due to their high computational demands such methods are limited to several parts only and are too slow for practical real-time implementation. Our algorithm is an accelerated version of the "Feature Synthesis" (FS) method [1], which uses multiple object parts for detection and is among state-of-the-art methods on human detection benchmarks, but also suffers from a high computational cost. The proposed Accelerated Feature Synthesis (AFS) uses several strategies for reducing the number of locations searched for each part. The first strategy uses a novel algorithm for approximate nearest neighbor search which we developed, termed "KD-Ferns", to compare each image location to only a subset of the model parts. Candidate part locations for a specific part are further reduced using spatial inhibition, and using an object-level "coarse-to-fine" strategy. In our empirical evaluation on pedestrian detection benchmarks, AFS maintains almost fully the accuracy performance of the original FS, while running more than $\times 4$ faster than existing part-based methods which use only several parts. AFS is to our best knowledge the first part-based object detection method achieving real-time running performance: nearly 10 frames per-second on 640×480 images on a regular CPU.

1. Introduction

Detecting objects of a particular class in a given image remains a difficult challenge for computer vision. Such a capability can support a wide range of real-world applications from aid to the blind to pedestrian detection for advanced driver assistance systems. Although the current performance is improving, as reflected on standard benchmarks like the PASCAL VOC challenge [9] and the Caltech pedestrian benchmark [7], it remains poor compared to that of human vision. Nevertheless, vision-based pedestrian detection technology in vehicles is already commercially available [20]. Due to the limited computation resources, such systems use *template-based* methods [18] and are therefore limited to detecting fully visible upright pedestrians. *Part-based* methods [12, 11, 1] use object parts with a deformable configuration to model objects, increasing their ability to cope with partial occlusions and large appearance variations compared with template-based methods. Furthermore, using a large number of parts with diverse appearances improves detection accuracy [1]. Evidently, part-based methods are highly ranked on large scale benchmarks [9, 7]. Such methods, however, are either limited in the number of parts modeled [11] to be able to run in reasonable time, or are impractical in terms of run time [1].

Previous work accelerating object detection mainly focus on *template-based* methods [21, 5, 2, 4]. From these approaches we adopt the well-studied sliding-window technique [21] with a "coarse-to-fine" strategy for early window elimination and location refinement [17]. Accelerating *part-based* detection mostly focused on methods relying on a small number of parts such as the Deformable Part-based Model (DPM) [11], since computation time increases linearly with the number of parts. In [8] properties of the Fourier transform are exploited to speed up the computation of linear filters such as those used in the DPM. The Cascaded Deformable Part-based Model [10] (c-DPM) uses a cascade of part detectors to accelerate the original DPM and is considered the fastest part-based method available, but is still limited in the number of parts and does not reach real-time performance. We present the Accelerated Feature Synthesis (AFS) algorithm, which is based on the Feature Synthesis (FS) [1], a part-based detection method which uses hundreds of parts in its object model. In our architecture, in each image location, only the closest parts are compared, and for each part, only locally maximal-appearance positions are used for classification. In contrast, existing part-based methods (e.g. DPM,c-DPM) consider *all* parts in a *dense* grid of positions.

The Feature Synthesis (FS) method [1] is a particularly flexible framework which uses hundreds of part based features selected from feature families with increasing complexity. The families of features encapsulate the appear-

ance and relative location of one or more object parts. The method was shown to be state-of-the-art on several human detection benchmarks [1], but suffers from a high-computational cost making it impractical for real-time applications. Our first contribution, the Accelerated Feature Synthesis (AFS), is a variant of the FS which proposes a combination of several speedup strategies making multiple-part based detection practical. Our second contribution, is the KD-Ferns, a novel algorithm for fast approximate nearest neighbors enabling a reduction in the number of searched parts in each image location. The AFS algorithm uses a coarse-to-fine strategy: first, a "coarse" part-based detector is used to eliminate most image regions and then, a "fine" such detector is used to detect the object in the remaining regions. To speed up the coarse level the KD-Ferns algorithm is used to compare only a small subset of the parts to each image location. In addition, for a specific part, only a *sparse* set of locations is considered using spatial inhibition. Finally, we modify the FS representation for object parts to allow sharing computation between the different parts. We evaluate the AFS on the pedestrian detection task using the INRIA pedestrians [3] and the Caltech pedestrian benchmark [7]. The detection accuracy loss compared to the FS is minor, and the AFS remains competitive with state-of-the-art methods. We compare the run time of the AFS with the methods evaluated on the Caltech pedestrian benchmark. The AFS is $\times 4.5$ faster than the part-based c-DPM [10], and is on par with the fastest *template-based* method for this benchmark, the "Fastest Pedestrian Detection in the West" (FPDW) [5].

Matching local image descriptors such as the SIFT [15] to a pre-stored database of descriptors is a fundamental problem in many computer vision algorithms, often facilitated by efficiently searching for nearest neighbors. The kd-tree algorithm [13] is a popular method for nearest neighbor search but quickly loses effectiveness in high dimensions. In such cases one must resort to finding Approximate Nearest Neighbors (ANN) in which a close enough neighbor is found with a high probability. ANN methods such as the randomized kd-trees [19] and hierarchical k-means tree [14] often rely on indexing the database points in a tree-structure, allowing only partial traversal of the database. Such methods often perform ANN search in sub-linear computation time in the number of examples, and are successfully applied to databases containing millions of examples. However, since visiting each tree node is associated with complex operations such as updating a priority queue [19], or a full dimensional distance computation [14], exhaustive search is in practice more efficient for small databases. The algorithm we propose, termed *KD-Ferns*, performs sub-linear runtime ANN search in practice for small databases of high-dimensional points. This is useful in particular for part based object detection in which we need to find the nearest "part descriptors" from a relatively small set of $O(100)$ parts in the model. Since this operation is done for almost every image location and in each image scale, efficiency is highly important. In the next section we present the KD-Ferns algorithm. In Section 3 we present the AFS method for object detection, the experimental evaluation in Section 4, and our conclusions in Section 5.

2. The "KD-Ferns" algorithm for approximate nearest neighbor search

Consider the exact nearest neighbor search problem: given a database of points $P \subset \mathbb{R}^k$ and a query vector $\mathbf{q} \in \mathbb{R}^k$ find $\arg\min_{\mathbf{p}\in P} \|\mathbf{q}-\mathbf{p}\|$. A popular search technique uses the KD-Tree data structure in which a balanced binary tree containing the database points as leaves is constructed. Each node specifies an index to its splitting dimension, $d \in \{1,\ldots,k\}$, and a threshold τ defining the splitting value. Given a query $\mathbf{q}$, (with $\mathbf{q}(d)$ denoting its d-th entry), the tree is traversed root to leaf by computing in each node the binary value of $\mathbf{q}(d) > \tau$ and following the right branch on 1 and left one on 0. Upon reaching a leaf dataset point, its distance to the query is computed and saved. In addition, each traversed node defined by d, τ is inserted to a priority queue (PQ) with a key which equals its distance to the query: $|\mathbf{q}(d) - \tau|$. After a leaf is reached the search continues by descending in the tree from the node with the minimal key in PQ. The search is stopped when the minimal key in PQ is larger than the minimal distance found, ensuring an exact nearest neighbor is returned.

A "KD-Fern" is a KD-Tree with the following property: all nodes in the same level (depth) of the tree have the same splitting dimension d and threshold τ. The search algorithm is identical to the one described for the KD-Tree but due to its restricted form can be implemented more efficiently. A KD-Fern with maximal depth L can be represented by an ordered list of dimension indexes and thresholds, $((d_1,\tau_1),\ldots,(d_L,\tau_L))$. As in the KD-Tree we insert each dataset point to a tree leaf. For a dataset point $\mathbf{p}$, $B(\mathbf{p})$ is a binary string defining its tree position. We now consider the inverse mapping M from binary strings of length $<= L$ to points in P. The domain of M can be transformed to all binary strings of length L by concatenating shorter strings with all possible suffixes and mapping them to the same point $\mathbf{p}$. Given a query $\mathbf{q}$ we create its binary string $B(\mathbf{q})$ by comparing it to each entry in the list: $B(\mathbf{q}) = ((q(d_1) > \tau_1),\ldots,(q(d_L) > \tau_L))$. $\mathbf{p} = M(B(\mathbf{q}))$ is then the dataset point in the leaf reached with query $\mathbf{q}$. For small enough dataset sizes $|P|$ the entire mapping can be stored in a memory-based lookup table with 2^L entries, and computing $M(B(\mathbf{q}))$ can be done in a single table access. The priority queue can also be efficiently implemented using bin-sorting due to the limited number of possible values, L. The downside is that a balanced tree

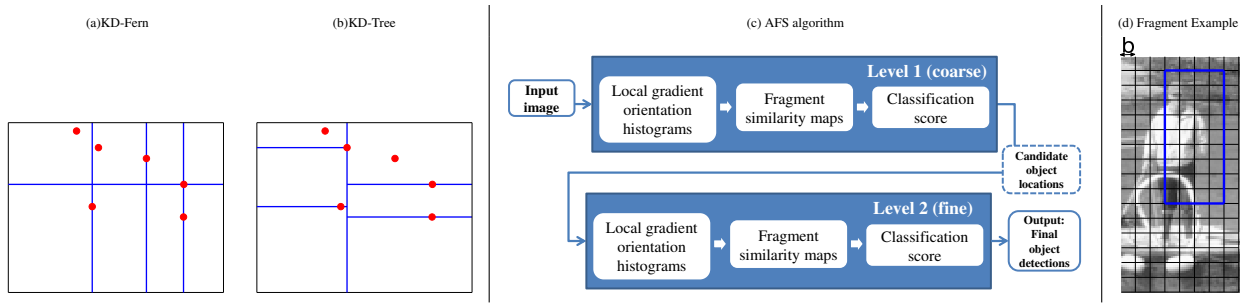

Figure 1. Space partition for 6 points in $2D$ using the **KD-Fern (a)** and the **KD-Tree (b)** construction algorithms. **(c) Accelerated Feature Synthesis (AFS) detection algorithm flow**. First level processes a full scale pyramid of the image while the second level processes only regions around candidate locations from level 1 and returns the final detections. **(d) Fragment example**. An example of a selected appearance fragment (blue rectangle) within the training image it was extracted from. The grid represents the spatial bins of size $b \times b$ used for computing the local gradient orientation histograms and the SIFT descriptor of the fragment.

with the KD-Fern property does not necessarily exist, and therefore the maximal depth L is no longer logarithmic in $|P|$. A new construction algorithm is therefore required.

The original KD-tree construction algorithm is applied recursively in each node splitting the dataset to the created branches. For a given node, the splitting dimension d with the highest variance is selected, and τ is set to the median value of $\mathbf{p}(d)$ for all dataset points in the node $\mathbf{p}$. The KD-Fern construction algorithm (Algorithm 1) sequentially chooses the d, τ for each level using a greedy strategy. In each level the splitting dimension is chosen to maximize the conditional variance averaged over all current nodes (line 1) for increasing discrimination. The splitting threshold is then chosen such that the resulting intermediate tree is as balanced as possible by maximizing the entropy measure of the distribution of dataset points after splitting (line 3(b)). Figure 1 shows the resulting data space partition obtained using the KD-Fern construction algorithm (a) for a toy set of six points in 2D, alongside the KD-Tree partition (b). KD-Ferns basically partitions the space to hyper-rectangles. In analogy to the randomized KD-trees [19], we extend our method to randomized KD-Ferns, in which several ferns are constructed randomly. Instead of choosing the splitting dimension d_l according to maximal average variance (line 1) a fixed number of dimensions K_d with maximal variance are considered, and d_l is chosen randomly among them. An approximate nearest neighbor is returned by limiting the number of visited leafs.

Algorithm 1 The KD-Fern construction algorithm

Input: A dataset, $P = \{\mathbf{p}_j\}_{j=1}^{N} \subset \mathbb{R}^n$. **Output**: $((d_1, \tau_1), \ldots, (d_L, \tau_L))$: An ordered set of splitting dimensions and thresholds, $d_l \in \{1 \ldots n\}, \tau_l \in \mathbb{R}$.

Initialization: $l = 0$ (root level). To each dataset point $\mathbf{p} \in P$, the l length binary string $B(\mathbf{p})$ represents the path to its current leaf position in the constructed binary tree. Initially, $\forall \mathbf{p}. B(\mathbf{p}) = \phi$.
Notations: $N_B(b) = |\{\mathbf{p}|B(\mathbf{p}) = b\}|$ is the # of points in the leaf with binary representation b. $\mathbf{p}(d) \in \mathbb{R}$ is entry d of point $\mathbf{p}$.
While $\exists \mathbf{p}, \mathbf{q}$ such that: $\mathbf{p} \neq \mathbf{q}$ and $B(\mathbf{p}) = B(\mathbf{q})$ **do**:

1. Choose the splitting dimension with maximal average variance over current leafs:

   $$d_{l+1} = \arg\max_d \sum_{b \in \{0,1\}^l} \frac{N_B(b)}{N} \cdot \mathbf{Var}_{\{\mathbf{p}|B(\mathbf{p})=b\}} [\mathbf{p}(d)]$$

2. Set $Max\_Entropy = 0$
3. For each $\tau \in \{\mathbf{p}(d_{l+1}) | \mathbf{p} \in P\}$
   (a) Set $\forall \{\mathbf{p} \in P\} : B'(\mathbf{p}) = [B(\mathbf{p}), \{\mathbf{p}(d_{l+1}) > \tau\}]$
   (b) Set $Entropy = -\sum_{b \in \{0,1\}^{l+1}} \frac{N_{B'}(b)}{N} \cdot \ln \frac{N_{B'}(b)}{N}$
   (c) if $(Entropy > Max\_Entropy)$:
   - Set $Max\_Entropy = Entropy$. Set $\tau_{l+1} = \tau$, Set $B = B'$.
4. $l = l + 1$

3. The Accelerated Feature Synthesis

The Accelerated Feature Synthesis (AFS) is a *sliding window* object detection method, based on the Feature Synthesis (FS) [1] method. We start by describing the FS method. In the FS, a part-based classifier model C discriminates sub-image windows I_s of fixed size $w_x \times w_y$ as tightly containing the object or not. C is trained using a sequential feature selection method and a linear-SVM classifier. C is parameterized by F, a set of classifier features, R, a set of rectangular image fragments extracted from training images, and $W = \{W_f\}$ the linear classifier weights. Computing $C(I_s) \in \mathbb{R}$, the *classification score* of sub-image I_s proceeds as follows. For each fragment $r \in R$ the "fragment similarity map" $a^r(x, y)$ represents the appearance similarity of r to each (x, y) position in I_s. $a^r(x, y)$ is computed as the inner-product between the 128-dimension SIFT descriptor [15] of r and that of the image fragment in position (x, y). Subsequent stages use a list of *spatially sparse* fragment detection locations $L^r = \{l^k = (x^k, y^k)\}_{k=1}^{K}$ computed by finding the $K = 5$ top local maxima in a^r. The appearance score of each location $l \in L^r$ is then $a^r(l)$. Each feature $f \in F$ is a func-

tion $f : I_s \mapsto \mathbb{R}$, computed using the fragment detections L^r of one or more fragments r. Each feature f represents different aspects of object-part detections. From the families of features suggested in [1], we use in the AFS only ones which significantly contribute to performance: **GlobalMax**, **Sigmoid**, **Localized**, **LDA** and **HoG component** features. For example, a **localized** feature is computed as: $f(I_s) = \max_{l \in L^r} \mathcal{G}(a^r(l)) \cdot \mathcal{N}(l; \mu_r, \sigma I_{2\times 2})$ where $\mathcal{N}$ is a 2D Gaussian function of the detection location l and $\mathcal{G}$, a learned sigmoid function on the appearance score. Such features represent location sensitive part detection, attaining a high value when both the appearance score is high and the position is close to a preferred part location μ_r, similar to parts in a star-like model [11]. For more details on computing the features please refer to [1]. The final classification score is a linear combination of the feature values: $C(I_s) = \sum_{f \in F}(W_f \cdot f(I_s))$. We next describe the AFS method for detecting objects in *full images* focusing on the major modifications relative to the FS.

Single fragment descriptor. The original FS uses image fragments $r \in R$ with different sizes and aspect ratios, all represented by a 128-dimensional SIFT (4×4 spatial bins and 8 orientation bins), and therefore the spatial bin size B_x, B_y is different for each fragment and equal to the fragment size $|r|_x, |r|_y$ divided by 4. In order to share the computation of local orientation gradient histograms between many fragments we use at most two different spatial bin sizes $B_{\{x,y\}} = b$ in our representation, but keep the different fragment sizes. For orientation we use $|ori| = 8$ orientation bins. We eliminate the spatial histogram smoothing from the original SIFT to speed up the computation. The result is for each fragment r a variable dimension descriptor $SIFT_b(r)$ with dimension $k(r) = \frac{|r|_x}{b} \cdot \frac{|r|_y}{b} \cdot 8$. An example of a selected fragment is illustrated in figure 1(d). We denote by $C = (F, R, W)$ a classifier model as defined previously with this modified fragment descriptor.

The input of the AFS is a full sized image Im and the output is the object detections represented by a set of bounding boxes at multiple locations and scales in the image and their classification scores. The AFS algorithm flow (see Figure 1(c)) is composed of a two-level *coarse-to-fine* cascade. The coarse level uses the sliding window methodology. It uses a trained coarse classifier $C_1 = (F_1, R_1, W_1)$ to compute the classification score for a dense set of sub-windows sampled in scale and position space. For a specific scale, sub-windows are sampled on a regular grid with a spatial stride $s = s_1$ pixels. Image locations which received a large enough score are then passed to the second level. Around each such location a local region is defined and sub-windows are sampled in that region on a finer grid with stride $s = s_2$ and processed by the second level with classifier $C_2 = (F_2, R_2, W_2)$ to produce the final classification score. At the end a standard non-maximal suppression (NMS) stage identical to the one described in [1] is used to locate the locally maximal detections. Computing the classification score for each sampled sub-window is similar for both cascade levels. We refer to this procedure as a *one-level detection* (blue rectangles in Figure 1(c)). The input to the first-level detection is the entire scale pyramid of Im and to the second level detection only the candidate image regions. We represented both types of input by a set of rectangular image regions $\{I\}$. Since each region is independently processed we describe the *one-level detection* for a single image region I ($|I| = n \times m$). Denote by $A = m \cdot n$ the area of I. Performing *one-level detection* using classifier model $C(F, R, W)$ is composed of three sequential stages that compute the following intermediate results: *local gradient orientation histograms*, *fragment similarity maps* and *classification scores*, as we describe next.

Local gradient orientation histograms. The first stage computes the image local gradient orientation histograms of I for spatial bins of size $b \times b$ corresponding to the bin size used to describe fragments $r \in R$. We first compute the gradient orientation and magnitude in each pixel. We then compute for each of the 8 orientations $\theta \in ori$ a map of orientation energy E_θ of size $n \times m$. A single computed gradient with orientation θ' contributes its magnitude to the two closest orientation bins weighted inversely by the distance from θ' to their centers as in the original SIFT. We then compute for each orientation energy map E_θ at each location on a grid with stride s, the energy sum in a spatial bin of size $b \times b$. Since this is a simple un-weighted rectangular summation it can be efficiently implemented using integral images. The output is a $(\frac{n}{s} \times \frac{m}{s} \times 8)$ hyper-image where each hyper-pixel is an 8 component *gradient orientation histogram* of the corresponding spatial bin. The time complexity of this stage is composed of the time it takes to compute the gradients and image integrals ($O(A)$), and the gradient histograms ($O(\frac{A}{s^2})$).

Fragment similarity maps. In this stage we compute for each fragment $r \in R$ with bin size b its similarity with the image in a dense set of locations. Given the position x, y in the image region I, the similarity is the dot product: $a^r(x, y) = SIFT_b(r) \cdot SIFT_b(I([x, x + |r|_x], [y, y + |r|_y]))$. We compute this measure for positions x, y sampled on a regular grid with stride s. For each fragment we pre-compute $SIFT_b(r)$. Computing $SIFT_b(I([x, x + |r|_x], [y, y + |r|_y]))$ is made efficient using the gradient orientation histograms for bin size b computed in the previous stage. It remains to get the pre-computed values for *bin centers* located in the rectangle corresponding to image positions $[x, x + |r|_x], [y, y + |r|_y]$ from each orientation map and concatenate them into one vector.

Denote by R^k the subset of fragments $r \in R$ with SIFT dimension k. The time complexity of this stage for all fragments $r \in R^k$ is $O(k \cdot |R^k| \cdot \frac{A}{s^2})$. We introduce a significant

speedup at the first-level detection by computing $a^r(x,y)$ for each image location (x,y) only for fragments r which are *the most similar* to that image location, setting the score for the rest to zero. This is possible using the following observation: since a feature is later computed using only several *local maxima* (L_r) of the fragment r similarity in the entire detection window, setting positions in which r is not maximal relative to *other fragments* to zero, will rarely change L_r (and the feature value). To find the most similar descriptors we search a KD-Ferns structure constructed in advance for all descriptors of $r \in R^k$, with $N << |R^k|$ trees. In each search we use the N (not necessarily unique) leafs returned by the KD-Ferns search, as the most similar descriptors to that particular image location. The complexity is then reduced to $O(k \cdot N \cdot \frac{A}{s^2})$.

Classification scores. We compute classification scores for each sub-window I_s of size $(w_x, \times w_y)$ in image I positioned on a grid with stride s. The features $f(I_s)$ and classification score $C(I_s)$ are computed as previously described. For each part-based feature f relying on appearance fragment r we compute L_r from the map a_r. We obtain a significant reduction of considered part locations by using only $|L_r| = K = 1$ locally maximal fragment detections per window instead of $K = 5$. This is a form of spatial *inhibition* in which the strongest fragment detection suppresses the nearby detections, producing a much sparser set of detections. The HoG-component features are not based on fragments and are fast to compute directly from the local gradient orientation histograms. An additional speedup is gained by using pre-stored lookup tables for computing the geometric score $N(l; \mu_r, \sigma I_{2\times 2}))$ of each location $l \in L_r$. The computation can then be accomplished in time $O(|R| \cdot \frac{A}{s^2})$ for obtaining local detection maxima and $O(|F| \cdot \frac{A}{s^2})$ for computing the feature and classifier score. In general $|R|$ and $|F|$ are of the same order, and the complexity is therefore $O(|F| \cdot \frac{A}{s^2})$.

Coarse and fine detection levels. Consider running *one-level detection*, applied to all scales (coarse level) or all regions (fine level), with a total pixel area of $\hat{A}$. By adding the time complexity for each of the three stages above we get: $O(\hat{A} + \hat{A} \cdot (\bar{k}N + |F|) \cdot \frac{1}{s^2})$ where $\bar{k} = \text{average}_{r\in R}(k(r)) \cdot |\text{vals}(k(r))|$. To make the first level faster we therefore use a larger stride s, shorter fragment descriptors $\bar{k}$ (by taking a larger spatial bin size b) and less features $|F|$ in the coarse classifier C_1. The result is a coarse (large s, b) first-level detection running at several orders of magnitude faster than the second-level detection, which uses a fine classifier C_2 with parameters set to reach the best classification accuracy. Each of the two classifiers C_1, C_2 used in the two corresponding detection levels is independently trained on a set of cropped positive and negative examples $\{I_s^{Train}\}$. To train each classifier we compute for each example $f(I_s^{Train})$ for a large set of candidate features $\{f\}$ using all stages above applied to a *single window*. We then use the exact same FS training procedure described in [1] to select the features and learn their weights W.

4. Experimental Results

To qualitatively evaluate the proposed object detection method we chose the pedestrian detection task due to the high availability of benchmarks and tested methods [7, 3] and due to the practical need for real-time detection. The AFS pedestrian detector used throughout the following experiments was trained on the INRIA pedestrians dataset [3].

Fine Classifier. We trained the second-level fine classifier C_2 as following. An initial fragment pool consisting of $40,000$ fragments was used, in sizes ranging from 8×8 to 80×32 pixels. Half of the fragments (the smaller ones) were represented using spatial bin size $b = 4$ pixels and the other half using $b = 8$. The stride for detection was $s_2 = 4$ pixels. In the training stage a total number of 500 features of the different families were selected. We refer to this classifier as `AccFeatSynth_L2`.

Coarse classifier. The first-level coarse classifier C_1 was trained with an initial pool of $20,000$ fragments all with size 32×32 pixels and with a single bin size $b = 16$. The stride for detection was $s_1 = 8$. The trained classifier is composed of 200 selected features belonging to all families out of which 90 were part based and the rest are HoG-components. To speedup the part-based feature computation we used the KD-Fern algorithm which computes the similarity of each fragment descriptor with only 25 candidates in each location. The details of the implementation of the KD-Fern are presented at the end of this section. For an input image we create a pyramid with 5 full-octave scales and 4 scales per octave. The full image AFS detection method with both levels is denoted by `AccFeatSynth` in the following evaluation graphs. We next present per-window and full image evaluation of the AFS.

Per-window evaluation. We evaluate the final classifier `AccFeatSynth_L2` using the per-window evaluation on the INRIA pedestrian dataset as specified in [3]. This type of evaluation allows a fair one-to-one comparison of the performance of the AFS with the original FS which is too slow to run on full images (the full image FS results shown in [1] use another classifier as a first level cascade and process the returned windows only). The results are shown in figure 2(a). The AFS achieves 6.5% miss rate at 10^{-4} false alarms per window (FPPW), which is a small decrease in performance compared to the original FS (`FeatSynth`: 5.6% miss rate at 10^{-4} FPPW). Although there are several methods better in the per-window evaluation, the failure of per-window performance in predicting full-image detection performance which is the true objective is discussed in detail in [7]. This is also the case for the AFS which is ranked highly in the full-image evaluation.

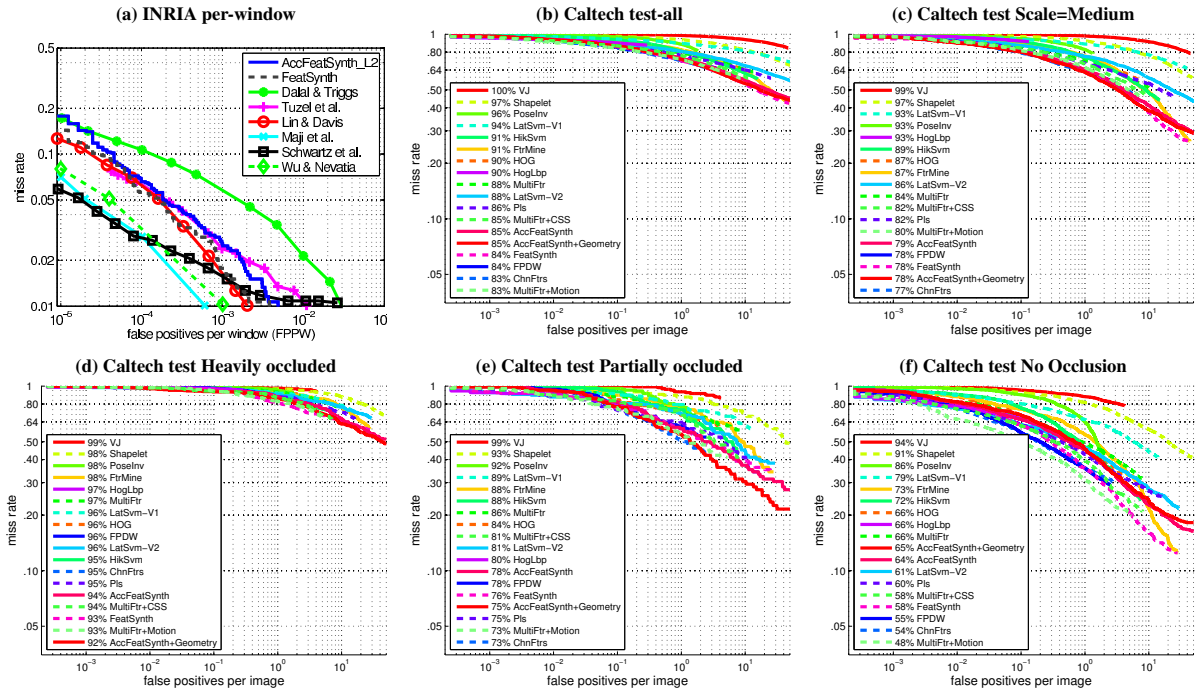

Figure 2. **(a) INRIA dataset per-window results.** Miss rate versus false positives per window. See [1] for details on the evaluated methods **(b) Results on the full Caltech pedestrian test dataset.** In parenthesis: the log-average of miss rates between 10^{-2} and 10^{0} false positives per image **(c-f) Caltech pedestrian test on several partitions.** See [7] for more details on the dataset and the evaluated methods.

Full image evaluation. The Caltech pedestrian benchmark [7] is divided into 10 different sessions containing movies taken from a moving vehicle. The *test* portion of the set which we used for evaluation consists of sessions $S6 - S10$. This is the largest available set for pedestrian detection containing over $100,000$ frames of video with $155,000$ instances of pedestrians in difficult real world scenarios. To reach the full range of annotated pedestrians in the dataset we used a $\times 3$ up-scaling of the images.

We also evaluated the effect of using geometric context. The `AccFeatSynth+Geometry` restricts the searched locations using "weak-geometry" constraints. Using known camera calibration and assumptions on the height of pedestrians it is possible to significantly narrow the space of window locations searched. This is not possible in the Caltech pedestrian dataset since the positioning of the camera in each session is slightly different. However, since camera positions are roughly similar, we can obtain some bounds on possible pedestrian locations in the image. We used the Caltech pedestrian *training set* to gather statistics on the height of each bounding box and its bottom y-axis image position, and fitted piece-wise linear bounds to this distribution. This allows us to narrow the scale search for bounding boxes starting at specific image y-positions. We show results with geometry (`AccFeatSynth+Geometry`) and without it (`AccFeatSynth`).

Figure 2 summarizes the evaluation on the Caltech set using the suggested methodology [7]. The evaluation uses the PASCAL criteria of a minimal 0.5 ratio between the intersection and the union of ground truth and detection bounding boxes. The DET curves plot the miss rate as a function of the number of false positives per image (fppi) on a log-log scale. Curves are compared by computing the log-average of the miss rates between 10^{-2} and 10^{0} fppi. All other presented methods were also trained on the INRIA training set allowing a fair comparison in terms of available training data. The `AccFeatSynth` achieves 85% log-average miss rate on the entire set (figure 2(b)) comparable to 83% achieved by the state-of-the-art method `multiFtr+Motion` [22] which combines several types of features including motion cues. The methods corresponding to the Deformable Part-based Model (DPM) are denoted by `Lat-SVM1` and `Lat-SVM2` achieve 94% and 88% log-average miss rates respectively. As discussed in [7], the medium range, defined in this set as pedestrians occupying between 30 and 80 image pixels in height, is the most relevant section of the dataset for the automotive case. In this portion of the dataset (figure 2(c)) the `AccFeatSynth+Geometry` is the second best performing method (78%), close to the leading one, `ChnFtrs` [6] (77%), and the `AccFeatSynth` performs similarly (79%). An interesting experiment is to test the sensitivity to different levels of occlusions using the available annotations: no occlusion, partially occluded ($= 1\% - 35\%$

occlusion) and heavily occluded ($= 35\% - 80\%$ occlusion). We expect multiple-part based methods to be less sensitive to higher levels of occlusions compared with template based methods or methods with only several parts. Indeed, the `AccFeatSynth`,`FeatSynth`,`AccFeatSynth+Geo.`, have a clear advantage for occluded pedestrians (ranked in places $5, 3, 1$ respectively), an advantage that gradually decreases for partial occlusion and no occlusion. This may suggest combining template-based methods for unoccluded pedestrians with part-based methods for handling occlusions.

Runtime evaluation. We measured the speed of our detection method implemented in C++ on a Intel Core i7-2600K@3.4Ghz computer with 8GB RAM similar in speed to that used to measure the methods we compare to in [7]. Figure 3(a) shows the log-average miss rate versus running time of all the methods tested on the Caltech dataset on pedestrians over 100 pixels. In this setting our method processes the original 640×480 without image up-scaling. The `AccFeatSynth` is the fastest method, running in 105 milliseconds per frame, close to 10 frames per second (fps), with 38% log-average miss rate. The second setting, defined as "reasonable" in [7] (pedestrians over 50 pixels which are not no heavily occluded) requires us to up-scale the image by a factor of 2. In this setting (Figure 3(b)) our method is the second fastest running at 1.72 fps with 65% log-average miss rate, preceded only by the FPDW [5] method (2.67 fps), which is a *template-based* method for pedestrian detection. In both settings our method provides an excellent accuracy and speed combination. Table 1(right) provides a breakdown of the AFS average runtime using a single thread on the 640×480 Caltech test images. The running time for the Cascaded Deformable Part-based method (c-DPM) [10] is not reported for the Caltech dataset, and we therefore measure it ourselves using the provided code (`voc-release4`, 1 component with 8 parts person model). The running times, summarized in table 1(left) (rows 1 and 2, single-thread), show that our method is $\times 4.5$ faster than the provided implementation of the c-DPM, which is currently considered the fastest part-based detection method implementation. Using geometry runtime speed is further increased by $\times 2.6$ on a single thread (third row of table 1(left)). Our multi-thread implementation (without geometry), independently processes different scales in parallel achieving a $X2.5$ speedup with 4 threads (bottom row), and reaching the stated 105 ms/frame.

KD-Ferns evaluation. In the AFS coarse level the KD-Ferns search is used to reduce for each image location the number of candidate model fragments for which similarity is computed. We construct a KD-Fern structure with $N = 25$ trees from the coarse-level database of 90 fragment descriptors, each of length 32. At detection time, the descriptor at a single position serves as an input query descriptor to the KD-Ferns algorithm, which returns the N closest candidates according to the search algorithm described in 2. N was chosen as the smallest number which maintains the performance when considering all candidates. We did a separate experiment to compare the KD-Ferns with existing approximate nearest neighbor (ANN) algorithms and with naive exhaustive search for searching our fragment descriptor database. For this comparison, we use the $\{\epsilon, \delta\}$-ANN task: find with probability δ a neighbor with euclidian distance d which is not larger than $(1 + \epsilon) \cdot d^*$ where d^* is the distance to the true nearest neighbor. We compare with two ANN algorithms: the hierarchical k-means tree algorithm [14] and the randomized kd-trees algorithm [19]. These algorithms were shown to best perform on similar tasks in [16], which also provides an efficient C++ implementation of the algorithms (FLANN) which we use here. As the searched database we use our 90 fragment descriptors set. The test query set consists of $50,000$ fragment descriptors densely sampled from caltech dataset images, as the ones used in our detection system. Using the KD-Ferns algorithm constructed as described above, we achieve a $\epsilon = 0.1$, $\delta = 0.94$ performance on the test set. Using a separate set of training queries, we automatically tune the optimal parameters (in terms of running time) of the kd-trees and k-means tree algorithms for achieving this $\{\epsilon, \delta\}$-approximation. The optimal randomized kd-trees uses 5 trees and the hierarchical k-means tree uses the "gonzales" initialization and a branching factor of 6. In [16] an automatic algorithm and parameter tuning is suggested, but for this dataset this algorithm always chooses the naive exhaustive search which is faster. Using profiling tools we analyzed why for our database these algorithms fail to run in sub-linear time. Each of these algorithms does reduce the number of query to database descriptor comparison, but has additional cost in traversing the trees and updating priority queues. For small databases (up to several hundred fragment descriptors in our experiments), this additional cost is higher than the saved cost of comparing descriptors. At test time, we optimize the running time of these two algorithms by limiting the number of visited leafs to the minimal number required to provide the $\{\epsilon, \delta\}$-approximation. The run time test was conducted on a single thread, using the same hardware described previously. The results are summarized in table 1(Right): the KD-Fern algorithm is $\times 2.4$ faster than exhaustive search, and $\times 8.6$,$\times 3.4$ faster than the kd-trees and k-means tree respectively. When used in the AFS for candidate reduction the KD-Ferns provides a $\times 1.5$ speedup of the fragment similarity map stage.

5. conclusions

We presented the AFS, a method for multiple-part based object detection running in real-time. We also introduce KD-Ferns, a new ANN search algorithm particularly efficient for searching small multi-dimensional datasets. In the

Caltech run-time	ms per frame
c-DPM (1 thread)	1164
AFS (1 thread)	252.1
AFS+Geo. (1 thread)	125
AFS (Multi-thread)	105

run time (ms)	Level 1	Level 2	Both levels
Local orient histograms	96.1	13.7	109.8
Fragment similarity maps	9.7	41.9	51.6
Classification score	37.4	53.3	90.7
Total	143.2	108.9	252.1

ANN Method run time	microsec/query
Randomized kd-trees	4.756
Hierarchical k-means tree	1.896
Exhaustive search (linear)	1.323
KD-Ferns	**0.553**

Table 1. **Left.** Running times in milliseconds per frame for Cascade DPM [10] (c-DPM) and for the Accelerated Feature Synthesis (AFS) using single thread, geometry constraints and multi-thread. **Middle.** Runtime breakdown (average ms) of the AFS on Caltech 640×480 images, for each level in the cascade and each processing stage using a single thread. **Right.** ANN method run time comparison in microseconds per query. Run time is averaged over 1000 measuring iterations and over 50K queries.

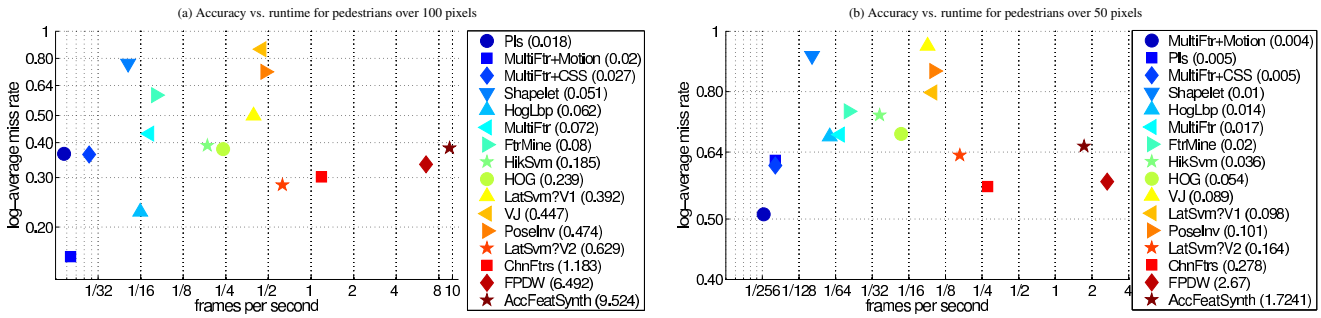

Figure 3. **Runtime evaluation** Log-average miss rate versus the runtime of each detector on the caltech test 640×480 images for: (a) pedestrians over 100 pixels, (b) pedestrians over 50 pixels. See [7] for details on comparison methodology and compared methods.

future we plan to extend the AFS by incorporating a multi-component model and large scale training for pushing forward state-of-the-art complex object detection.

Acknowledgements: We wish to thank Inna Stainvas and Ran Gazit for their useful comments and Lilach Levi for her support.

References

[1] A. Bar-Hillel, D. Levi, E. Krupka, and C. Goldberg. Part-based feature synthesis for human detection. In *ECCV 2010*, volume 6314, pages 127–142. 2010.

[2] R. Benenson, M. Mathias, R. Timofte, and L. J. V. Gool. Pedestrian detection at 100 frames per second. In *CVPR*, pages 2903–2910. IEEE, 2012.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[4] P. Dollár, R. Appel, and W. Kienzle. Crosstalk cascades for frame-rate pedestrian detection. In *ECCV*, 2012.

[5] P. Dollár, S. Belongie, and P. Perona. The fastest pedestrian detector in the west. In *BMVC*, 2010.

[6] P. Dollar, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, 2009.

[7] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *PAMI*, 99, 2011.

[8] C. Dubout and F. Fleuret. Exact acceleration of linear object detectors. In *Proceedings of the European Conference on Computer Vision*, 2012.

[9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2009 (VOC2009) Results. http://www.pascal-network.org/challenges/VOC/voc2009/workshop/index.html.

[10] P. F. Felzenszwalb, R. B. Girshick, and D. A. McAllester. Cascade object detection with deformable part models. In *CVPR*, pages 2241–2248. IEEE, 2010.

[11] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, 2010.

[12] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale invariant learning. In *CVPR*, 2003.

[13] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[14] K. Fukunaga and P. M. Narendra. A branch and bound algorithms for computing k-nearest neighbors. *IEEE Trans. Computers*, 24(7):750–753, 1975.

[15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60:91–110, 2004.

[16] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)*, pages 331–340, 2009.

[17] M. Pedersoli, A. Vedaldi, and J. Gonzàlez. A coarse-to-fine approach for fast deformable object detection. In *CVPR*, pages 1353–1360. IEEE, 2011.

[18] A. Shashua, Y. Gdalyahu, and G. Hayun. Pedestrian detection for driving assistance systems: Single-frame classification and system level performance. In *intelligent vehicles symposium*, pages 1–6, 2004.

[19] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *CVPR*, 2008.

[20] `www.mobileye.com`.

[21] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *CVPR*, 1:511, 2001.

[22] C. Wojek and B. Schiele. A performance evaluation of single and multi-feature people detection. In G. Rigoll, editor, *DAGM-Symposium*, volume 5096 of *Lecture Notes in Computer Science*, pages 82–91. Springer, 2008.