Cascade Object Detection with Deformable Part Models*

Pedro F. Felzenszwalb
University of Chicago
pff@cs.uchicago.edu

Ross B. Girshick
University of Chicago
rbg@cs.uchicago.edu

David McAllester
TTI at Chicago
mcallester@ttic.edu

Abstract

We describe a general method for building cascade classifiers from part-based deformable models such as pictorial structures. We focus primarily on the case of star-structured models and show how a simple algorithm based on partial hypothesis pruning can speed up object detection by more than one order of magnitude without sacrificing detection accuracy. In our algorithm, partial hypotheses are pruned with a sequence of thresholds. In analogy to probably approximately correct (PAC) learning, we introduce the notion of probably approximately admissible (PAA) thresholds. Such thresholds provide theoretical guarantees on the performance of the cascade method and can be computed from a small sample of positive examples. Finally, we outline a cascade detection algorithm for a general class of models defined by a grammar formalism. This class includes not only tree-structured pictorial structures but also richer models that can represent each part recursively as a mixture of other parts.

1. Introduction

A popular approach for object detection involves reducing the problem to binary classification. The simplest and most common example of this approach is the sliding window method. In this method a classifier is applied at all positions, scales, and, in some cases, orientations of an image. However, testing all points in the search space with a non-trivial classifier can be very slow. An effective method for addressing this problem involves applying a cascade of simple tests to each hypothesized object location to eliminate most of them very quickly [16, 12, 4, 15, 2, 13].

Another line of research, separate from cascade classifiers, uses part-based deformable models for detection. In this case an object hypothesis specifies a configuration of parts, which leads to a very large (exponential) hypothesis space. There has been significant success in algorithmic methods for searching over these large hypothesis spaces, including methods that are "asymptotically optimal" for tree-structured models [9]. However, these methods are still relatively slow when compared to simple classifiers defined by cascades. In this paper we describe a method for building cascades for part-based deformable models such as pictorial structures. In the most general setting, this method leads to a cascade version of top-down dynamic programming for a general class of grammar based models.

(a) (b)

Figure 1. Visualization of the amount of work performed by our algorithm over different regions of an image (top) using a car model (a) and a person model (b).

We focus primarily on the case of star-structured models due to their recent strong performance on difficult benchmarks such as the PASCAL datasets [11, 8, 5, 6, 7]. For star models, we obtain a fairly simple algorithm for early hypothesis pruning. This algorithm leads to a detection method over 20 times faster than the standard detection algorithm, which is based on dynamic programming and generalized distance transforms, without sacrificing detection accuracy. Figure 1 illustrates the amount of work done by our algorithm in different areas of an image using two different models.

As described in [11, 8], detection with a deformable part model can be done by considering all possible locations of a distinguished "root" part and, for each of those, finding the

*This research has been supported by NSF grant IIS-0746569.

best configuration of the remaining parts. In this case we need to compute an optimal configuration for each location of the root. These problems are not independent because the possible locations of the remaining parts are shared among different root locations. For tree-structured models one can use dynamic programming to account for this sharing [9].

In practice one is only interested in root locations that lead to high scoring configurations. The basic idea of our algorithm is to use a hierarchy of models defined by an ordering of the original model's parts. For a model with $n+1$ parts, including the root, we obtain a sequence of $n+1$ models. The i-th model in this sequence is defined by the first i parts from the original model. Using this hierarchy, we can prune low scoring hypotheses after looking at the best configuration of a subset of the parts. Hypotheses that score high under a weak model are evaluated further using a richer model. This process is analogous to a classical cascade and is similar to the cascades of [2, 15] in that the score of a weaker model is reused when computing the score of a richer one. However, when using deformable part models individual applications of the cascade are not independent, so, in analogy to classical dynamic programming, work done evaluating one hypothesis is also reused when evaluating others.

Our sequential search for parts is related to [1], where the authors propose a sequential search for semi-local features that fit a global arrangement. The work in [1] also considered the problem of selecting parameters that lead to fast search with a low false negative rate, by making some assumptions on the form of the distribution of local features and analyzing statistics of training data. We use an alternative approach (see below) that makes fewer assumptions and relies more heavily on the training data.

The time it takes to evaluate a hypothesis for a part-based model is highly dependent on the complexity of the individual part models. Besides simplifying a model by removing some of its parts, we also consider simplifications that arise from replacing the original part appearance models with simpler ones that are much faster to compute. In this case, for a model with $n+1$ parts we get a hierarchy of $2(n+1)$ models. The first $n+1$ models are obtained by sequentially adding parts with simplified appearance models. The second $n+1$ models are obtained by sequentially replacing each simplified appearance model with its full one.

Our algorithm prunes partial hypotheses using thresholds on their scores. Admissible thresholds would not prune any partial hypothesis that leads to a complete detection scoring above a global threshold. We define the error of a set of thresholds to be the fraction of full hypotheses scoring above the global threshold that are incorrectly pruned. To select pruning thresholds, we introduce the notion of probably approximately admissible (PAA) thresholds. PAA thresholds have a low error with high probability.

We show that PAA thresholds can be obtained by looking at statistics of partial hypothesis scores over positive examples. This leads to a simple method for picking *safe* and *effective* thresholds. The thresholds are safe because they have low error with high probability. They are effective because they lead to a fast cascade with significant pruning.

[9] notes that by using dynamic programming and distance transforms the relationships among parts in a tree-structured model can be taken into account "for free." That is, it takes very little additional time to detect whole object configurations as opposed to individually detecting parts on their own. Our results push this idea further. In practice we find that it is possible to detect whole object configurations much faster than detecting each individual part.

2. Object Detection with Star Models

We start by defining a general framework for object detection with star-structured deformable part models that includes the setting in [11, 8].

Let M be a model with a root part v_0 and n additional parts $v_1, \ldots, v_n$. Let Ω be a space of locations for each part within an image. For example, $\omega \in \Omega$ could specify a position and scale. Let $m_i(\omega)$ be the score for placing v_i in location ω. This score depends on the image data, but we assume the image is implicitly defined to simplify notation.

For a non-root part, let $a_i(\omega)$ specify the ideal location for v_i as a function of the root location. Let Δ be a space of displacements, and let $\oplus : \Omega \times \Delta \to \Omega$ be a binary operation taking a location and a displacement to a "displaced location." Let $d_i(\delta)$ specify a deformation cost for a displacement of v_i from its ideal location relative to the root.

An object configuration specifies a location for the root and a displacement for each additional part from its ideal location relative to the root. The score of a configuration is the sum of the scores of the parts at their locations minus deformation costs associated with each displacement.

$$\text{score}(\omega, \delta_1, \ldots, \delta_n) = m_0(\omega) + \sum_{i=1}^{n} m_i(a_i(\omega) \oplus \delta_i) - d_i(\delta_i) \quad (1)$$

We can define an overall score for a root location based on the maximum score of a configuration rooted at that location. In a star model each part is only attached to the root, so the score can be factored as follows.

$$\text{score}(\omega) = m_0(\omega) + \sum_{i=1}^{n} \text{score}_i(a_i(\omega)) \quad (2)$$

$$\text{score}_i(\eta) = \max_{\delta_i \in \Delta} (m_i(\eta \oplus \delta_i) - d_i(\delta_i)) \quad (3)$$

Here $\text{score}_i(\eta)$ is the maximum, over displacements of the part from its ideal location, of the part score minus the deformation cost associated with the displacement.

For the models in [11, 8], $m_i(\omega)$ is the response of a filter in a dense feature pyramid, and $d_i(\delta)$ is a (separable) quadratic function of δ. To detect objects [11, 8] look for root locations with an overall score above some threshold, $\text{score}(\omega) \geq T$. A dynamic programming algorithm is used to compute $\text{score}(\omega)$ for *every* location $\omega \in \Omega$. Using the fast distance transforms method from [9] the detection algorithm runs in $O(n|\Omega|)$ time if we assume that evaluating the appearance model for a part at a specific location takes $O(1)$ time. In practice, evaluating the appearance models is the bottleneck of the method.

3. Star-Cascade Detection

Here we describe a cascade algorithm for star models that uses a sequence of thresholds to prune detections using subsets of parts.

Note that we are only interested in root locations where $\text{score}(\omega) \geq T$. By evaluating parts in a sequential order we can avoid evaluating the appearance model for most parts almost everywhere. For example, when detecting people we might evaluate the score of the head part at each possible location and decide that we do not need to evaluate the score of the torso part for most locations in the image. Figure 2 shows an example run of the algorithm.

We use $\tilde{m}_i(\omega)$ to denote a memoized version of $m_i(\omega)$. In a memoized function whenever a value is computed we store it to avoid recomputing it later. Memoization can be implemented by maintaining an ω-indexed array of already-computed values and checking in this array first whenever $\tilde{m}_i(\omega)$ is called to avoid computing the appearance model at the same location more than once.

The cascade algorithm (Algorithm 1) for a star-structure model with $n+1$ parts takes a global threshold T and a sequence $2n$ of intermediate thresholds. To simplify the presentation we assume the root appearance model is evaluated first even though this ordering is not a requirement.

For each root location $\omega \in \Omega$, we evaluate $\text{score}(\omega)$ in n stages. The variable s accumulates the score over stages. In the i-th stage we compute $\text{score}_i(\eta)$, the contribution of part v_i, using the variable p. During evaluation of $\text{score}(\omega)$ there are two opportunities for pruning.

Hypothesis pruning: If the score at ω with the first i parts is below t_i, then the hypothesis at ω is pruned without evaluating parts v_i through v_n (line 5). Intuitively, placing the remaining parts will not make $\text{score}(\omega)$ go above T.

Deformation pruning: To compute v_i's contribution we need to search over deformations $\delta \in \Delta$. The algorithm will skip a particular δ if the score of the first i parts minus $d_i(\delta)$ is below t'_i (line 8). Intuitively, displacing v_i by δ costs too much to allow the $\text{score}(\omega)$ to go above T.

Note that memoizing the appearance models is important because several root locations might want to evaluate $m_i(\omega)$ at the same location.

For a fixed global threshold T, any input to Algorithm 1 that correctly computes $\text{score}(\omega)$ whenever it is above T is called a T-admissible set of thresholds. If given T-admissible thresholds, Algorithm 1 returns exactly the same set of detections as the standard dynamic programming algorithm. In the next section we investigate the case of *good inadmissible* thresholds that produce a cascade with a low error rate but still allow aggressive pruning.

The worst-case time of Algorithm 1 is $O(n|\Omega||\Delta|)$ if $m_i(\omega)$ is taken to cost $O(1)$, which is slower than the standard dynamic programming algorithm with distance transforms. However, in practice, for typical models Δ can be safely made relatively small (*c.f.* Section 6). Moreover, searching over Δ is usually no more expensive than evaluating $m_i(\omega)$ at a single location, because the spatial extent of a part is of similar size as its range of displacement. The worse-case time of both methods is the same, $O(n|\Omega||\Delta|)$, if we assume evaluating $m_i(\omega)$ takes $O(|\Delta|)$ time.

Data: Thresholds $((t_1, t'_1), \ldots, (t_n, t'_n))$ and T
Result: Set of detections D

```
1  D ← ∅
2  for ω ∈ Ω do
3      s ← m̃_0(ω)
4      for i = 1 to n do
5          if s < t_i then skip ω
6          p ← −∞
7          for δ ∈ Δ do
8              if s − d_i(δ) < t'_i then skip δ
9              p ← max(p, m̃_i(a_i(ω) ⊕ δ) − d_i(δ))
10         end
11         s ← s + p
12     end
13     if s ≥ T then D ← D ∪ {ω}
14 end
15 return D
```

Algorithm 1: star-cascade

Figure 2 shows how, in practice, the cascade algorithm avoids evaluating $m_i(\omega)$ for most locations $\omega \in \Omega$ except for one or two parts.

4. Pruning Thresholds

Suppose we have a model M and a detection threshold T. Let $x = (\omega, I)$ be an example of a location within an image I where $\text{score}(\omega) \geq T$. Let $\mathcal{D}$ be a distribution over such examples.

For a sequence of thresholds $t = ((t_1, t'_1), \ldots, (t_n, t'_n))$ let $\text{csc-score}(t, \omega)$ be the score computed for ω by the cascade algorithm using t. If the cascade prunes ω we say $\text{csc-score}(t, \omega) = -\infty$.

We define the *error* of t on $\mathcal{D}$ as the probability that the cascade algorithm will incorrectly compute $\text{score}(\omega)$ on a

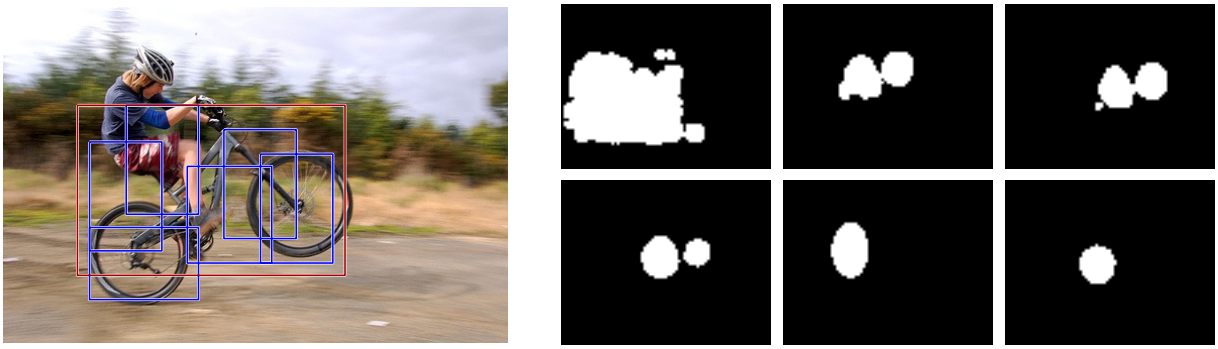

Figure 2. A sample image with a bicycle detection (left). Each image in the right shows (in white) positions where a particular part appearance model was evaluated at the scale of the bicycle detection. The images are shown in "cascade order" from left to right and top to bottom. The image for the root part, which was evaluated first, is not shown because its appearance model is evaluated at all locations. After evaluating the first non-root part, nearly all locations were pruned by the cascade.

random example from $\mathcal{D}$,

$$\text{error}(t) = P_{x\sim\mathcal{D}}(\text{csc-score}(t,\omega) \neq \text{score}(\omega)). \quad (4)$$

We would like to find a sequence of thresholds that has a small error. Note that in practice we are only interested in having a small error on positive examples. In particular we do not care if the cascade incorrectly prunes a negative example that scores above T. Thus we can take $\mathcal{D}$ to be a distribution over high-scoring positive examples.

Below we show that we can learn a good sequence of thresholds by looking at a small number of examples from $\mathcal{D}$. In analogy to PAC learning [14] we would like to select thresholds that have a small error with high probability.

We say a sequence of thresholds t is (ϵ, δ) *probably approximately admissible* (PAA) if the probability that the error of t is greater than ϵ is bounded by δ,

$$P(\text{error}(t) > \epsilon) \leq \delta. \quad (5)$$

Let $\delta_1, \ldots, \delta_n$ be the optimal displacements for the non-root parts of M on an example $x = (\omega, I)$. We can define partial scores that take into account the first i parts and the first i parts minus the i-th deformation cost,

$$x_i = m_0(\omega) + \sum_{j=1}^{i-1} m_j(a_j(\omega) + \delta_j) - d_j(\delta_j), \quad (6)$$

$$x'_i = x_i - d_i(\delta_i). \quad (7)$$

The star-cascade algorithm will find an optimal configuration and score for x if and only if $x_i \geq t_i$ and $x'_i \geq t'_i$ for all $1 \leq i \leq n$.

Let X be m independent samples from $\mathcal{D}$. We can select thresholds by picking,

$$t_i = \min_{x\in X} x_i \qquad t'_i = \min_{x\in X} x'_i \quad (8)$$

These are the tightest thresholds that make no mistakes on X. The following theorem shows they also have low error with high probability provided that m is sufficiently large.

Theorem 1. *If we select t according to equation (8) using $m \geq 2n/\epsilon \ln(2n/\delta)$ samples from $\mathcal{D}$ then t is (ϵ, δ) probably approximately admissible with respect to $\mathcal{D}$.*

Proof. By a union bound $\text{error}(t) \leq \epsilon$ if $P_{x\sim\mathcal{D}}(x_i < t_i)$ and $P_{x\sim\mathcal{D}}(x'_i < t'_i)$ are $\leq \epsilon/(2n)$ for all i. Thus $\text{error}(t) \leq \epsilon$ unless for some i all m samples of the x_i or x'_i are above the $\epsilon/(2n)$-th percentile of their distribution. The probability of that event is $\leq 2n(1 - \epsilon/(2n))^m$. To bound this by δ we only need $m \geq 2n/\epsilon \ln(2n/\delta)$ samples. □

5. Simplified Part Appearance Models

So far we have considered a cascade for star models that is defined by a hierarchy of $n + 1$ models. Given a predefined part order, the i-th model is formed by adding the i-th part to the $(i - 1)$-st model. The goal of the cascade is to detect objects while making as few appearance model evaluations as possible. The star-cascade algorithm achieves this goal by pruning hypotheses using intermediate scores.

A complementary approach is to consider a simplified appearance model for each part, $\hat{m}_i(\omega)$, that computes an inexpensive approximation of $m_i(\omega)$.

A hierarchy of $2(n + 1)$ models can be defined with the first $n + 1$ models constructed as before, but using the simplified appearance models, and the second $n + 1$ models defined by sequentially removing one of the remaining simplified appearance models $\hat{m}_i$ and replacing it with the original one m_i.

With simplified parts, the star-cascade operates as before for stages 1 through $n+1$, except that pruning decisions are based on the (memoized) evaluations of the simplified appearance models $\hat{m}_i(\omega)$. During the remaining stages, full appearance models replace simplified ones. When replacing an appearance model, the algorithm must redo the search over the deformation space because the optimal placement may change. But now we will often prune a hypothesis before evaluating any of the expensive appearance models.

This version of the algorithm requires $4n + 1$ intermediate thresholds. Just as before, these thresholds can be selected using the method from the previous section.

For the models in [11, 8], simplified appearance models can be defined by projecting the HOG features and the weight vectors in the part filters to a low dimensional space. We did PCA on a large sample of HOG features from training images in the PASCAL datasets. A simplified appearance model can be specified by the projection into the top k principal components. For the 31-dimensional HOG features used in [8], a setting of $k = 5$ leads to appearance models that are approximately 6 times faster to evaluate than the original ones. This approach is simple and only introduces a small amount of overhead — the cost of projecting each feature vector in the feature pyramid onto the top k principal components.

6. General Grammar Models

Here we consider a fairly general class of grammar models for representing objects in terms of parts. It includes tree-structured pictorial structure models as well as more general models that have variable structure. For example, we can define a person model in which the face part is composed of eyes, a nose, and either a smiling or frowning mouth. We follow the framework and notation in [10].

Let $\mathcal{N}$ be a set of nonterminal symbols and $\mathcal{T}$ be a set of terminal symbols. Let Ω be a set of possible locations for a symbol within an image. For $\omega \in \Omega$ we use $X(\omega)$ to denote the placement of a symbol at a location in the image.

Appearance models for the terminals are defined by a function $\mathrm{score}(A, \omega)$ that specifies a score for $A(\omega)$. The appearance of nonterminals is defined in terms of expansions into other symbols. Possible expansions are defined by a set of scored production rules of the form

$$X(\omega_0) \stackrel{s}{\rightarrow} Y_1(\omega_1), \ldots, Y_n(\omega_n), \tag{9}$$

where $X \in \mathcal{N}$, $Y_i \in \mathcal{N} \cup \mathcal{T}$, and $s \in \mathbb{R}$ is a score.

To avoid enumerating production rules that differ only by symbol placement, we define grammar models using a set of parameterized production schemas of the form

$$X(\omega_0(z)) \stackrel{\alpha(z)}{\rightarrow} Y_1(\omega_1(z)), \ldots, Y_n(\omega_n(z)). \tag{10}$$

Each schema defines a collection of productions consisting of one production for each value of a parameter z. Given a fixed value of z, the functions $\omega_0(z), \ldots, \omega_n(z)$, and $\alpha(z)$ yield a single production of the form in (9).

Star models can be represented using a nonterminal X_i and a terminal A_i for each part. We have $\mathrm{score}(A_i, \omega) = m_i(\omega)$. A placement of the root nonterminal X_0 defines ideal locations for the remaining parts. This is captured by an instance of the following production for each ω,

$$X_0(\omega) \stackrel{0}{\rightarrow} A_0(\omega), X_1(a_1(\omega)), \ldots, X_n(a_n(\omega)). \tag{11}$$

We can encode these productions using a schema where z ranges over Ω. These rules are called *structural rules*.

A part can be displaced from its ideal location at the expense of a deformation cost. This is captured by an instance of the following production for each ω and δ,

$$X_i(\omega) \stackrel{-d_i(\delta)}{\rightarrow} A_i(\omega \oplus \delta). \tag{12}$$

We can encode these productions using a schema where z ranges over $\Omega \times \Delta$. These rules are called *deformation rules*.

We restrict our attention to acyclic grammars. We also require that no symbol may appear in the right hand side of multiple schemas. We call this class *no-sharing acyclic grammars*. It includes pictorial structures defined by arbitrary trees as well as models where each part can be one of several subtypes. But it does not include models where a single part is used multiple times in one object instance such as a car model where one wheel part is used for both the front and rear wheels.

For acyclic grammars, we can extend the scores of terminals to scores for nonterminals by the recursive equation

$$\mathrm{score}(X, \omega) = \max_{X(\omega) \stackrel{s}{\rightarrow} Y_1(\omega_1), \ldots, Y_n(\omega_n)} s + \sum_{i=1}^{n} \mathrm{score}(Y_i, \omega_i), \tag{13}$$

where the $\max$ is over rules with $X(\omega)$ in the left hand side. Since the grammar is acyclic, the symbols can be ordered such that a bottom-up dynamic programming algorithm can compute score tables $V_X[\omega] = \mathrm{score}(X, \omega)$.

Scores can also be computed by a recursive top-down procedure. To compute $\mathrm{score}(X, \omega)$ we consider every rule with $X(\omega)$ in the left hand side and sequentially compute the scores of placed symbols in the right hand side using recursive calls. Computed scores should be memoized to avoid recomputing them in the future.

For object detection we have a root symbol S and we would like to find all locations ω where $\mathrm{score}(S, \omega) \geq T$. It is natural to introduce pruning into the top-down algorithm in analogy to the star-cascade method (Algorithm 1).

As the top-down method traverses derivations in depth-first left-right order, we can keep track of a "prefix score" for the current derivation. Upon reaching $X(\omega)$ we can compare the current prefix score to a threshold $t(X)$. If the prefix score is below $t(X)$, then we could pretend $\mathrm{score}(X, \omega) = -\infty$ without computing it. This is a form of pruning. However, generally there will be multiple requests for $\mathrm{score}(X, \omega)$ and pruning may be problematic when memoized scores are reused. The value memoized for $X(\omega)$ depends on the prefix score of the first request for $\mathrm{score}(X, \omega)$. Due to pruning, the memoized value might be different than what a later request would compute. In particular, if a later request has a higher prefix score, the associated derivation should undergo less pruning.

To address this issue, we define $\text{pscore}(Y, \omega)$ to be the maximum prefix score over all requests for $Y(\omega)$. In a grammar with no sharing there is a single schema with Y in the right hand side. For each schema of the form in (10) we have

$$\text{pscore}(Y_i, \omega) = \max_{z \in \omega_i^{-1}(\omega)} \text{pscore}(X, \omega_0(z)) + \alpha(z) + \sum_{j=1}^{i-1} \text{score}(Y_j, \omega_j(z)). \quad (14)$$

Thus pscores for Y_i can be computed once we have pscores for X and scores for $Y_1, \ldots, Y_{i-1}$. The set of parameter values z yielding $\omega = \omega_i(z)$ is denoted by $\omega_i^{-1}(\omega)$.

The grammar-cascade algorithm goes over schemas in a depth-first left-right order. It computes $\text{pscore}(X, \omega)$ before computing $\text{score}(X, \omega)$, and it prunes computation by comparing $\text{pscore}(X, \omega)$ to a threshold $t(X)$.

The procedure `compute` takes a symbol X and a table of prefix scores $P_X[\omega] \approx \text{pscore}(X, \omega)$. It returns a table of values $V_X[\omega] \approx \text{score}(X, \omega)$. These tables are not exact due to pruning. As in the star-cascade, we can pick thresholds using a sample of positive examples. For each symbol X we can pick the highest threshold $t(X)$ that does not prune optimal configurations on the positive examples.

Data: X, P_X
Result: V_X

1 $V_X[\omega] \leftarrow -\infty$
2 **if** $X \in \mathcal{T}$ **then**
3 **if** $P_X[\omega] \geq t(X)$ **then** $V_X[\omega] \leftarrow \text{score}(X, \omega)$
4 **return** V_X
5 **end**
6 **foreach** $X(\omega_0(z)) \overset{\alpha(z)}{\rightarrow} Y_1(\omega_1(z)), \ldots, Y_n(\omega_n(z))$ **do**
7 $V_0[\omega] \leftarrow -\infty$
8 **if** $P_X[\omega] \geq t(X)$ **then** $V_0[\omega] \leftarrow P_X[\omega]$
9 **for** $i = 1$ **to** n **do**
10 $P_i[\omega] \leftarrow \max_{z \in \omega_i^{-1}(\omega)} \alpha(z) + \sum_{j=0}^{i-1} V_j[\omega_j(z)]$
11 $V_i \leftarrow \texttt{compute}(Y_i, P_i)$
12 **end**
13 $V[\omega] \leftarrow \max_{z \in \omega_0^{-1}(\omega)} \alpha(z) + \sum_{j=1}^{n} V_j[\omega_j(z)]$
14 $V_X[\omega] \leftarrow \max(V_X[\omega], V[\omega])$
15 **end**
16 **return** V_X

Procedure `compute`

For a terminal X, `compute` evaluates $\text{score}(X, \omega)$ at locations ω with high pscores. For a nonterminal, `compute` loops over schemas with X in the left hand side. Line 10 computes pscores for Y_i before calling `compute` recursively to obtain scores for Y_i. Line 13 computes scores for X under a particular schema. The result V_X is the running max of scores under different schemas.

To understand the worst case runtime of this algorithm we need to consider the $\max$ over z that appears in lines 10 and 13. Suppose each schema is a structural rule similar to (11) with a bounded number of symbols in the right hand side or a deformation rule similar to (12).

For a structural rule (11), $\omega_0^{-1}(\omega) = \{\omega\}$. In this case, line 13 simply sums the scores of the right hand side symbols after shifting each by its ideal displacement $a_i(\omega)$. For a deformation rule (12), $\omega_0^{-1}(\omega) = \{\omega\} \times \Delta$. In this case, line 13 takes a $\max$ over Δ. The situations are similar for line 10. Thus, assuming it takes $O(1)$ time to evaluate $\text{score}(A, \omega)$ for a terminal, the runtime of the algorithm is $O(|\Omega||\Delta|)$ per schema. When scores over deformation rules can be computed via fast distance transforms the runtime becomes $O(|\Omega|)$ per schema.

Detection is performed by calling `compute` on a root symbol S with $P_S[\omega] = 0$. In the case of tree-structured pictorial structure models where fast distance transforms can be used, the time of $\texttt{compute}(S, P_S)$ is $O(n|\Omega|)$ for a model with n parts. This is the same as bottom-up dynamic programming with fast distance transforms. In this case the grammar-cascade algorithm has better worst-case time than the star-cascade algorithm, but by implementing both methods we found that the specialized star-cascade algorithm outperformed `compute` by about a factor of two. This empirical result comes from a confluence of two factors: the restricted structure of the star model avoids the need to maintain prefix score tables, and for the models we consider $|\Delta|$ is small enough that brute force search over it, for a small number of locations, outperforms fast distance transforms over the full space Ω.

7. Experimental Results

To evaluate our algorithm for star-structured models we compared it to the baseline detection method based on dynamic programming and distance transforms. We used the publicly available system from [8] as a testbed. We note that [8] provides an implementation of the baseline detection algorithm that is already quite efficient.

We evaluated our algorithm by looking at the detection time speedup and average precision (AP) score with respect to the baseline. The evaluation was done over all 20 classes of the PASCAL 2007 dataset [5] as well as on the INRIA Person dataset [3]. For the PASCAL experiments we obtained the six-component models used by the UoC-TTI entry in the 2009 PASCAL VOC Challenge [7]. For the INRIA experiments we obtained the one-component model from [8]. These models achieve state-of-the-art detection results. Our experiments show that the cascade algorithm achieves a significant speedup, of more than 20 times on average, with negligible decrease in detection accuracy.

The PASCAL models were trained on the 2009 training and validation data, which includes the 2008 data as a sub-

set. We wanted "fresh" positive training examples for selecting thresholds, separate from the examples used to train the models, so we conducted our evaluation on the PASCAL 2007 dataset. Testing on the 2007 dataset ensured that the statistics for the threshold training and test data were the same. Note that testing on the 2007 dataset using models trained on the 2009 dataset might not lead to the best possible detection accuracy, but we are only interested in the relative performance of the cascade and the baseline method.

In the case of the INRIA Person dataset we did not have access to fresh positive examples, so we used the same examples with which the model was trained. Even though the PAA threshold theory does not apply in this setting, the cascade achieved exactly the same AP scores as the baseline.

Our implementation of the cascade algorithm has a single parameter controlling the number of components used for the PCA approximation of the low-level features. This was set to 5 in advance based on the magnitude of the eigenvalues from the PCA of HOG features.

We compared the runtime of the cascade algorithm versus the baseline for two global detection threshold settings. A higher global threshold allows for more pruning in the cascade at the cost of obtaining a lower recall rate. The first setting was selected so that the resulting precision-recall curve should reach the precision-equals-recall point. Empirically we found that this setting results in a detector with typical AP scores within 5 points of the maximum score. This setting is tuned for speed without sacrificing too much recall. The second setting results in the maximum possible AP score with less emphasis on speed. This configuration requires picking a global threshold so the detector achieves its full recall range. We approximated this goal by selecting a global threshold such that the detector would return results down to the precision equals 5% level. For each global detection threshold we picked pruning thresholds using the procedure outlined in Section 4.

Figure 3 illustrates precision-recall curves obtained with the cascade and baseline methods. The performance of the cascade algorithm follows the performance of the baseline very closely. The complete experimental results are summarized in Tables 1 and 2. We see that the cascade method achieves AP scores that are essentially identical to the baseline for both global threshold settings. Sometimes the cascade achieves slightly higher AP score due to pruning of false positives. The maximum recall obtained with the cascade is only slightly below the baseline, indicating that very few true positives were incorrectly pruned. The difference in recall rates is reported as the *recall gap*.

For the purpose of timing the algorithms, we ignored the time it takes to compute the low-level feature pyramid from an image as that is the same for both methods (and can be shared among different detectors). Feature pyramid generation took an average of 459ms per image on the PASCAL dataset and 730ms on the INRIA dataset. With the precision-equals-recall threshold, the cascade detector ran 22 times faster than the baseline on average. As an example, the mean detection time per image for the motorbike model was 10.1s for the baseline versus 313ms for the cascade, and the mean time per image for the person model was 8.5s for the baseline versus 682ms for the cascade.

We also tested the star-cascade algorithm without PCA filters. In this mode, the mean speedup dropped to 8.7 over the baseline at the precision-equals-recall level.

Note that [8] reports detection times of around 2 seconds per image because it uses a parallel implementation of the baseline algorithm. We turned off that feature to facilitate comparison. Both the baseline and the cascade are equally easy to parallelize. For example, one could search over different scales at the same time. All experiments were conducted using single-threaded implementations on a 2.67GHz Intel Core i7 920 CPU computer running Linux.

8. Conclusion

The results of this paper are both theoretical and practical. At a theoretical level we have shown how to construct a cascade variant of a dynamic programming (DP) algorithm. From an abstract viewpoint, a DP algorithm fills values in DP tables. In the cascade version the tables are partial — not all values are computed. Partial DP tables are also used in A* search algorithms. However, the cascade variant of DP runs without the overhead of priority queue operations and with better cache coherence. A second theoretical contribution is a training algorithm for the thresholds used in the cascade and an associated high-confidence bound on the error rate of those thresholds — the number of desired detections that are missed because of the pruning of the intermediate DP tables.

At a practical level, this paper describes a frame-rate implementation of a state-of-the-art object detector. The detector can easily be made to run at several frames per second on a multicore processor. This should open up new applications for this class of detectors in areas such as robotics and HCI. It should also facilitate future research by making richer models computationally feasible. For example, the techniques described in this paper should make it practical to extend the deformable model paradigm for object detection to include search over orientation or other pose parameters. We believe the performance of deformable model detectors can still be greatly improved.

References

[1] Y. Amit and D. Geman. A computational model for visual selection. *Neural Computation*, 11(7):1691–1715, 1999.

[2] L. Bourdev and J. Brandt. Robust object detection via soft cascade. In *CVPR*, 2005.

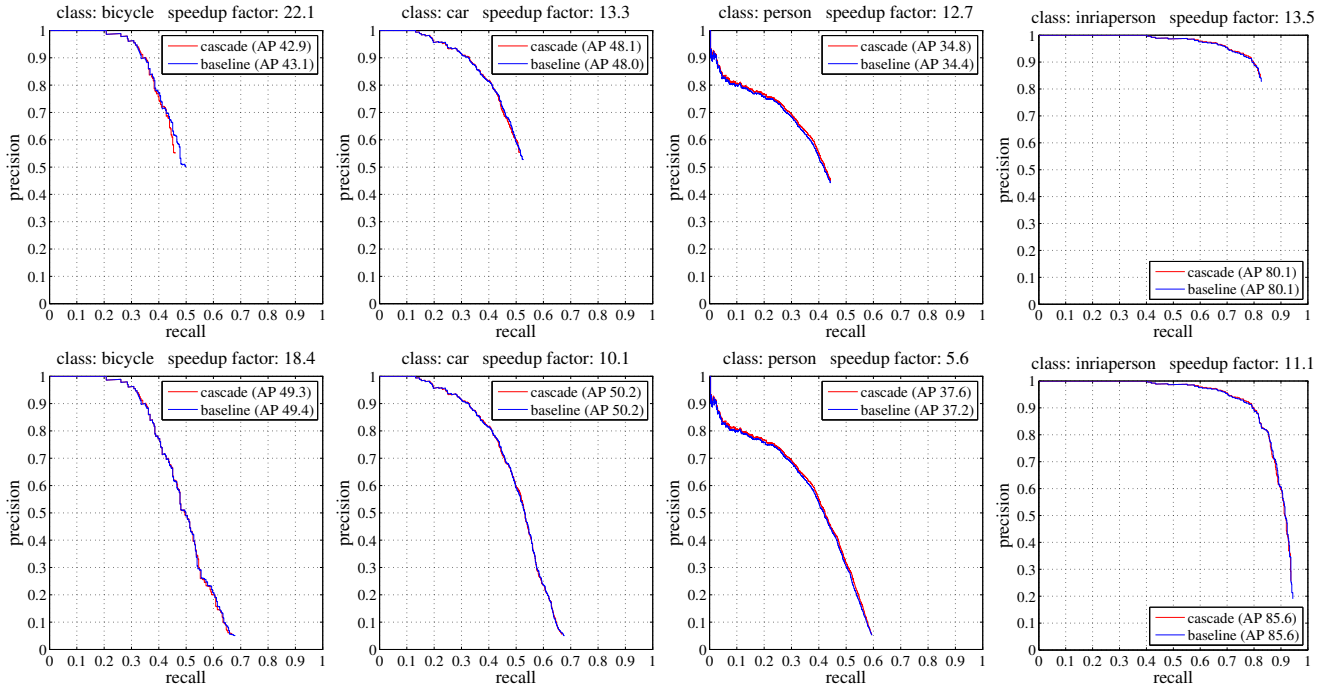

Figure 3. Sample precision-recall curves for bicycle, car, person, and INRIA person with the global threshold set to hit the precision-equals-recall point (top row) and the precision = 0.05 level (bottom row).

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	inria
Speedup factor	22.7	22.1	16.5	11.6	22.1	36.0	13.3	25.6	23.4	23.2	29.8	15.2	16.2	32.6	12.7	23.3	32.8	18.1	23.3	27.2	13.5
Baseline AP	21.1	43.1	10.6	12.2	24.0	42.2	48.0	15.9	13.4	19.0	7.1	10.7	31.3	32.9	34.4	12.0	20.3	20.8	29.3	36.3	80.1
Cascade AP	21.1	42.9	10.4	12.4	24.1	42.5	48.1	15.5	13.4	19.0	8.0	10.7	31.3	33.0	34.8	12.0	20.3	20.2	28.8	36.5	80.1
Recall gap	0.7	3.9	1.1	3.0	0.4	4.7	1.3	2.0	1.2	2.0	1.5	1.8	0.9	1.5	0.0	1.0	3.7	1.7	4.6	0.3	0.3

Table 1. Results for the global threshold set so each PR curve would reach the precision = recall point. Mean speedup for all classes = 22.0.

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	inria
Speedup factor	13.6	18.4	16.9	9.9	12.3	19.0	10.1	13.6	13.4	15.1	19.1	12.0	13.1	21.5	5.6	11.6	23.9	15.9	14.3	9.8	11.1
Baseline AP	22.8	49.4	10.6	12.9	27.1	47.4	50.2	18.8	15.7	23.6	10.3	12.1	36.4	37.1	37.2	13.2	22.6	22.9	34.7	40.0	85.6
Cascade AP	22.7	49.3	10.6	13.0	26.6	47.4	50.2	18.8	15.7	23.1	11.3	12.3	36.0	37.1	37.6	13.6	22.7	23.1	34.2	40.0	85.6
Recall gap	0.4	1.2	0.2	0.8	1.5	2.3	0.7	0.8	0.1	1.6	0.0	1.0	1.4	1.8	0.0	2.3	3.7	1.7	2.1	0.0	0.7

Table 2. Results for the global threshold set so each PR curve would reach precision = 0.05. Mean speedup for all classes = 14.3.

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

[4] M. Elad, Y. Hel-Or, and R. Keshet. Pattern detection using a maximal rejection classifier. *PRL*, 23(12):1459–1471, 2002.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL VOC 2007 Results.

[6] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL VOC 2008 Results.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL VOC 2009 Results.

[8] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *PAMI*, 2009.

[9] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *IJCV*, 61(1):55–79, 2005.

[10] P. Felzenszwalb and D. McAllester. Object detection grammars. *University of Chicago, CS Dept., Tech. Rep. 2010-02*.

[11] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008.

[12] F. Fleuret and D. Geman. Coarse-to-fine face detection. *IJCV*, 41(1):85–107, 2001.

[13] S. Gangaputra and D. Geman. A design principle for coarse-to-fine classification. In *CVPR*, 2006.

[14] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

[15] J. Šochman and J. Matas. Waldboost-learning for time constrained sequential detection. In *CVPR*, 2005.

[16] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.