Learning to Detect Objects of Many Classes Using Binary Classifiers

Ramana Isukapalli[1], Ahmed Elgammal[2], and Russell Greiner[3]

[1] Lucent Technologies, Bell Labs Innovations, Whippany, NJ 07981, USA
[2] Rutgers University, New Brunswick, NJ 08854, USA
[3] University of Alberta, Edmonton, CA T6G 2E8, CA

Abstract. Viola and Jones [VJ] demonstrate that cascade classification methods can successfully detect objects belonging to a single class, such as faces. Detecting and identifying objects that belong to any of a set of "classes", *many class detection*, is a much more challenging problem. We show that objects from each class can form a "cluster" in a "classifier space" and illustrate examples of such clusters using images of real world objects. Our detection algorithm uses a "decision tree classifier" (whose internal nodes each correspond to a VJ classifier) to propose a class label for every sub-image W of a test image (or reject it as a negative instance). If this W reaches a leaf of this tree, we then pass W through a subsequent VJ cascade of classifiers, specific to the identified class, to determine whether W is truly an instance of the proposed class. We perform several empirical studies to compare our system for detecting objects of any of M classes, to the obvious approach of running a *set* of M learned VJ cascade classifiers, one for each class of objects, on the same image. We found that the detection rates are comparable, and our many-class detection system is about as fast as running a *single* VJ cascade, and scales up well as the number of classes increases.

1 Introduction

The pioneering work of Viola and Jones [17, 16] has led to a successful face detection method based on "cascade classifiers", where each classifier is a binary classifier that is learned by applying Adaboost [3] (or some related algorithm [15, 18, 8, 13]) to a database of training images of faces and non-faces. The underlying principle in all these algorithms is to learn many binary classifiers during the training phase, then at performance time, run these classifiers as a "cascade" (*i.e.*, in a sequence one after another) on each region (at various resolutions) of the test image, eliminating non-faces at each stage. This work has also been used to detect objects of many other "classes" (like cars, motorbikes, etc.). Many researchers have extended the cascade detection method to solve several other related problems [7, 4, 9].

Our goal, however, is detecting and identifying objects (*i.e.*, assigning a class label) of *different* classes. One possible way to solve this problem is to build M different "single class Viola-Jones" (SC-VJ) cascades, one to detect objects of each class, then run them *all* at performance time to detect and identify objects of multiple classes. However, this does not scale up well; it requires running one cascade for each class of objects and is therefore expensive. Moreover, it can be ambiguous if more than one

A. Leonardis, H. Bischof, and A. Pinz (Eds.): ECCV 2006, Part I, LNCS 3951, pp. 352–364, 2006.

classifier labels a instance as positive. Another approach is to build one *many-class cascade* of classifiers and use it to detect objects of multiple classes. That is, let $T = T^+ \cup T^-$ be a training set images of positive examples (T^+) and negative examples (T^-), such that $T^+ = \cup_{i=1}^{M} T_i$ where T_i has images of class i and T^- does not have any images of any of the M classes. We can run the Viola-Jones algorithm on this set and produce N binary classifiers, such that each classifier can detect objects of *any* of the M classes (with a certain false positive rate) as a positive instance, but cannot assign a class label. We refer to each of these *binary* classifiers as a "many-class classifier" or, "MC-classifier". This approach has two problems: (1) Since MC-classifiers themselves are binary, during performance, they just label any object in T^+ as positive, but they cannot assign a more specific class label to it. (2) A single MC-classifier, built using objects of different classes as positive examples, can have a high false positive rate. This is not surprising: Many of these individual classes will naturally correspond to disjoint clusters (see below), and this MC-classifier corresponds to their *union*. Any algorithm that attempts to form a convex hull around such disjoint clusters is likely to include many extraneous instances.

In this paper, we present a "multi-class detector and identifier" that is built using MC-classifiers and several single-class cascades. We show that our *many-class detection algorithm* (MCDA) takes much less time than running M class-specific cascades, one for each of the M classes. We also show empirically that the accuracy of MCDA, in detecting and categorizing $M = 4$ diverse classes of objects, is similar to the detection rate of the class specific SC-VJ cascade.

Section 2 motivates and summarizes our framework. Section 3 explains the details of how we build our learning system and how we use it to detect objects of many classes. Section 4 provides empirical results in detecting objects of four classes and discusses how they compare with the SC-VJ cascade detection method, with respect to accuracy, efficiency and ROC curves. Section 5 discusses relevant work related to our research.

2 Motivation and Framework

The Viola-Jones learning algorithm "VJ" takes as input a set of images that are each correctly labeled as either a face or a non-face, and produces a cascade of boosted classifiers. Every classifier consists of many "linear separators", each built using one "rectangle feature", that is a rectangular sub-region in the (24×24 pixel) training images. The algorithm uses three kinds of rectangle features, each using rectangular regions of the same height and width adjacent to each other: (1) a two-rectangle feature (see Figure 1) that computes the difference between the sum of the intensities of the pixels of two adjacent rectangular regions; (2) a three-rectangle feature that computes the sum within two outside rectangles subtracted from the sum in a center rectangle; and (3) a four-rectangle feature that computes the difference between the diagonal pairs of rectangles. There are many (over a hundred thousand) possible combinations of rectangle features each of which can potentially be used as input for a linear separator. The learning algorithm chooses the best linear separators (those that can best separate faces in training data from non-faces, like the region across the mouth and nose; see the human face on the left in Figure 1) from these candidates, which are then used to build classifiers.

Let C_i be any classifier based on k linear separators. C_i classifies any sub-image W of a test image as a face if $\sum_{i=1}^{k} \alpha_i \cdot c_i(W) \geq \frac{1}{2} \sum_{i=1}^{k} \alpha_i$ where α_i is the weight[1] given to i^{th} linear separator c_i and $c_i(W)$ is the boolean classification result of c_i on W as a face or non-face (see [17] for details). We refer to the quantity $V_j(W) = \sum_{i=1}^{k} \alpha_j \cdot c_j^i(W)$, as the *SCO*-value ("sum of classifier output values") of C_i on W.

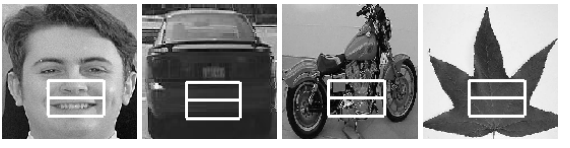

Fig. 1. Features of different classes on a rectangular region

VJ can be used to detect objects of classes other than faces. We use VJ to build N MC-classifiers using a training set of M different classes of classifiers, *i.e.*, $T = T^+ \cup T^-$ where $T^+ = \cup_{i=1}^{M} T_i$. Each of the N classifiers can detect objects of M classes (but cannot assign a class label). We define its "classifier space" as the N-dimensional space formed by using the *SCO*-value of each of N MC-classifiers as a dimension. That is, the N classifiers collectively map each input image to a point in the N-dimensional classifier space. We anticipate that the *SCO*-values of objects in a single class should be similar, and that objects from different classes should have different *SCO*-values. Our results show that this holds — in that each class will form a "cluster" in the classifier space; see Figure 2. For each cluster, we can assign the class label ℓ based on the number of images of each class in the cluster; see Section 3.1 for details.

Figure 2 shows various clusters of four classes of objects — cars, leaves, motorbikes and faces — plotted using the *SCO*-values of 2 of the MC-classifiers, on training images of these four classes of objects.[2]. We selected the classifiers shown in Figure 2 $(C_1, C_2, \ldots, C_8)$ manually to clarify our ideas. Of course, we do not anticipate that the *SCO*-values of *every* pair of MC-classifiers (or for that matter every set of $k \leq N$ classifiers) will form clusters.

Note that one *subset* of the N classifiers may be sufficient to distinguish class#1 from class#2; here, it would clearly be inefficient to consider all N classifiers. Unfortunately, a different subset may be necessary to separate class#1 from class#3, and a third subset for class#2 vs class#3, and so forth. There may be no small set of classifiers that is sufficient distinguish each class of objects from the others. That is why we use a dynamic process to find the most appropriate subset of classifiers: For each input image, this process sequentially decides which classifier to use next, based on the values observed from the classifiers previously executed on this window. The challenge is to *learn* the dynamic sequence of classifiers that can effectively distinguish the clusters corresponding to different classes.

[1] The weights for linear separators are learned during training.

[2] We presented clusters in two dimensions for clarity; in general there may be clusters in a p-dimensional space for $2 \leq p \leq N$.

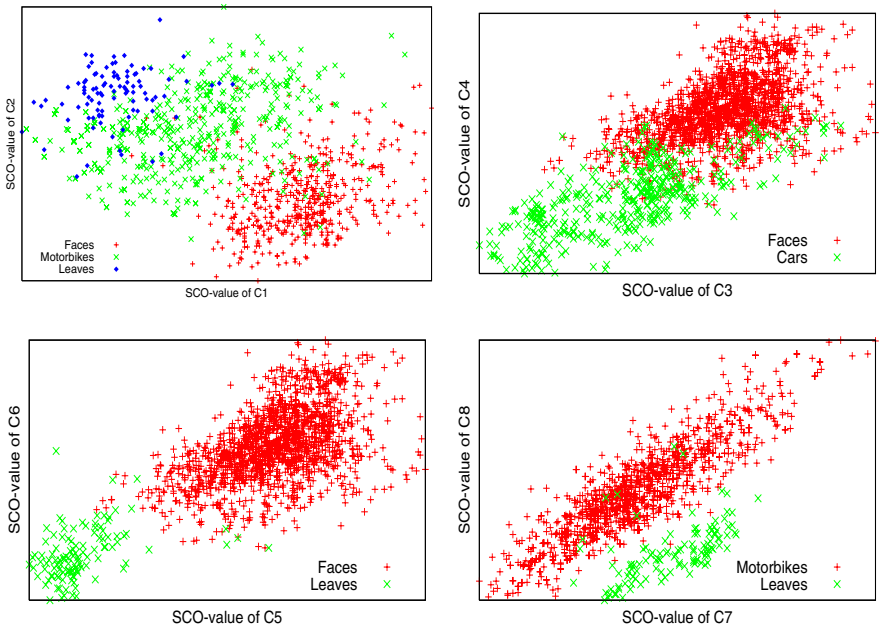

Fig. 2. Clusters of objects of the same class in cascade space

We use dynamic programming to find a sequence of MC-classifiers that optimally partition the training images into clusters. Our learning algorithm therefore builds a depth-d "decision tree classifier" (DTC, see Figure 3) that attempts to identify the appropriate label for each instance, using the learned MC-classifiers as features. Each DTC leaf, corresponding to one class of objects, also includes a further VJ cascade of classifiers, to verify an instance qualifies.

State representation: We can identify each node n in DTC with a state s. We represent s with the *SCO*-values of the classifiers on the path from the root of DTC to n (see Figure 3). That is, $s = \langle [V_{min,1}, V_{max,1}], \ldots [V_{min,k}, V_{max,k}] \rangle$, where for each i, $[V_{min,i}, V_{max,i}]$ is the range of *SCO*-values of C_i. We say two states are "δ-equivalent", written $s_1 \approx_\delta s_2$, iff

- s_1 and s_2 have applied the same set of classifiers, not necessarily in the same order
- For every classifier C_i used in s_1 and s_2, $|V^{(1)}_{min,i} - V^{(2)}_{min,i}| \leq \delta$ and $|V^{(1)}_{max,i} - V^{(2)}_{max,i}| \leq \delta$, where δ is a pre-defined constant. We set $\delta = 70$ in this work.

We use the equivalence property of states for two reasons: (1) during training, to merge all δ-equivalent states into one, and (2) during performance, to find the closest matching state from the training results and use the best classifier associated with it.

At run time, to classify a sub-image W within the current test image, MCDA basically follows DTC: it dynamically selects a classifier to apply to W, based on the responses of the previously run classifiers on W. If all the classifiers on the path from the root to

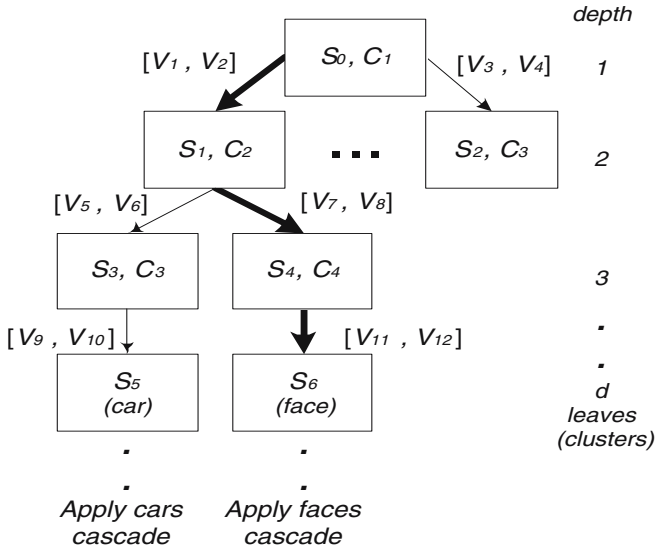

Fig. 3. Decision tree classifier. Each node is associated with a state, and every internal node with a particular classifier. Each leaf represents a cluster (*i.e.*, a single class), and has an associated cascade.

the leaf of DTC label W positively, we classify W with the class label ℓ of the corresponding leaf. We then apply a cascade, specific to this leaf, to W, to confirm that W is an instance of class ℓ. If any of the classifiers in DTC or the class specific cascade label W negatively, we stop processing W and proceed to the next sub-image. We can summarize the framework as follows:

- Use training data $T = T^+ \cup T^-$ to build N many-class boosted (binary) MC-classifiers, each designed to classify objects of any of the M classes as positive instances. (Note this does not distinguish these different classes).
- Use dynamic programming to build a DTC using these N MC-classifiers as binary "features", where each leaf corresponds to a cluster of a single class.
- For every leaf in DTC, find class label ℓ (explained in Section 3.1) and assign a cascade C of class ℓ to the leaf.
- At performance time, scan through each sub-image W of the test image. For each W, follow the decision tree DTC. If any of the classifiers encountered label W as negative, stop. Otherwise, if all label it as positive, tentatively assign W the class label ℓ associated with the leaf reached. Run the cascade associated with this leaf to confirm W is an instance of class ℓ.

We can contrast this approach with the other obvious algorithm for detecting M classes of objects: just use M class-specific cascades, each having N classifiers. We will call this "M-SC-VJ". This means classifying each instance would require running $M \times N$ classifiers. As our detection method chooses classifiers carefully in the first stage, we can assign a tentative class label using only $p \leq M$ classifiers (see clusters in Figure 2), then run *one* length-N cascade. Hence, we need only run a total of at most

$M + N$ classifiers for each image, which is clearly more efficient.[3] Section 4 presents empirical results confirming this.

3 Learning to Detect Objects of Many Classes

This section presents the details of using dynamic programming to construct a DTC of MC-classifiers, and then how we use this DTC to detect objects of many different classes.

3.1 Building DTC Classifier

Figure 4(a) presents the learning algorithm. We build N MC-classifiers using the images of T. We then produce a depth d decision tree (DTC) using these N classifiers.

Exploring sequences of classifiers: We explore every possible sequence of d MC-classifiers on the images of T^+ to find the sequence that yields the best clusters. That is, we first apply any MC-classifier C_1 on each image $w \in T^+$. We ignore all the images that C_1 labels as negatives. We sort the remaining images based on their *SCO*-values $V_1(w)$ of C_1 on these images, *i.e.*, $\langle w_1, \ldots, w_{m/2}, w_{m/2+1}, \ldots, w_m \rangle$, where $V_1(w_j) > V_1(w_k)$ when $j > k$. We split them into two equal halves $\langle T_1^L \rangle$ and $\langle T_1^R \rangle$ (denoting the left and right branches), such that $\langle T_1^L \rangle$ contains $\{w_1, \ldots, w_{m/2}\}$ and $\langle T_1^R \rangle$ contains $\{w_{m/2+1}, \ldots, w_m\}$. We then apply another MC-classifier $C_2 \neq C_1$ on $\langle T_1^L \rangle$ and $\langle T_2^L \rangle$ separately resulting in (1) $\langle T_1^L, T_2^L \rangle$ and $\langle T_1^L, T_2^R \rangle$ that each represents one half of the classifiers of $\langle T_1^L \rangle$ that C_2 labeled as positives (2) $\langle T_1^R, T_2^L \rangle$ and $\langle T_1^R, T_2^R \rangle$ that each represents one half of the classifiers of $\langle T_1^R \rangle$ that C_2 labeled as positives.

We repeat the process for d steps, applying a sequence of d classifiers, $\langle C_1, \ldots, C_d \rangle$. The resulting 2^d leaves are clusters. Note that this is for one (random) sequence of d classifiers. When we consider $\binom{N}{d}$ different sequences of d classifiers, it leads to a total of $\binom{N}{d} \times 2^d$ clusters. While many clusters can have the same class, this is not be a problem. Any sub-image will now be matched to one of the smaller clusters. In general, we identify each node at depth i with a state s that is a list of ranges of *SCO*-values, of the form $\langle [V_{min}^1, V_{max}^1], \ldots, [V_{min}^i, V_{max}^i] \rangle$.

Computing the utilities of clusters: We want to determine the best decision tree within this tableau — the one that leads to the "purest" leaf nodes. Each leaf of the tree represents a cluster. We want the clusters that are as "pure" as possible, *i.e.*, which group images of only one class together. For every cluster s_d, we compute the probability that images in s_d are of class i, weighted by the size of class i:

$$U(s_d) \quad = \quad \max_i \frac{P(\, T_i \,|\, s_d \,)}{|T_i|}$$

where of course $P(\, T_i \,|\, s_d \,)$ is the fraction of images of class i in s_d and $|T_i|$ is the number of images in training set T_i. Basically, we are computing the fraction of images

[3] While some classifiers are more expensive to apply than others, the difference in the cost of applying any two classifiers is negligible compared to the overall cost. So, running $M + N$ classifiers is better than running $M \times N$ classifiers.

Learn_DTC($T = T^+ \cup T^-$: TrainingSet)
- Build N MC-classifiers$\{C_1, C_2, \ldots C_N\}$ using images in T.
- Let All be the set of all $\binom{N}{d}$ sequences of d of MC-classifiers
- For every sequence of d classifiers $\langle C_1, \ldots, C_d \rangle$ from All
  - Let $\langle T_0 \rangle$ be set of all (+)images, *i.e.*, T^+
  - For i = 1 to d do
    - Apply C_i separately on each partition $\langle T_{i-1} \rangle$ produced by applying $\langle C_1, \ldots, C_{i-1} \rangle$
    - For each partition $\langle T_{i-1} \rangle$, remove all images that C_i labels as negatives
    - Split the rest of images into two (left and right) partitions
  - For each cluster s_d produced by applying the sequence of d classifiers
    - Assign maximum probable class label ℓ
    - Compute utility $U(s_d)$
  - For every state s_i resulting after applying $i < d$ classifiers
    - Compute utility $U(s_i) = max_j U(s^j_{i+1})$
    - Let C^*_i be associated classifier that yielded max. utility when applied in s_i
    - Associate C^*_i with s_i, store $\langle s_i, C^*_i \rangle$
- Merge all the δ-equivalent states into one, store one classifier C^*_i with the max. utility
- The resulting $\langle s_i, C^*_i \rangle$ for $1 \leq i \leq d$ is DTC

MCDA (I_t : Test Image)
- $\star$ For each window W (of 24×24 pixels) within I_t
  - For $1 \leq i \leq d$,
    - Find state s_i "closest" to W,
    - Apply the C^*_i associated with s_i, to W
  - If all of $\langle C_1, C_2, \ldots C_d \rangle$ label W as a positive
    - Find the corresponding cluster, *i.e.*, s_d,
    - Assign the mostly likely class ℓ associated with s_d, to W
    - Apply SC-VJ cascade $\langle C^\ell_1, C^\ell_2 \ldots C^\ell_P \rangle$ to W
    - If class ℓ cascade also labels W as a positive, mark W as an instance of class ℓ
- Resize I_t by a factor of 0.8
- If I_t.length ≥ 24 and I_t.width ≥ 24, goto $\star$
- Return all windows marked as positive instances with class labels (and correct sizes)

Fig. 4. (a) Learning algorithm to produce DTC; (b) Dynamic classification algorithm

of each class i and normalizing it with the fraction of images in the training set, so that any class that has a high number of training images does not have an unfair advantage. We assign the class label ℓ that has the maximum utility to s_d. Recall this is after applying any sequence of d classifiers, $\langle C_1, C_2 \ldots C_d \rangle$. The idea is to assign high utility value to clusters that group images of the same class together. For any state s_i with possible "children" $\{s^j_{i+i}\}_j$, (each corresponding to the application of one other classifier), we compute the utility, $U(s_i) = \max_j \{U(s^j_{i+i}\}$ *i.e.*, the maximum utility of any state produced by applying an additional $i + 1^{st}$ classifier.

Building DTC**:** We collect the $\langle s_i, C^*_i \rangle$ tuples and also the corresponding utilities, for all i, $1 \leq i \leq d$, where s_i denotes the state resulting after applying i classifiers, C^*_i

denotes the classifier that, when applied to s_i, transitions it to another state s^*_{i+1}, with the maximum utility among the states resulting after applying $(i+1)$ classifiers. For every two states s_i and s_j $(i \neq j)$ that are δ-equivalent we retain only one state that has higher utility and the corresponding classifier. Note that the $\langle s_i, C^*_i \rangle$ tuples for $1 \leq i \leq d$ tell us precisely the i classifiers applied so far, their individual *SCO*-values and the best classifier C^*_i to apply in s_i. This corresponds precisely to the DTC decision tree.

3.2 Detection

Our detection algorithm, MCDA shown in Figure 4(b), uses classifiers built by the cascade classifiers method [16, 18]. It examines each 24×24 pixel window in the image, then rescales by a factor 0.8 (*i.e.*, resizes the current height and width of the test image by a factor of 0.8) and repeats. For each window W, DTC first applies the classifier C^*_1 associated with root (see Figure 3). This might reject W; if so the process terminates (*i.e.*, DTC continues with the next window). Otherwise, DTC computes the *SCO*-value associated with C^*_1 on W and uses this value to decide which subsequent classifier C^*_2 to apply. Again this could reject W, but if not, C^*_2's *SCO*-value identifies the next classifier C^*_3 to apply to W. This can continue for at most d steps, until W reaches a leaf (cluster). If all the d classifiers label W as a positive instance, DTC finds the class label ℓ associated with the cluster. We then run the SC-VJ cascade $\langle C^\ell_1, C^\ell_2, \ldots C^\ell_P \rangle$ associated with this leaf (of class ℓ) and declare W to be an object of class ℓ only if it passes all of these classifiers. Otherwise, we reject it as a negative instance.

4 Experimental Results

4.1 Experimental Setup

Data Used: We used four classes of objects in our experiments: faces, cars (rear view), leaves and motorbikes. We used a total of 1600 images of faces, collected from popular face image databases (including ones from Olivetti Research and AT&T, PIE, UMIST, Yale, etc.) in the training set of faces, T_F. We used the entire MIT-CMU database of faces, which has a total of 178 images with 532 faces, as the test set for faces. For the other three classes (cars, motorbikes and leaves), we used images from Caltech image database [1]. We split the 526 images of cars into two random sets of 476 and 50 images. We used the first set as the training set for cars, T_C, and the second set (with a total of 67 cars) as a test set. Similarly we split the 826 images of motorbikes into two random sets of 776 and 50 images and used the first set as the training set for motorbikes T_M and the second set of 50 images (with a total of 50 motorbikes) as the test set. Caltech database uses three different types of leaves (see Figure 5) and has a total of 186 images of leaves. We split this into two random sets of 156 and 30 images and used the first set as the training set for leaves T_L and the remaining set of 30 as the test set. We also used another 37 images of leaves (that we captured using a digital camera, with various backgrounds and sizes) in the test set. So, our entire test set for leaves has a total of 67 images, one leaf per image. Our training set for the negative examples, T^- has a total of 2320 images; none of these has any pictures of faces or cars or leaves or motorbikes.

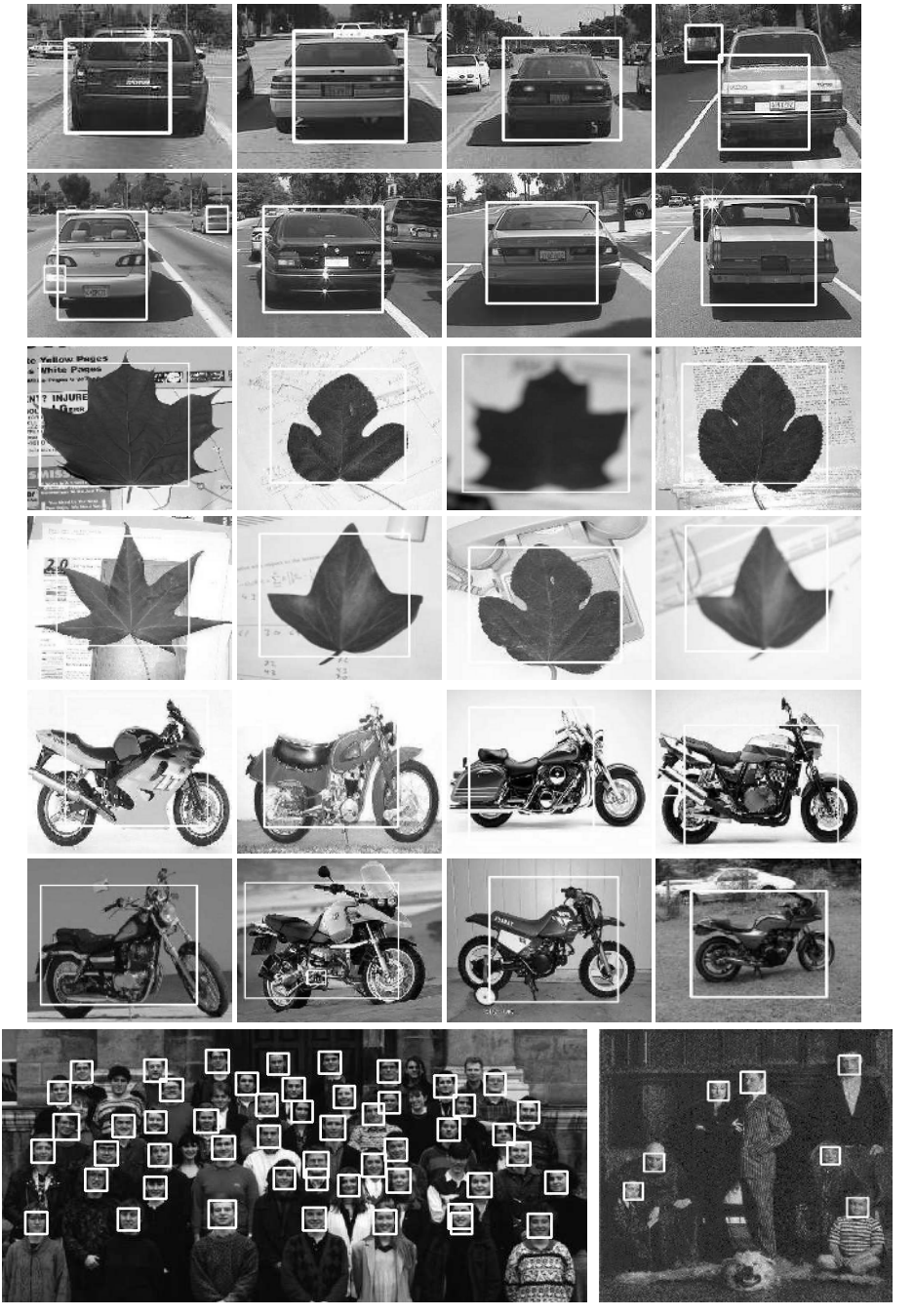

Fig. 5. Performance on test images of cars (rear view), leaves, motorbikes (side view) and faces

Building VJ Classifiers: We used T_C, T_L, T_M, T_F and T^- to build 4 SC-VJ cascade classifiers (one for each class), that involved 18, 17, 17 and 21 classifiers for cars, leaves, motorbikes and faces, respectively. We also built $N = 10$ MC-classifiers that can detect objects of any of the four classes. Since we have four different classes, and with the application of each classifier (carefully, using DTC) we can distinguish between two classes, we set $d = 3$.[4] That is, we built a DTC upto a depth of 3 using our learning algorithm as explained in Section 3.1.

Training time: Our system required about 3 hours to build each of the 4 class specific cascades and another 1 hour to build the MC-classifiers[5]. It then required about 5 minutes to build DTC, so the total training time was approximately 18 hours.

Results: We compared MCDA to the standard set of $M = 4$ SC-VJ cascades, with respect to accuracy, ROC curves and efficiency. Note that MCDA applies d MC-classifiers (within DTC) to determine which class label to consider for each test sub-image, and then applies a cascade specific to that class. The SC-VJ detection algorithm has an easier task, as we explicitly identify which single class of objects it should seek for each image, which means it does not need to apply any MC-classifiers. This is why we do not expect the performance of MCDA to be better than SC-VJ, in terms of either efficiency or accuracy. However, our results indicate that MCDA does quite well in detecting objects as well as assigning class labels. In fact, our algorithm runs at least twice as fast as running M VJ cascades to detect $M = 4$ classes of objects; see Section 4.4.

4.2 Accuracy and Execution Time

Figure 5 shows some test images in which MCDA could successfully detect cars (rear view), leaves, motorbikes and faces. Table 1 compares MCDA with the SC-VJ cascade algorithm in terms of accuracy and efficiency. The peak accuracy, as we vary the number of cascade classifiers at the leafs,[6] for SC-VJ and MCDA are given in Table 1. These values are statistically indistinguishable at $p < 0.05$. While MCDA is slower than SC-VJ, by an additive 63%, 83.7%, 67.7% and 22.26%, we attribute this to: (1) the time needed to run the extra $d = 3$ classifiers using DTC and (2) the overhead involved in assigning a class label to each sub-image of any test image. Note that this is much better than the obvious M-SC-VJ alternative.

4.3 Number of Class-Specific Classifiers

We ran the following experiment to determine how these two approaches (MCDA and SC-VJ) each scale with the number d' of class-specific classifiers: We first applied $d = 3$ classifiers within DTC, then applied d' class-specific classifiers at each leaf, varying d' in the ranges [10–18], [10–17], [10–17] and [12–21] for cars, leaves, motorbikes and

[4] We tried larger values of d, but the results were not any better.

[5] 1. We use the Wu and Rehg [18] implementation of VJ.
2. All results presented here were run on a 1 GHz. Intel Pentium processor with 256 Mbytes of memory running Windows-2000.

[6] We define "peak accuracy" as the accuracy value with negligible rate of increase with increasing values false positives.

Table 1. Comparison of test results for SC-VJ cascade, MCDA and MSC-VJ algorithms

Class	TestData			#Windows	Peak Accuracy		Av.Detcn.Time(sec)		
	#Images	#Objects	Av.Image Size		SC-VJ	MCDA	SC-VJ	MCDA	M-SC-VJ
Cars	50	65	265× 360	10,114,613	87.69%	86.15%	0.495	0.806	1.787
Leaves	67	67	318× 436	20,607,663	97.01%	95.52%	0.454	0.834	2.006
Motorbikes	50	50	279× 297	8,680,218	97.0%	92.0%	0.574	0.963	1.912
Faces	169	532	403× 402	76,957,710	92.11%	92.0%	1.541	1.883	4.558

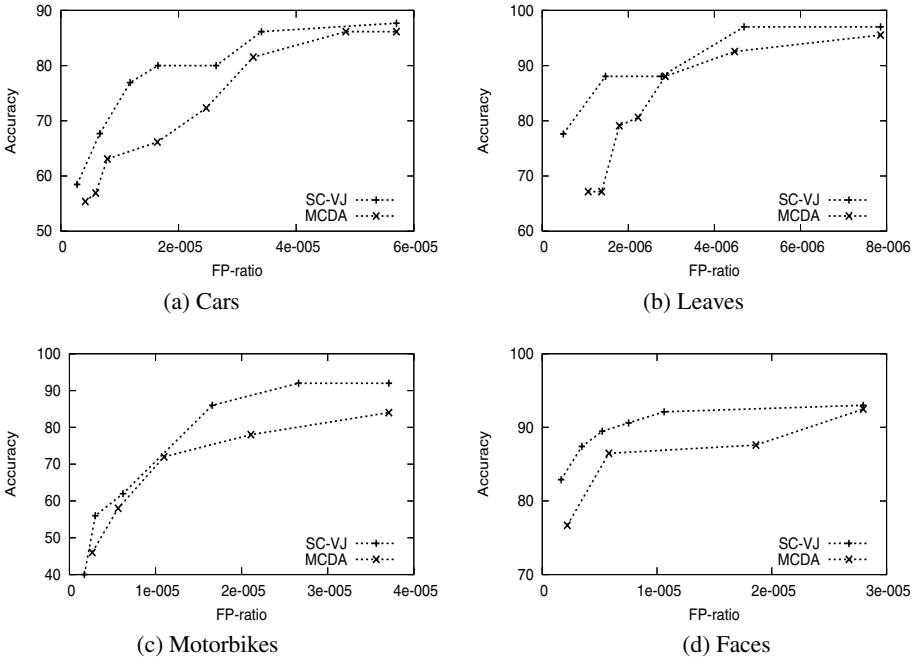

Fig. 6. ROC curves for SC-VJ cascade detection and MCDA for (a) Cars (rear); (b) Leaves; (c) Motorbikes; and (d) Faces

faces, respectively. For each value of d' we recorded the number of false positives and accuracy, as well as the total number of windows (24×24 pixel sub-images) processed. We also did this for SC-VJ. Each graph in Figure 6 plots accuracy against the number of false positives per window processed. We see that SC-VJ detection method performs better than MCDA, while the overall detection for MCDA is comparable to SC-VJ.

4.4 Comparison to M-SC-VJ

On the test set of each class, we ran each of the four cascade classifiers and recorded the execution time; see Table 1. As the execution time of this algorithm is linear in the number of classes, it does not scale as well as MCDA, which does not need to run multiple cascades.

5 Related Work

There has been a lot of interest in multiclass object detection recently. Torralba *et al.* [14] train binary classifiers "jointly" (for several classes) and use the common features to detect objects of the various classes. They show that feature sharing is a fundamental aspect of multiclass detectors that scale up well with the number of object classes. In our work, we use MC-classifiers that use features of multiple classes of objects to detect member of these classes. Different classes of objects have different features, *i.e.*, a classifier's *SCO*-value is different for different classes of objects. Using the *SCO*-values of the first d classifiers of DTC, we assign a class label to any sub-image of a test image. That is, using *SCO*-values, we reach some partition of the feature space where a single object class exists. Hence, our work implicitly utilizes feature sharing. But our learning and detection algorithms are significantly different. Fan [2] presents an algorithm that learns a hierarchical partitioning of the hypothesis space, which they use to do a coarse to fine search in the hypothesis space, pruning groups of hypotheses at every stage. Li, Fergus and Perona [6] use a generative probabilistic model to represent the shape and appearance of a constellation of features of an object. They learn the parameters of the model incrementally in a Bayesian manner. They test it on 101 different object categories. Lin and Liu [7] argue that face detection itself is a multiclass detection problem because of the variations in the appearance of a face caused by different poses, lighting conditions, expressions, occlusions, etc. They present a boosting algorithm to detect faces to account for all these variations. Our work is different from this as we try to assign a class label based on the clusters, and then detect objects using class specific cascades. We addressed related issues in a feature-based face-recognition system [5] by posing the task a "Markov Decision Problem (MDP)". We use dynamic programming to produce an optimal policy π^*, that maps "states" to "actions" (feature detectors) for that MDP, then used that optimal policy to recognize faces *efficiently*. We use similar techniques here in this work, as we again find the best sequence of classifiers. The current work differs because it considers *multiple* classes of objects.

6 Conclusions

This research provides a way to use learned binary classifiers to detect and identify objects in diverse classes. We first observe that images of each class can form clusters in the classifier space of MC-classifiers, and that different subsets of MC-classifiers may be sufficient to distinguish different pairs of classes. Hence, an efficient approach should select these classifiers dynamically. We present a learning algorithm that produces a decision tree, DTC, that first applies a dynamic sequence of classifiers to propose a possible class label for each sub-image of a test image, then applies a cascade of classifiers, specific to that class, that is effective for pruning away false positives.

We present empirical results to demonstrate that our approach is effective. In particular, we show that our implementation can detect and identify objects belonging to any of $M = 4$ classes, obtaining roughly the same accuracy and ROC-curve performance as the naive approach of simply running M different VJ systems. Moreover, our approach is about as fast as running a *single* VJ cascade, and will scale well as the number of object classes grows.

Acknowledgments

The authors thank Jianxin Wu and Jim Rehg for making the cascade detection code available. They also thank Caltech computational vision group and members of many other groups (including PIE, MIT/CMU, UMIST, Yale, Olivetti Research and AT&T) for making their image databases available.

References

1. R. Fergus, P. Perona and A.Zisserman Object class recognition by unsupervised scale-invariant learning In *CVPR*, 2003, pp. 264–271
2. X. Fan Efficient multiclass object detection by a hierarchy of classifiers In *CVPR*, 2003
3. Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Computational Learning Theory: Eurocolt*, 1995.
4. E. Grossmann. Adatree: boosting a weak classifier into a decision tree. In *IEEE Workshop on Learning in Computer Vision and Patter Recognition*, 2004.
5. R. Isukapalli and R. Greiner Use of Off-line Dynamic Programming for Efficient Image Interpretation *IJCAI*, Acapulco, Mexico, Aug 2003
6. F.F. Li, R.Fergus and P. Perona Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories In *Proceedings of the Workshop on Generative Model Based Vision*, Washington, D.C., June 2004
7. Y. Lin and T. Liu Robust face detection with multi-class boosting In *CVPR* 2005, pp.680-687
8. C. Liu and H.Shum, Kullback-Leibler Boosting In *CVPR* 2003, pp.587–594
9. E-J. Ong and R. Bowden A boosted classifier tree for hand shape detection *International Conference on Automatic Face and Gesture Recognition*, 2004, pp.889–894
10. H. Rowley, S. Baluja and T. Kanade. Neural network-based face detection. *IEEE Transactions on Patten Analysis and Machine Intelligence (PAMI)*, 1998.
11. D. Roth, M. Yang and N. Ahuja. A snowbased face detector. In *NIPS*, 2000.
12. H. Schneiderman and T. Kanade. A statistical method for 3d object detection applied to faces and cars. In *ICCV*, 2000.
13. J. Sun, J.M. Rehg and A.Bobick Automatic cascade training with perturbation bias In *CVPR*, 2004
14. A. Torralba, K. Murphy and W.T. Freeman Sharing features: efficient boosting procedures for multiclass object detection In *CVPR*, 2004.
15. P. Viola and M. Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. In *CVPR*, 2001.
16. P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.
17. P. Viola and M. Jones. Robust real-time face detection. In *IJCV*, 2004.
18. J. Wu, J.M. Rehg and M.D. Mullin. Learning a rare event detection cascade by direct feature selection. In *NIPS*, 2003.