

Learning a Dynamic Classification Method to Detect Faces and Identify Facial Expression

Ramana Isukapalli¹, Russell Greiner², and Ahmed Elgammal³

¹ Lucent Technologies, Bell Labs Innovations, Whippany, NJ 07981, USA

² University of Alberta, Edmonton, CA T6G 2E8, CA

³ Rutgers University, New Brunswick, NJ 08854, USA

Abstract. While there has been a great deal of research in face detection and recognition, there has been very limited work on identifying the *expression* on a face. Many current face detection projects use a [Viola/Jones] style “cascade” of Adaboost-based classifiers to interpret (sub)images — e.g., to identify which regions contain faces. We extend this method by learning a *decision tree* of such classifiers (DTC): While standard cascade classification methods will apply the same sequence of classifiers to each image, our DTC is able to select the most effective classifier at every stage, based on the outcomes of the classifiers already applied. We use DTC not only to detect faces in a test image, but to identify the expression on each face.

1 Introduction

The pioneering work of Viola and Jones [12] has led to a host of face detectors based on “cascade classifiers”, where each classifier is learned by applying Adaboost [4] (or some related algorithm [11,13]) to a database of training images of faces and non-faces. The underlying principle in all these algorithms is to learn multiple classifiers during the training phase, then (at performance time), run these classifiers as a “cascade” on each region (at various resolutions) of the test image — *i.e.*, in a sequence one after another, eliminating non-faces at each stage.

In general, there can be many *sub-clusters* within the class of face images — in particular, perhaps one sub-cluster corresponds to people with the same facial expression while another corresponds to people with some other expression.

Moreover, one of our learned classifiers might do very well on one cluster, but relatively poorly on another. Consider, for example, the examples shown in Figure 1, and notice the positive instances can be grouped into 3 clusters. (Here, imagine every instance labeled “+” corresponds to a HappyFace, “◇” to a SadFace and “□” to an AngryFace”. Note all 3 are faces — *i.e.*, should be labeled positively by a face detector.) Now consider two possible classifiers, corresponding to the set of separating lines labeled C_1 (respectively C_2). Neither is perfect. If we are trying to separate only the “+” labeled positive instances from the negative instances, we would get better results using C_1 ; but if dealing with “◇” labeled or “□” labeled instances, then C_2 would be preferred. Of course, this same idea holds for different possible *cascade* of classifiers: different cascades might be preferable for different clusters of images.

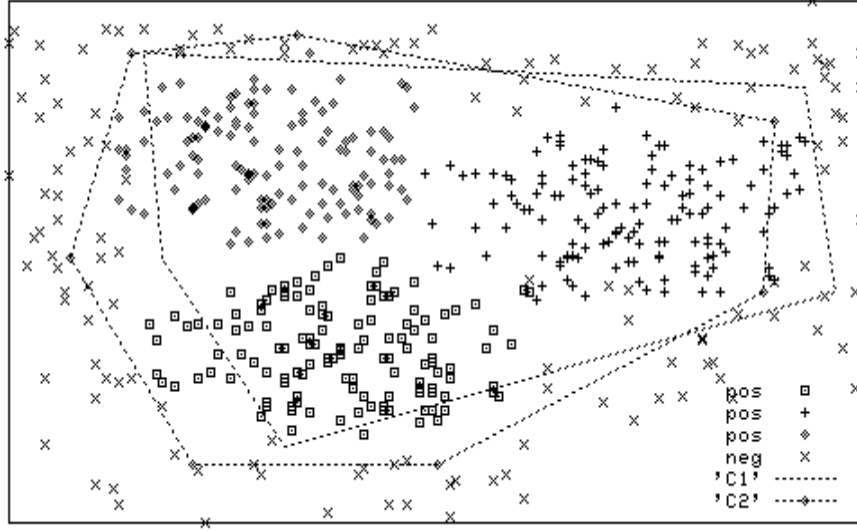


Fig. 1. Two classifiers on set of positive and negative examples. Note various sub-clusters of positive examples.

At performance time, our performance system DTC will scan through an unlabeled image. For each sub-image W , it will (1) quickly sort W to the cluster best able to process this sub-image, and (2) apply to W the classifier cascade appropriate to this cluster. If W passes all of those classifiers, it is declared a “face”, and moreover assigned the expression E associated with this cluster. The challenge is learning these two sub-procedures. To do this, we first run the [12] algorithm on a training set whose image regions are labeled only as face (vs non-face), but without any expression data. This produces a set of Adaboost classifiers $\mathcal{C} = \{C_1, \dots, C_N\}$. We then use these C_i s as features, to produce a decision tree that attempts to partition a second labeled training set, which is a collection of images whose faces are each labeled with an expression (as well as non-faces), into sub-collections containing faces with the same expression. At each leaf node of this decision tree we then assemble a sequence of classifiers (each taken from the \mathcal{C} set) to form a cascade appropriate to this cluster. (Note each cluster can have its own classifier sequence.) By construction, the resulting face detection system will not only detect faces, but also associate every such face detected in a test image with an expression.

This framework has two advantages over the standard model of cascade classifiers. First, the standard model can only assign a binary “face vs non-face” label to each subimage. By contrast, our system can return several labels, corresponding to facial expression (as well as “non-face”). Second, while one classifier (or cascade of classifiers) might work well in identifying images in one cluster, that classifier (cascade) might not work well for another. Unfortunately the existing cascade-based detection techniques are *static* and *image-independent*, in that they apply a fixed set of classifiers in a fixed order to *any* image. By contrast, our *dynamic classification technique* DTC can decide which classifier to use at any stage, based on the outcomes of the previous

classifiers. That is, while DTC also applies a sequence of classifiers to detect faces, the actual sequence can vary — hence it can apply different cascades of classifiers to different images. Our evidence suggests that such dynamic classifiers have slightly better accuracy in detecting faces.

Our DTC corresponds to an *augmented decision tree*, whose internal nodes each correspond to a version of a classifier (which computes a real-valued score given an image) and whose arcs each correspond to a range of those values. Each leaf node corresponds to a partitioning of the possible sub-images; we then identify with each such leaf node both a further sequence of classifiers and also a specific facial expression. Note that each sub-cluster (and hence the leaf node) identifies the facial expression. We also identify a sequence of classifiers with each leaf node. To classify a test (sub)image I , DTC will apply the classifier associated with root of the decision tree to I , then use the outcome of that classifier to determine which subtree to explore. It then recurs: applying the classifier at the root of that subtree, etc. (Hence, the classifier applied at stage s depends on the outcomes of the classifiers at stage $1, 2, \dots, s-1$.) On reaching a leaf node ℓ , DTC will then apply the associated classifier cascade, and accepts I as a face if all of the classifier agree with this claim. If I is labeled as a face, it is also assigned ℓ 's facial expression.

Section 2 describes how we produce this DTC system from a collection of labeled images. In particular, this section shows how we use dynamic programming and abstractions to sidestep the serious combinatorial issues — e.g., the exponential number of possible decision trees, and possible cascades.

This section also explains the details of how we build a DTC and how we use it to detect faces and identify the expressions on each detected face. Section 3 evaluates our work in the domain of face detection; Section 4 presents relevant work related to our research.

2 Learning the DTC Classifier

This section describes how we learn our DTC from two sets of labeled images — one training set FLT¹, that identifies the embedded faces (but not the expressions), and a second smaller one, FELT², that labels the embedded faces with the associated expressions. Each face image in the FELT set contains a single face that is hand-labeled with one of { sad, fear, surprise, happy, no_expression }.

There are three steps to building the DTC classifier. We first use the standard approach [12] to build a cascade of Adaboost classifiers $\mathcal{C} = \langle C_1, \dots, C_N \rangle$, based only on the FLT training set; see Section 2.2. We next use the FELT data to produce a fixed-depth decision tree whose nodes each correspond to one of these C_i s, with the goal of partitioning the FELT set into sub-clusters of face images with similar expressions; see Sections 2.3. Each leaf node is now identified with a single expression. Third, we add a further fixed-size sequence of classifiers, taken from $\{C_1, \dots, C_N\}$, to each leaf node; see Section 2.4.

¹ Face-Labeled Training set.

² Face-Expression-Labeled Training set.

Later, to use DTC to classify some region of an image, we first drop that region into the decision tree, and let it sort itself down to a leaf. We then run the associated sequence of classifiers. If that image “passes” each member of that sequence, it is labeled as a face, and given the expression label associated with the leaf node. Otherwise it is declared a non-face; see Section 2.5.

We give the motivation to our approach first and then present the details of our algorithm.

2.1 Motivation

Figure 1 shows two classifiers C_1 and C_2 that are each, independently, trying to separate positive from negative examples; each line of C_1 (resp., C_2) denotes a linear separator, whose intersection corresponds to the classifier. At a high level, the positive examples can be approximately grouped into three sub-clusters.

Below, we will consider classifiers that map an image into a real “SCO-value” (defined later), not just a bit; see below. A set of N such classifiers therefore map each image into a point in N -dimensional space, and face images can be seen as forming sub-clusters in this space. If we can partition them meaningfully along “ $d_1 \ll N$ ” (a predetermined constant, see below) dimensions, then we can retrieve the sub-clusters. Partitioning the images based on any single classifier C_i is equivalent to finding its projection on the i^{th} dimension in the N dimensional space; we can then split the images based on the classifier’s value, into two (perhaps equal) groups. If we take such projections and partition the images repeatedly along d_1 dimensions corresponding to d_1 classifiers, then we can retrieve the (at most 2^{d_1}) sub-clusters.

It is also important to choose the d_1 most effective classifiers. The positive instances in Figure 1 include members of a left-most sub-cluster, labeled “ \diamond ” and a right-most one labeled “+”. We can see that the positive instances in these two sub-clusters have different ranges for their X -values, while they have a similar range for the Y -value. This means it is easy to separate the left-most cluster from the right-most if we project the data into the X -axis, but this is not true if we consider projection into the Y -axis.

Of the many ways to partition the images, we wanted the partitions that are associated with common facial expressions, based on the images in the FELT dataset. Section 2.3 shows how we build a fixed-depth decision tree, “DTC”, using the \mathcal{C} classifiers (from the FLT-based cascade) to do this.

Each leaf here corresponds to a single facial expression. We also find a sequence of another d_2 classifiers that are specifically chosen to remove the false positives from the sub-cluster.

2.2 Learning a Cascade of Classifiers

Our implementation first uses Viola-Jones approach [12] to produce a cascade of Adaboost classifiers, $\langle C_1, C_2, \dots, C_N \rangle$: First apply Adaboost to the entire collection of labeled images in the training set FLT to produce a classifier C_1 . Let T_1 be the images that C_1 labeled positively. Then apply Adaboost to T_1 to produce the C_2 classifier; then let C_3 be the classifier Adaboost produces when given T_2 (which are the images C_2 labeled positively), and so forth, to produce N classifiers. In our case we tell Adaboost

how many linear separators each classifier should use. However, the algorithm decides how many classifiers N it needs to build.

Our DTC will use this *set* of classifiers, but will structure them into a decision tree, rather than simply use them in this sequence.

SCO-Value: Each C_i classifier will reject many of the images. For each image it passes, it will also compute a real-value, as follows: Let $\langle c_i^1, c_i^2, \dots, c_i^k \rangle$ be the linear separators³ of the boosted classifier C_i and let $c_i^j(W)$ be the outcome of applying c_i^j to a training image W . We define SCO-value (“sum of classifier output values”), $V_i(W) = \sum_{j=1}^k c_i^j(W)$ as the sum of the outputs of the linear separators of C_i .

Adaboost is designed to choose the best features from the over-a-hundred-thousand possible candidates. They are likely to fall on salient features specific to faces — like eyes, nose, mouth, etc. Since SCO-value uses the outcomes of these classifiers, we anticipate partitioning face images based on these values should group images of similar features into one sub-cluster.

2.3 Building the Basic Decision Tree

Figure 2(a) presents the learning algorithm. It has two goals: first, to partition the images in the FELT set into meaningful sub-clusters of face images with the same face expression, and second, to find the most effective sequence of d_2 classifiers for each sub-cluster.

We first restrict the set of classifiers, paring the list from $\mathcal{C} = \{C_1, \dots, C_N\}$ down to a smaller set. To do this, we view each of these classifiers *individually*, as if we were planning to use only it to label images. For any classifier C_i and for a specified data set of images S (here, we use FLT.) we compute the score

$$R(C_i, S) = \ell \times FN(C_i, S) + FP(C_i, S) \quad (1)$$

where $FN(C_i, S_j)$ is the number of false negatives C_i returns over the set S and $FP(C_i, S)$ is the number of false positives in S . (We set $\ell = 10$ in this work, as false negatives are much worse than false positives — as a subsequent classifier may eliminate the false positive, but once any classifier has removed a false negative, it will never be recovered.) We then use only the best $M = 10$ such classifiers.

Next, we produce a depth- d_1 decision tree, whose features each correspond to one of these M classifiers. Its goal is to partition the FELT data, into clusters with the same facial expression. (We used $d_1 = 3$ here.) In general, achieving this objective can be very expensive, as there are several thousands of training images, and a huge number of possible decision trees. We use a two stage divide-and-conquer approach to help sidestep this. First, we use a dynamic programming tableau to learn an optimal depth- d_1 decision tree. Each node at depth d corresponds to the set of images based on the application of a specific sequence of d classifiers. The single depth-0 node $\langle \rangle$ contains all of the images considered, S . To compute the images in the $\langle (C_i, +) \rangle$ node: First let $C_1(S)$ be the subset of S that pass the C_1 classifier; assume these are sorted based on their SCO-value

³ Each c_i is a “rectangular feature”, the outcome of which is computed using “integral images”. See [12] for details.

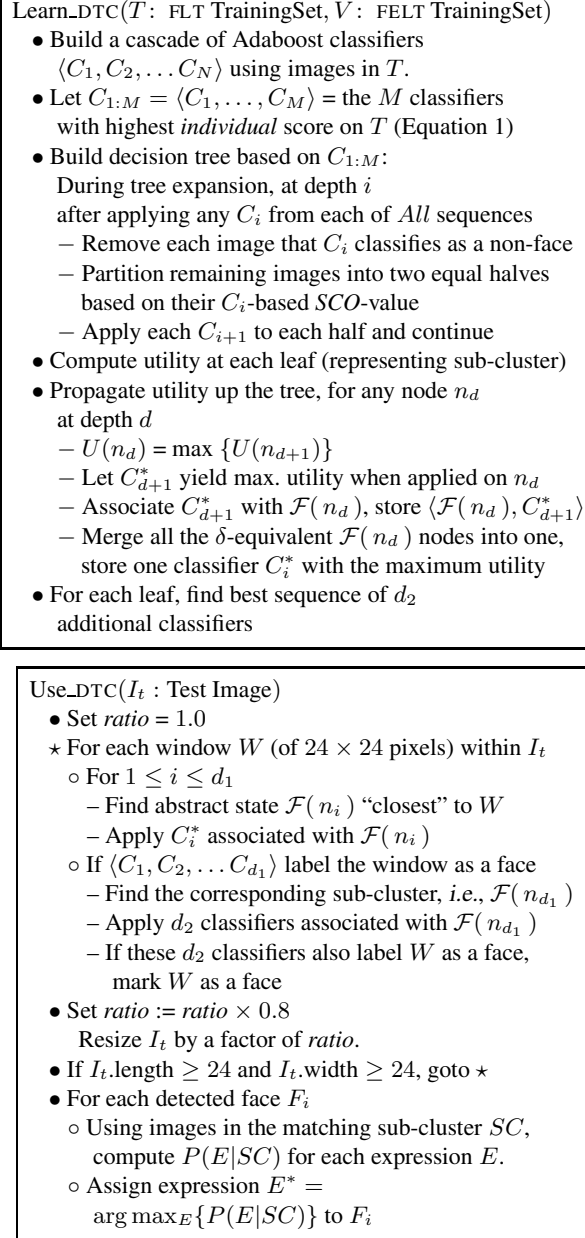


Fig. 2. (a) Learning algorithm to discover sub-clusters and find “effective” sequence of classifiers for each sub-cluster; (b) Dynamic classification algorithm

as $\langle w_1, \dots, w_{m/2}, w_{m/2+1}, \dots, w_m \rangle$, where $V_1(w_j) > V_1(w_k)$ when $j > k$. Then the $\langle (C_1, +) \rangle$ node contains $\{w_1, \dots, w_{m/2}\}$, and $\langle (C_1, -) \rangle$ contains $\{w_{m/2+1}, \dots, w_m\}$.

Similarly $\langle(C_2, +)\rangle$ contains half of the images in $C_2(S)$ — those with the highest $V_2(\cdot)$ values — and $\langle(C_2, -)\rangle$ contains the other half of the $C_2(S)$ images. And so forth for the other $\langle(C_i, \pm)\rangle$ nodes. We can then compute $\langle(C_1, +), (C_2, +)\rangle$ that contains half of the $\langle(C_1, +)\rangle$ images that pass the C_2 classifier, and $\langle(C_1, +), (C_2, -)\rangle$ that contains the other half, etc. We continue for d_1 levels, producing $\binom{M}{d_1} \times 2^{d_1}$ nodes. Each node at depth d_1 corresponds to a sub-cluster; we found that no sub-cluster contained more than 83 images.

We now want to determine the best decision tree within this tableau — the one that leads to the “purest” leaf nodes.

Computing the Utilities, First d_1 Classifiers: Each leaf of the tree represents a possible sub-cluster. We want the sub-clusters that are as “pure” as possible, which we compute based on the utility

$$U(S_{d+1}) = K_1 \times \gamma_{exp}^* \times |S_{d+1}| - K_2 \times FN(S_{d+1})$$

where $|S_{d+1}|$ denotes the number of images in the sub-cluster S_{d+1} (which might include some false positives) and $FN(S_{d+1})$ denotes total false negatives in S_{d+1} (recall this is after applying $\langle C_1, C_2 \dots C_d \rangle$) and $\gamma_{exp}^* = \max_e \{\gamma_e\}$ where γ_e is the fraction of face images in S_{d+1} with a particular expression $e \in \{\text{sad, fear, surprise, no_expression, happy}\}$. (We used the constants $K_1 = 100$ and $K_2 = 10$.) The idea is to assign high utility value to sub-clusters that group face images of the same expression and a low utility value to sub-clusters that have high false negatives. For any internal node S_i we define $U(S_i) = \max_j \{U(S_{i+1}^j)\}$ as the maximum utility of its children, $\{S_{i+1}^j\}_j$.

Using these, we propagate the utility up the tree. We collect the $\langle \mathcal{F}(S_i), C_i^* \rangle$ tuples and also the corresponding utilities, for all depths i , $1 \leq i \leq d_1$. $\mathcal{F}(S_i)$ represents the “abstract state” representation of S_i , (see below for the notion of abstract states) 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 children of S_i . For every two states S_i and S_j ($i \neq j$) that are δ -equivalent (see below for the definition of δ -equivalent states) we retain only one state that has a higher utility and the corresponding classifier.

State Abstraction: At any stage during detection (testing phase) our algorithm first tries to determine the closest matching sub-cluster for each face detected in a test image. To be more precise, after applying classifiers $\langle C_1, C_2, \dots, C_{i-1} \rangle$ to a window in the image, DTC applies the most appropriate classifier C_i . To choose C_i , we compare the performance of C_i in a “similar situation” on the training data. For that we need to have some kind of a “state representation” that allows us to effectively perform such comparisons *quickly*.

In general, with each node s in the decision tree, we can identify both the sequence of classifiers $\langle C_1, \dots, C_k \rangle$ on the path from root to s , and also a set of training images S_s that will reach s . We can also identify each s with an abstract state $\mathcal{F}(s) = \langle [V_{min,1}, V_{max,1}], [V_{min,2}, V_{max,2}], \dots [V_{min,k}, V_{max,k}] \rangle$, for each i , where $[V_{min,i}, V_{max,i}]$ is the range of SCO-values of S_s associated with classifier C_i .

Further, we say two abstractions are “ δ -equivalent”, written $\mathcal{F}(s_1) \approx_\delta \mathcal{F}(s_2)$, 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 $\mathcal{F}(s_1)$ and $\mathcal{F}(s_2)$, $|V_{min,i}^{(1)} - V_{min,i}^{(2)}| \leq \delta$ and $|V_{max,i}^{(1)} - V_{max,i}^{(2)}| \leq \delta$, where δ is a pre-defined constant. We set $\delta = 70$ throughout this work.

The result of abstraction is that a large number of complex states can be described by a small number of compact state descriptions. Of course, the same abstract state can represent multiple states — it is possible that $\mathcal{F}(s_1) = \mathcal{F}(s_2)$ even if $s_1 \neq s_2$.

2.4 Further Pruning

The d_1 classifiers leading to a leaf node both identify the sub-cluster, and also filter away many, but not all, of the non-faces. We therefore use another set of d_2 classifiers, specific to each leaf node, to help filter out the other non-faces. We noticed that each sub-cluster has just a fraction of the total number of the training data (none had more than 83, from the FELT data of 2672 images). Further, we noticed the number of false positives was not very high (less than 27) for each sub-cluster (see Section 3). So, we select a classifier $C_{d_1+1}^*$ for each state S_d ($d_1 \leq d \leq (d_1 + d_2)$) such that $C_{d_1+1}^*$ has not been used earlier and $C_{d_1+1}^* = \arg \max_{C_{d_1+1}} \{R(S_d, C_{d_1+1})\}$. We apply $C_{d_1+1}^*$, update false positives and false negatives and repeat the process until d_2 such classifiers are found. At performance time, this means our DTC algorithm will use at most a total of $d_1 + d_2$ classifiers for any image, which collectively form the most effective sequence of classifiers for that particular sub-cluster. (Of course, can be different sequences for different images.)

2.5 Detection Using the Dynamic Classifier

The DTC detection algorithm, shown in Figure 2(b), both detects faces within a given image and also identifies the expression of each such face into a sub-cluster. This face detection mechanism is very similar to the cascade classifiers method [12,13], except that it chooses the classifiers dynamically, based on the outcomes of the previous classifiers.

This process examines each 24×24 pixel window in the image; it then rescales and repeats. For each window W , DTC first applies the classifier C_1^* associated with root. This might reject the window 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 d_1 steps, until W reaches a leaf, ℓ . If so, DTC then runs the sequence of d_2 additional classifiers associated with ℓ , and declares W to be a face only if it passes all of these classifiers. (Here, it had passed all $d_1 + d_2$ classifiers.)

We also identify each detected face W with a facial expression: Recall there are many expression-labeled training images associated with this leaf node ℓ ; we assign to W the most common of these expressions.

2.6 Computational Complexity of Learning

Let N be the total number of classifiers and P be the total number of images. We sort the N classifiers based on their utility on the training data and choose M best from them. We expand the tree exhaustively up to a depth of d_1 using the M classifiers. At every depth, we apply a classifier C_d , collect the images classified as faces by C_d as positives, sort them based on the *SCO*-metric of C_d and partition the positive images into two halves⁴. Hence the total complexity of the process is $O(\binom{M}{d_1} \times P \lg(P))$. This is computationally expensive. However, since the number of training images is halved on each branch, this process terminates rapidly. In fact, starting with several thousands of images, we can obtain sub-clusters of size less than 100 by expanding the tree to a depth no greater than 3 or 4, which can be done in a few minutes.⁵

3 Empirical Studies — Face Detection

In this section we show how we apply our ideas to the challenging domain of face detection and show its performance on several images. The training set *flt* has 1600 images of faces and 2320 images of non-faces⁶, each of size 24×24 pixels. The face images include faces of many people, some having glasses, beard and many with different facial expressions, etc. Some training images of faces in FLT are shown in Figure 3(a) The FELT set has a total of 917 face images of the five basic expressions (sad, fear, surprise, no-expression and happy). It also has 2320 non-face images. Some of the face images in FELT are shown in Figure 3(b).

During the training stage, we built a cascade of 21 classifiers using Adaboost, based respectively on $\{7\ 15\ 30\ 30\ 50\ 50\ 50\ 100\ 120\ 140\ 160\ 180\ 200\ 200\ 200\ 200\ 200\ 200\ 200\}$ linear separators. We sort these classifiers based on their score $S(C_i, T)$ on the training data and select the 10 best classifiers. Using the 10 classifiers, we do a depth first search as explained in Section 2.3 up to a depth of $d_1 = 3$ and find sub-clusters. Each sub-cluster included between 26–83 images, some of which could be false positives. The number of false positives was in the range $\{0 \dots 27\}$, for any sub-cluster. Figure 4 shows face images of the same expression from some interesting sub-clusters that our algorithm learned.

For each sub-cluster we find another sequence $d_2 = 13$ classifiers⁷ that have the maximum score on the images of the sub-cluster, as explained in Section 2.3. The $(d_1 + d_2)$ classifiers learned by our algorithm form the most effective sequence for the sub-cluster they are associated with.

To detect faces in any given test image, we use the dynamic detection technique given in Figure 2(b) and explained in Section 2.5. We ran it over 150 images mostly

⁴ We compute the *SCO*-metric V_j^i of every classifier C_i on every training image I_j and also whether C_i classifies I_j as face or not, *a priori*. So, there are no extra computation during the tree expansion stage.

⁵ The training time for building the classification tree on a 1 GHz. computer with 256 Mb. RAM running Windows-2000, using MS Visual C++ was about 5 minutes.

⁶ Most of these images were downloaded from the web from popular databases like Olivetti Research & AT&T, Caltech, Yale, JAFFE, PICS, etc.

⁷ We set these values after initial experiments on many test images.



(a) Faces in the FLT set



(b) Faces in the FELT set with five different expressions — fear, happy, no expression, sad and surprised

Fig. 3. Face images in the FLT and FELT sets

from Olivetti Research database face images and could successfully identify the expression. Figure 5 shows the performance of our detection algorithm on a number of face images with various expressions. The figure also shows a graph plotting $P(E|SC)$, which indicates the probability of expression E for the detected face. We assign the expression E^* with maximum $P(E|SC)$ to the expression. Note that expressions can be mixed — like happy and surprised, sad and fear and so on. The graph below indicates the probability of each expression.

3.1 ROC Curve

To find out the effectiveness of our face detection algorithm, we ran it on 178 image face images of the MIT-CMU database with a total of over 532 faces and covering

**Fig. 4.** Various sub-clusters discovered from the FELT data — (a) Sad (b) Fear (c) Surprised (d) No Expression (e) Happy

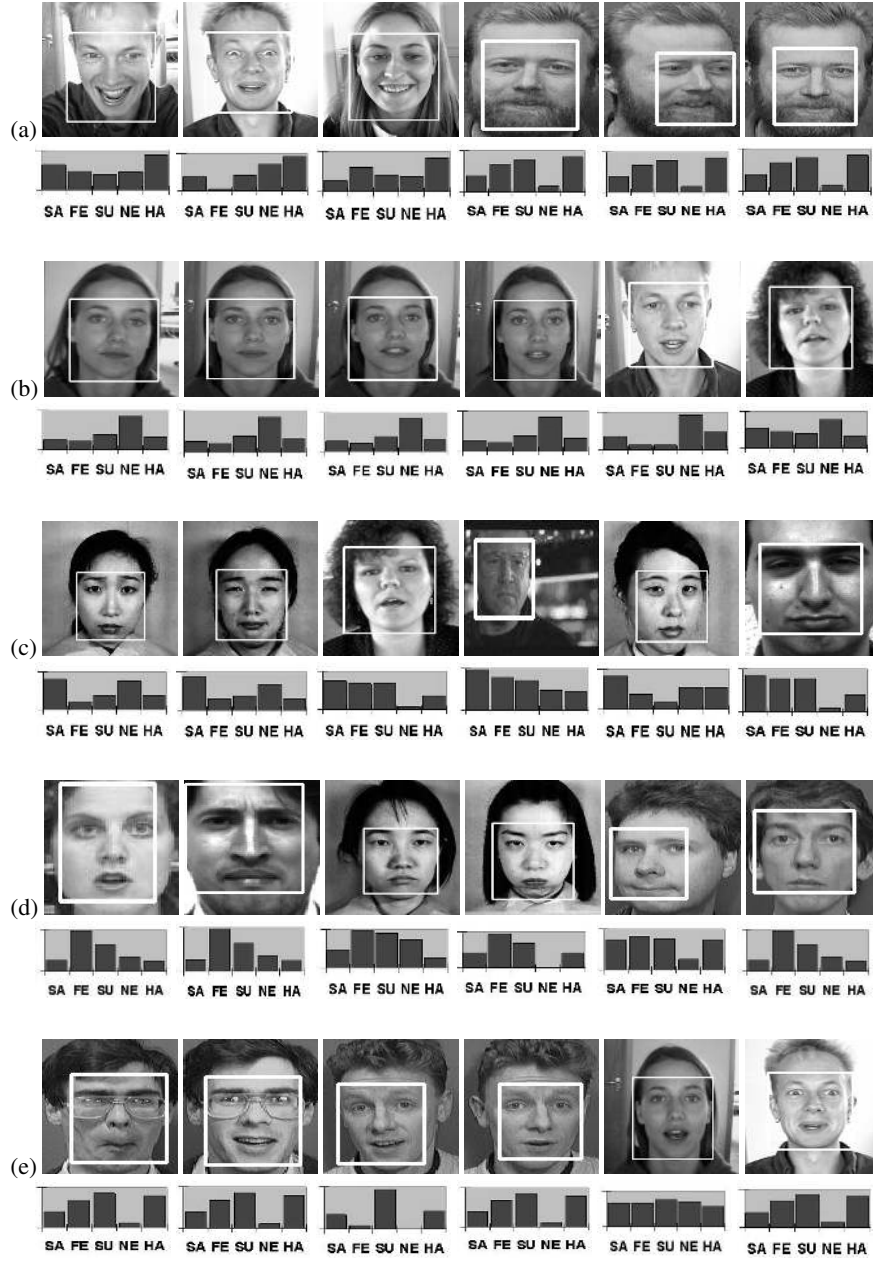


Fig. 5. Performance on several test images — (a) happy (b) no expression (c) sad (d) fear (e) surprise. For each test image, the corresponding graph below shows the probability of each of the five expressions.

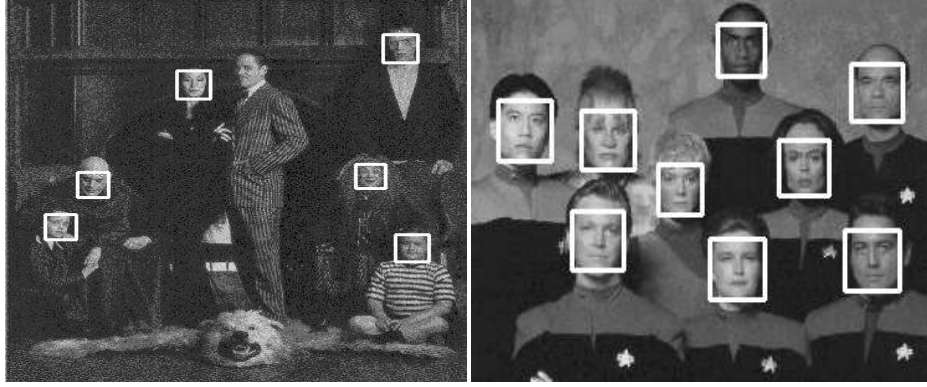


Fig. 6. Classification results on sample images from MIT-CMU database

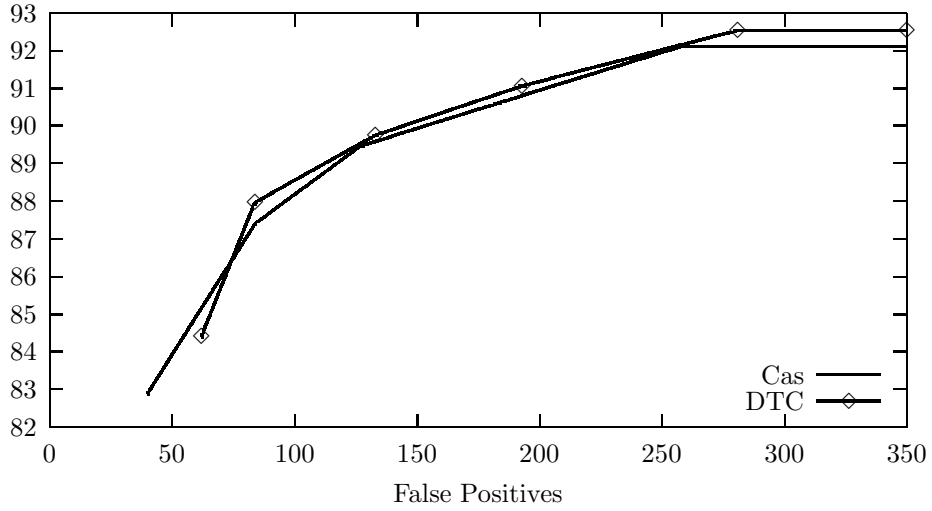


Fig. 7. ROC curve

more than 30 million windows. Figure 6 shows the performance of our algorithm on two images, each with several faces. Figure 7 compares the ROC curves for our face detection algorithm with that of Viola-Jones cascade classification algorithm.⁸ Note that Viola-Jones detection method uses 21 classifiers, while our algorithm uses only a subset of 16. Since our algorithm is built using the 21 classifiers of the Viola-Jones algorithm, all the parameters (like the number of linear separators for each classifier, the training data from which they are built) are exactly the same. An important point to note — since our algorithm uses only a subset of the classifiers as Viola-Jones method, it detects every face that Viola-Jones method detects.

⁸ This is the implementation of the Viola-Jones method [12] by Wu and Rehg.

The main difference is the way they choose the classifiers — Viola-Jones method is static, while our algorithm is dynamic. Our technique can adapt itself to choose the most effective classifier specific to the sub-cluster. Hence, it can apply complex classifiers (with a high number of linear classifiers) very early in the detection process.

The points on the ROC curve are obtained by varying the number of classifiers applied in the range [4–21] for Viola-Jones method and [4–16] for our method. As can be seen from the graph, both the methods seem to have similar performance, with a peak detection rate of 92.5%, while our algorithm does marginally better.

3.2 Efficiency

We ran our Viola Jones algorithm and our algorithm on 400 images of Olivetti Research database face images, each with a size of 92×112 pixels and each image having exactly one face. On average Viola-Jones algorithm took 0.189 seconds per image while our algorithm took 0.243 seconds. The increase in computational time can be attributed to two reasons — our algorithm not only detects faces but also assigns an expression to each detected face, that involves extra computation. Secondly, Viola-Jones algorithm is a static algorithm optimized for speed. It uses boosted classifiers with smaller number (less than 10) of weak classifiers, that are faster to execute, early in the detection process. We cannot detect the best matching sub-clusters (to find out expression) by choosing classifiers statically. Hence, our algorithm may select more complex classifiers (that take more time to execute) in the early stages of detection. However, complex classifier can also prune off several thousands of false positives initially increasing the overall detection efficiency.

4 Previous Work

There has been limited work in identifying face expressions. Liu and others [7,3,2] track facial features and analyze them for facial expressions. Others proposed several other methods [1] almost all of them are based on methods that do a local analysis of facial features, like mouth, eyes, etc. Our work is significantly different from all these — we don't use sequence of images or analyze facial features explicitly, but use training set to group face images of expression into sub-clusters. We associate each detected face with a sub-cluster to identify the expression.

Many researchers have recently proposed several methods for detecting faces in images; see [8,10,9] for a small sample. There has been a lot of interest in the cascade classification methods using classifiers, after the seminal work of Viola and Jones [12,11,13]. All of the cascade classification methods are static — the number of classifiers and the order in which they are applied is fixed; *i.e.*, the same for each instance. Although our approach also uses classifiers, it differs because the order of those sub-classifiers can change for different images. We present an algorithm to learn a dynamic classification method, that decides which classifier to apply to an image based on the outcome of the classifiers already applied. Grossman [5] presented a tree-based detection method, which selects a linear combination of weak classifiers dynamically, based on the outcome of the previous weak classifiers. Our work is also dynamic but differs

significantly as we build a classification tree using these subclassifiers in the internal nodes. Moreover, the aim of our work is not only to detect faces but to discover different sub-groups in training images of faces and to associate each face that is detected in a test image to one of these. Thus, it is an extension to simple face detection. [6] addressed related issues in a feature-based face-recognition system by posing the task a “Markov Decision Problem (MDP)”. They use dynamic programming to produce an optimal *policy*, π^* , that maps “states” to “actions” (feature detector) for that MDP, then used that optimal policy to recognize faces *efficiently*. We use similar techniques here in this work — we aim to find the best sequence of classifiers here, to detect faces and also each detected face to a sub-cluster.

5 Conclusions

We give future directions to extending this work and present the contributions of our work.

5.1 Directions for Future Work

In this work we use classifiers to achieve two objectives — to detect faces and to group each detected face into one sub-group. There are several interesting extensions to this work; it can be extended to partition training images into sub-clusters of faces with and without (external) features like glasses, moustache, etc. The detection system can be used to differentiate people with these features from those without these features.

It will be interesting to try our method on gender classification also. This work can be directly extended to do multiple object detection. Our concept of sub-clusters of faces can be directly correlated to different objects. We can still use a binary (positive and negative) classifier where the training images of all the objects to be detected are positive examples and the rest are negative examples. The training phase of computing the rewards and building a classifier will be quite similar, where the most effective sequence of classifiers will be those that can separate images of different objects into separate clusters. The detection phase will be guided by the dynamic classifier where a classifier with the maximum utility will be selected at each stage.

Finally, we believe that with suitable modifications our approach can be used for gesture recognition, where each sub-cluster corresponds to face images that belong to people with the same gesture.

5.2 Contributions

The main contribution of our work is that it assigns expression to faces during detection. Our training algorithm partitions training images into sub-clusters of similar expressions. During detection, every detected face is matched to one of these sub-clusters to identify the expression. Another novel aspect of our work is that our face detection method is dynamic. We present an algorithm to learn a dynamic classification method that applies the most effective classifier based on the outcome of the previously applied classifiers.

References

1. B. Abboud and F. Davoine Facial expression recognition and synthesis based on appearance model, *Signal Processing and Image Communication*, Elsevier, Vol. 19, No. 8, pp. 723-740, Sep. 2004
2. J.F. Cohn, T. Kanade, T.K. Wu, Y.T. Lien and A.Zlochower, Facial Analysis: Preliminary analysis of a new image processing based method International Society for Research in Motion, Toronto, 1996
3. J.F. Cohn, A. Zlochower, J. Lien, Y.T. Wu and T.Kanade Automated face coding: A computer vision based method of facial expression analysis Seventh European Conference on Facial Expression, Measurement and Meaning, Salzburg, Austria, 1997.
4. Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Computational Learning Theory: Eurocolt*, 1995.
5. E. Grossmann. Adatree: boosting a weak classifier into a decision tree. In *IEEE Workshop on Learning in Computer Vision and Pattern Recognition*, 2004.
6. R. Isukapalli, and R. Greiner Use of Off-line Dynamic Programming for Efficient Image Interpretation *IJCAI*, Acapulco, Mexico, Aug 2003
7. Y. Liu, K. Schmidt, J.F. Cohn and S. Mitra Facial asymmetry quantification for expression invariant human identification *Computer Vision and Image Understanding*, pp. 138–151, vol 91, 2003
8. H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 1998.
9. D. Roth, M. Yang, and N. Ahuja. A snowbased face detector. In *Neural Information Processing Systems (NIPS)*, 2000.
10. H. Schneiderman and T. Kanade. A statistical method for 3d object detection applied to faces and cars. In *International Conference on Computer Vision (ICCV)*, 2000.
11. P. Viola and M. Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.
12. P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.
13. J. Wu, J.M. Rehg, and M.D. Mullin. Learning a rare event detection cascade by direct feature selection. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2003.