Robust Object Detection Via Soft Cascade

Lubomir Bourdev
lbourdev@adobe.com

Jonathan Brandt
jbrandt@adobe.com

Office of Technology, Adobe Systems Inc., 345 Park Avenue, San Jose, CA 95110

Abstract

We describe a method for training object detectors using a generalization of the cascade architecture, which results in a detection rate and speed comparable to that of the best published detectors while allowing for easier training and a detector with fewer features. In addition, the method allows for quickly calibrating the detector for a target detection rate, false positive rate or speed. One important advantage of our method is that it enables systematic exploration of the ROC Surface*, which characterizes the trade-off between accuracy and speed for a given classifier.*

1. Introduction

The trade-off between accuracy and speed is a central aspect of the problem of object detection in images. Since the object can appear potentially anywhere in the image, the classification function must be applied at a comprehensive set of positions and scales. Furthermore, accurate classification is complex and slow due to the vast variation of appearances of the object, and even greater variation of the non-object class.

A common way to reduce the computational burden of evaluating a complex classifier over an entire image is to decompose the classifier into a linear sequence, or *cascade*, of sub-classifiers. For the problem of face detection, each sub-classifier, or stage, is a binary classification function that is trained to reject a significant fraction of the non-faces, while allowing almost all the faces to pass to the next stage. Each successive stage is trained based on the non-faces that pass all prior stages. The computational speed-up is achieved by weeding out the vast majority of non-faces in the early stages which are relatively simple to evaluate.

The cascade is not a new idea. It is in essence a triage strategy, and has appeared in various forms dating back to the 1970s, as was recently pointed out by Schneiderman [6]. It is often used implicitly by prefiltering obvious non-faces based on various heuristic criteria, such as the presence of skin tone [7] or intensity variance [10]. Féraud et al [1] propose a four stage cascade that includes a motion filter, color filter, a neural network and a PCA-based classifier. Heisele et al [2] propose a cascade of coarse-to-fine SVM-based classifiers. Li et al [3] propose a three level cascade with Haar-like features for a multi-view face detection system. Viola and Jones [10] propose a cascade-based face detector with stages consisting of fast Haar-like block filters. Each of the stages of the Viola-Jones cascade is trained using a variation of AdaBoost [4]. Subsequent work has refined and extended the Viola-Jones detector [3, 6, 11, 12, 8], while retaining the fundamental structure of the cascade.

Figure 1: *Examples of difficult face examples detected by a Soft Cascade trained for frontal upright faces.*

While the cascade structure has been shown to be effective at speeding up the detection process, it has several disadvantages that we address in this paper. Fundamentally, the cascade has the flaw that most of the information obtained from evaluating a given stage is discarded as it passes to the next stage. Consequently, the decision to accept or reject an instance at a given stage does not take into account how well the instance performed in the prior stages. This can result in a brittle classifier since a face can be misclassified just because it barely fails to satisfy the criteria of a single stage, and conversely for a non-face.

A second disadvantage of the cascade structure is the severe requirement it imposes on training each of the individual stages. In particular, a positive classification must pass every stage, which implies that the final detection rate is the product of the detection rates of all stages. For a

stage cascade to achieve a detection rate of 90% at false positive rate of 10^{-6}, each of its stages must be able to preserve close to 99% of the faces, while eliminating on average more than 25% of the non-faces. This may be trivial for the initial stages, but for later stages it becomes a very difficult task, which results in large stages that are difficult to train and time-consuming to evaluate.

A third disadvantage of the cascade structure is that the training parameters, namely the target detection and false positive rates at each given stage, provide at best only indirect control over the execution speed of the resulting classifier. Therefore, in order to explore the trade-off between speed and accuracy for a given detection problem, the cascade must be repeatedly retrained with varying training parameters. This is a serious problem since each training cycle can take weeks of computation to complete.

A fourth disadvantage of the traditional cascade is that there is no obvious way to pick optimal values for its free parameters. The number of stages, the ordering, the target detection rate and false positive rate at each stage all affect the execution speed for a given target accuracy. Sun et al [8] propose a method for picking values for some of these parameters that optimize the detection accuracy. However, it is not clear how to set the parameters to optimize speed at a given target accuracy.

Xiao et al [12] propose the Boosting Chain which addresses the first disadvantage of the cascade, that is, that subsequent stages are unable to make use of information obtained in evaluating prior stages. Specifically, one of the improvements they propose is propagating the cumulative sum from prior stages as input to the next stage and as a result are able to show an accuracy improvement over [10].

The Boosting Chain, as well as the traditional cascade, suffer from a fifth disadvantage — to reduce the false positive rate one must train new stages, which irreversibly reduces the detection rate. Lastly, traditional cascades are not suitable when the variability of positive examples is too high. Multi-view detection systems, for example, typically require training a distinct cascade for each viewpoint and using a decision tree to direct the search to the appropriate one.

In this paper we propose a new structure, the Soft Cascade, which retains the desirable execution efficiency of the cascade while addressing each of the problems described above. Our experimental results indicate that the Soft Cascade structure produces classifiers with speed comparable to that of the best published detectors while significantly improving the detector accuracy. The basic approach consists of two main ideas: (1) to generalize each stage to be a scalar-valued, rather than binary-valued, decision function proportional to how well the given instance passes the stage and to the relative importance of the stage; and (2) generalize the decision function that determines whether or not a given instance passes a given stage to depend on the values of each of the prior stages rather than just the value of the stage under consideration.

The Soft Cascade is similar to the Boosting Chain [12] in that it allows for monotonic accumulation of information as the classifier is evaluated. However, the Soft Cascade allows us to quickly explore the speed/accuracy trade-off for a given classifier without having to retrain it. This is in contrast with the conventional cascade and with the Boosting Chain, wherein training parameters that must be set prior to training dictate the ultimate speed of the classifier.

By exploiting the capability of the Soft Cascade to continuously trade-off speed versus accuracy we are able to generate a Receiver Operating Characteristic Surface (ROC Surface) which represents the interdependency of detection rate, false positive rate, and execution speed as a 3D surface. We believe that the ROC Surface provides greater insight into classifier performance than a simple ROC curve or family of curves can provide.

2. Formulation of the Soft Cascade

In order to motivate the Soft Cascade, consider training a face detector using AdaBoost and a set of Haar-like features as weak learners as in Viola-Jones [10]. However, rather than training a sequence of consecutive stages, consider training a single, potentially very long stage consisting of T features. The resulting classifier is of the form:

$$H(x) = \sum_{t=1,\ldots,T} c_t(x)$$

where $c_t(x) = \alpha_t h_t(x)$ are the set of thresholded Haar-based classifiers selected during AdaBoost training scaled by the associated weights.

Let $H_t(x) = \sum_{i=1,\ldots,t} c_i(x)$ be the partial sum up to and including the t-th feature, or the response of sample x at step t. The values of $H_t(x)$ as a function of t for a fixed sample x constitute a *sample trace*. Figure 2 depicts the traces for a collection of face and non-face samples on a classifier consisting of approximately 2500 features. Clearly, the traces corresponding to the faces steadily separate from the non-faces as the evaluation progresses, which demonstrates the effectiveness of the learning procedure. Also, it is evident from the traces that there is significant statistical dependency among the features. In particular, the sample traces progress in a fairly orderly manner, thereby enabling an early decision as to whether the sample is a face or not.

Within this context, the conventional cascade structure requires finding the smallest t and threshold r_t such that the fraction of faces with trace values greater than r_t is greater than the target detection rate and the fraction of non-faces with trace values greater than r_t is less than the target false

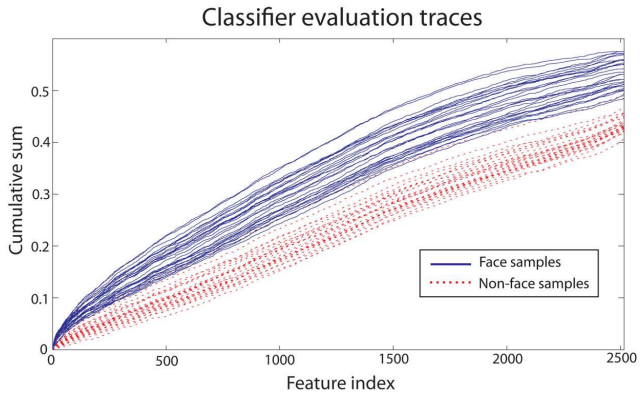

Figure 2: *Sample traces*

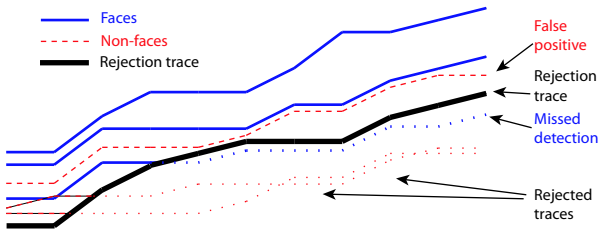

Figure 3: *Sample evaluation using Soft Cascade*

positive rate. New weak learners are trained until such conditions are met. The cumulative sum for a given instance is only computed to determine if it exceeds r_t and is thereafter discarded.

Alternatively, consider setting a threshold r_t for each of the T features. After each feature evaluation, the cumulative sum $H_t(x)$ is compared with the *rejection threshold* r_t. If the cumulative sum is less than the rejection threshold, then the sample is rejected as a non-face; otherwise evaluation continues. All samples which are fully evaluated and exceed the final rejection threshold r_T are accepted as faces. The rejection thresholds define the *rejection trace*, as shown of Figure 3. Figure 4 shows a pseudo-code specification of this procedure.

In general, the Soft Cascade framework is defined by selecting a stage output function $c_t(x)$, a scalar-valued function proportional to how well the given instance x passed stage t and to the relative importance of that stage and a rejection function $R_t(c_1(x) \dots c_t(x))$ after each stage t. The rejection function determines whether to terminate evaluation or to let it continue to stage $t+1$.

The first advantage of the proposed method is that it does not discard information prematurely, thus making better rejection decisions. The second advantage is that it does not impose a burden on each stage to preserve nearly all faces — a face may fail one or more stages and still be correctly classified. As a consequence, stages can be small, easy to train, and fast to evaluate.

```
bool sampleIsFace(x)
    d ← 0
    for t = 1...T
        d ← d + c_t(x)
        if d < r_t return false
    return true
```

Figure 4: *The Soft Cascade algorithm with thresholded sum as the rejection function*

We demonstrate the method by training a face detector via a variation of AdaBoost using thresholded Haar-like block filters as in [10]. In terms of the Soft Cascade, the stage output functions are the weighted features selected by AdaBoost and the rejection function is the thresholded cumulative sum of the stage output functions. The justification for the above choices is as follows: AdaBoost has been shown as a very effective learning method with good generalization performance [5]. The integral filters are among the fastest possible effective weak learners. The weights that AdaBoost assigns to each weak learner correspond to its importance and our choice of sum of weights as the rejection function is also based on AdaBoost's sum.

The next section presents a training procedure for determining the stage output functions. The subsequent section presents a method to reorder the stages and determine the rejection functions. These latter two steps comprise Soft Cascade *calibration*.

3. Soft Cascade Stage Training

The goal of this section is to determine a set of weak classifiers (our stage output functions c_t) that, when added together, result in a strong classifier. The basic AdaBoost algorithm would be perfect for this goal, except that we do not have a good way of obtaining representative samples of the space of non-faces. Accurate representation of the space of non-faces would require impractically many negative samples. The common strategy to address this problem is to use bootstrapping — to start training with a random set of negative samples and keep adding new ones during training that the classifier misclassifies. Table 1 presents a modified AdaBoost algorithm that incorporates bootstrapping.

In Step 1, for performance reasons, we do not search exhaustively for all possible features to find an error minimizing one, but sample only a random subset of them and do a spatially localized search at each random sample to find the corresponding local error minimum. We found this strategy to be very effective in finding a feature very close to the optimal while sampling only a few percent of the features. Step 2 follows the AdaBoost algorithm as described in [10]. It finds a weak classifier that minimizes the weighted error over the training set. We introduce bootstrapped non-

Input:

- a,b is the number of negative and positive samples, respectively.
- Training samples $(x_1, y_1), \ldots, (x_{a+b}, y_{a+b})$ where $y_i = 0, 1$ for negative and positive samples, respectively.
- T is the target number of weak classifiers.

Initialize:

- $w_{0,i} \leftarrow \frac{1}{2a}, \frac{1}{2b}$ for $y_i = 0, 1$ respectively.

For $t = 1, \ldots T$**:**

1. For each feature j train a classifier h_j restricted to using the feature. The error of the classifier is defined as $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$
2. Choose the classifier h_t with the minimum error ϵ_t. Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$, $\alpha_t = -\log \beta_t$, and $c_t = \alpha_t h_t$.
3. Add K bootstrapped negative samples and scale back the weights of the existing negative samples to keep their sum constant:
   3.1. $\forall i, y_i = 0 : w_{t-1,i} \leftarrow w_{t-1,i} \frac{a}{a+K}$
   3.2. For $k = 1 \ldots K$, add samples $(x_{N+k}, 0)$ with $w_{t,N+k} = \frac{1}{2a}$ such that
   $$\sum_{j=1}^{t} c_t(x_{N+k}) \geq \frac{1}{2} \sum_{j=1}^{t} \alpha_j.$$
   3.3. $a \leftarrow a + K$.
4. Optionally remove weak features.
5. Decrease the weights of all samples correctly classified by h_j as in [10]:
   $$\forall i, h_j(x_i) = y_i : w_{t-1,i} \leftarrow \beta_t w_{t-1,i}$$
6. Normalize the weights: $w_{t,i} = \frac{w_{t-1,i}}{\sum_j w_{t-1,j}}$

Output:

- The stage output functions c_t.

Table 1: *The algorithm to determine stage output functions*

faces in Step 3 we add K non-face samples that the classifier misclassifies. Although AdaBoost tends to correct itself over time, we found that training is improved if we keep the relative weight distribution between positive and negative samples as we introduce new samples (Step 3.1). The number K of new non-faces to add at each training cycle affects the focus of AdaBoost on improving the detection rate vs. the false positive rate. If K is too large, AdaBoost will not be able to catch up and the error rate over the negative training set will be high. Alternatively, if K is too small, training a classifier with good false positive rate would require too many weak classifiers. We believe a good strategy is to adjust K at each step to keep the error rate over the bootstrapped non-faces constant.

Introducing bootstrapped negative samples during training changes the terms of the problem as we go along. Therefore features that AdaBoost deemed appropriate earlier may prove to be suboptimal after introducing new negative samples. Such features are removed in Step 4. This gives further justification to not spending too much effort trying to find the optimal feature in Step 2 but just a very good one; the optimal feature at the current step may be suboptimal once new bootstrapped non-faces are introduced.

Li et al [3] propose a backtrack-based strategy for weeding out weak features called FloatBoost, which could be utilized in Step 4. We do not use it because of its performance when thousands of features are involved and also because, as discussed above, working hard to remove optimally weak features is not necessary, since the problem changes as we go along. We chose a simpler strategy: remove features whose removal results in a decrease in the error rate.

4. Soft Cascade Calibration

In this section we present an algorithm that, given a target detection rate D and target execution time S determines nearly optimal values to the parameters of the Soft Cascade (Figure 4) that minimize the false positive rate. We call the process of determining these parameters *classifier calibration*.

4.1. ROC surface

Each calibrated classifier is associated with a point in the space obtained by extending the plane of the ROC curve along an axis corresponding to expected evaluation cost. The set of points in this space that are spanned by all possible calibrations of a given classifier constitute its operating domain. For any fixed detection rate and false positive rate in the operating domain, there is a unique point in the operating domain of minimum evaluation cost. The set of all such points constitutes the ROC surface for the classifier. (Note that the familiar ROC curve forms a portion of the boundary of the ROC surface.) Any calibration corresponding to a point not on the ROC surface is suboptimal since in this case there exists another equally accurate, yet faster, calibration that is on the ROC surface.

The calibration algorithm determines a suitable ordering of the stages and the rejection thresholds r_t after each stage. The ordering of the stages defines the sample traces, whereas our choice for rejection thresholds defines the rejection trace (Figure 3). Intuitively, we want to pick more discriminating stages and stages that complement each other closer to the beginning so as to maximize the separation between the positive and negative samples. (While the first few stages picked by AdaBoost are very discriminating, the order of the remaining is typically suboptimal.)

As for the rejection trace, if we use larger rejection thresholds and have a higher rejection trace, more of the

positive samples will cross it and thus we will get lower detection rate but also lower false positive rate. Note also that the "area" of traces above the rejection threshold defines the speed of the detector. If the rejection trace increases steeply initially, it will cross and terminate most of the traces early on and produce a fast detector, which will also not be very accurate because the positive and negative samples are not well separated in the initial stages. Alternatively, if our rejection trace is initially conservative and rises only in the later stages, we will obtain a slower but more accurate detector. The most conservative rejection trace that is zero everywhere except at the last stage corresponds to the uncalibrated detector described in the previous section.

Let $v_t \geq 0$ be the minimum fraction of faces that we can miss at the t-th stage. We call the vector $v = (v_1, \ldots, v_T)$ the *rejection distribution vector*. The vector v fully defines the position of the detector on the ROC surface: the sum of its elements is $1 - D$ where D is the target detection rate, while its distribution controls the trade-off between expected execution time and false positive rate (as outlined above).

In Section 4.2 we describe an algorithm that projects a given rejection distribution vector onto a point on the ROC surface (i.e. determines its expected false positive rate and execution time). It also determines the Soft Cascade parameters associated with that point. In Section 4.3 we describe a method that uses that ROC surface projection algorithm to search for the rejection distribution vector corresponding to the minimum false positive rate for a given detection rate and execution time and outputs its corresponding Soft Cascade parameters.

4.2. ROC Surface projection algorithm

In the following discussion, for a given predicate x, let $pred(x)$ be 1 if x is true and 0 otherwise.

Given a particular v, the calibration algorithm described in Table 2 reorders the stages and sets the rejection threshold for each stage. The calibration algorithm requires a set of positive and negative training samples which are kept distinct from the training set used to train the stages.

The algorithm consists of T cycles. In each cycle t we select the stage output function that will occupy the t-th place in the final ordering and we also select its corresponding rejection threshold r_t. Throughout the algorithm we keep track of the fraction of faces p we can reject at the current step. At each cycle we add to it the allowance we have for the current cycle v_t (Step 1) and we subtract the fraction of faces actually removed in the current cycle (Step 5). Sometimes we cannot consume our entire allowance in the current cycle, in which case the remainder is carried over to the next cycle. In Step 2 we select as the next stage output function the one that if applied to the calibration samples would result in the largest separation between the average

Input:

- Calibration samples $X = \{(x_1, y_1) \ldots (x_N, y_N)\}$ where $y_i = 0, 1$ for negative and positive samples, respectively. Let $a = \sum(1 - y_i)$, $b = \sum y_i$ be the number of negative and positive calibration samples, respectively.
- $v_1 \ldots v_T$ is the rejection distribution vector.
- $\{C\}$ is the set of T stage output functions c_t determined by the training algorithm (Table 1).

Initialize:

- The sample responses $d_{0,i} \leftarrow 0$.
- The face rejection fraction $p \leftarrow 0$.
- The expected execution time $m \leftarrow 0$.
- The number of negative samples used so far $A \leftarrow a$

For $t = 1 \ldots T$**:**

1. $p \leftarrow p + v_t$. $a_t = \sum_{(x_i,y_i)\in X}(1 - y_i)$. $b_t = \sum_{(x_i,y_i)\in X} y_i$.
2. From the stages in $\{C\}$ select the index of the stage that maximizes the separation between the positive and negative samples:
$$q(t) = \arg\max_j \quad \sum_i f_{t,i,j} y_i / b_t - \sum_i f_{t,i,j}(1-y_i)/a_t \quad ,$$
where $f_{t,i,j} = d_{t,i-1} + c_j$.
3. Update the sample traces: $d_{t,i} \leftarrow d_{t-1,i} + c_{q(t)}(x_i)$.
4. Select the rejection threshold as the maximum one that removes no more than p fraction of the faces. That is, let r_t be the largest value r for which $\sum_i pred(d_{t,i} \leq r) y_i \leq pb$.
5. Update:
$$p \leftarrow p - \sum_i pred(d_{t,i} \leq r_t) y_i / b.$$
$$X \leftarrow X - \{(x_i, y_i) : d_{t,i} < r_t\}.$$
$$C \leftarrow C - \{c_{q(t)}\}.$$
$$m \leftarrow m + cost(c_{q(t)}) a_t.$$
6. Search A_t number of randomly drawn negative samples until finding K bootstrapped ones:
   - 6.1. For $k = 1 \ldots K$ add samples $(x_{N+k}, 0)$ with response $d_{t,N+k} = \sum_{j=1}^{t} c_{q(j)}(x_{N+k})$ chosen such that $\forall j \in \{1 \ldots t\}, \sum_{m=1}^{j} c_{q(m)}(x_{N+k}) \geq r_j$.
   - 6.2. $N \leftarrow N + K$.
   - 6.3. $A \leftarrow A + A_t$.

Output:

- The stage output functions $c_{q(t)}$ and r_t.
- The expected false positive rate $F = a_T / A$.
- The expected execution time $M = m/A$.

Table 2: *The algorithm to compute the rejection thresholds and the ordering of stages*

positive and the average negative sample response. We can improve this rule by weighting the samples, that is, giving higher weights to the harder positive and negative samples. That improvement gives marginally better results but complicates the formulas and slows down calibration. In Step 3 we update the traces of our calibration samples. It Step 4 we choose the maximum threshold that removes no more than the fraction p of the total number of faces we are allowed to remove. By setting the threshold as high as possible, we allow for removing the maximum possible number of negative samples. In Step 5 we update the fraction of faces we are allowed to reject, remove from consideration the samples whose traces fall below the rejection threshold and remove the just chosen stage output function from the set to choose from. In Step 6 we replenish the negative samples by bootstrapping, choosing only ones whose traces stay above the rejection trace. The larger the number K of new samples we add, the more representative our data is, but also the more time consuming the calibration becomes. In our implementation we keep a target number of negative samples at each cycle that starts high and gradually decreases. This approach gives us a reasonable trade-off between calibration speed and generalization.

We discovered a reasonable alternative to the bootstrapping Step 6 which results in a significantly faster calibration with only a small degradation to the result. Once we train a classifier (Table 1), we collect a representative set of non-faces specific to it and we use that set on each calibration run for that classifier instead of step 6. We collect representative non-faces by doing a one-time sampling of a large number (in the order of 100 million) of random non-face windows and ordering them in bins based on the classifier response. The response distribution approaches a Gaussian with a peak of 0.44. We randomly discard the vast majority of samples around the peak of the Gaussian but preserve more of the ones in the upper tail, which correspond to the harder non-faces. We also assign weights of the samples in each bin based on the fraction of the number of samples falling into that bin that we end up keeping. Using this method the algorithm on Table 2 takes less than five minutes per run.

The calibration algorithm also outputs an estimated false positive rate F and execution time M. To compute the false positive rate we simply divide the number of negative samples remaining at the end by the total number A of negative samples ever considered. We obtain the expected execution time M by accumulating the number of negative samples that pass each stage multiplied by the cost of the stage output function (Step 5) and dividing the total by the number of negative samples considered. In our implementation all stages have relatively equal cost, so our cost function is 1. Thus in our implementation the expected execution time is represented as the average number of features evaluated per sample. We use only the negative samples when estimating execution time because the fraction of positive samples in real data is negligibly small.

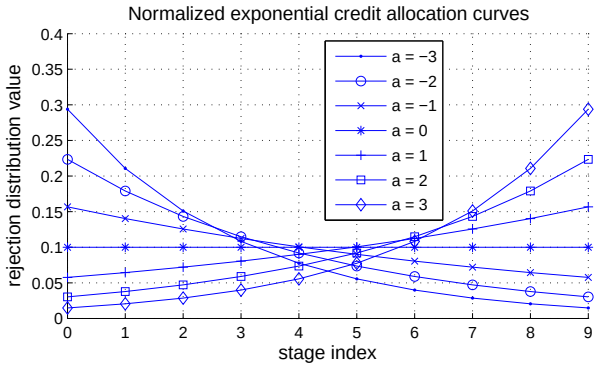

Figure 5: *Exponential family of rejection distribution vectors for a 10 stage classifier*

4.3. Finding point close to the ideal ROC surface

The algorithm in the previous section returns the execution time and false positive rate for a particular rejection distribution vector v. In this section we use it to find the vector v that minimizes the false positive rate given a target detection rate D and execution time S. The elements of vector v must be non-negative and their sum must equal $1 - D$ to satisfy the target detection rate requirement.

We have found empirically that the specific shape of the vector is not as important as the distribution of its elements towards the beginning (corresponding to faster classifiers) or towards the end (corresponding to classifiers with better false positive rate).

We use the exponential function family of the form:

$$v_t = \begin{cases} ke^{-\alpha(1-\tau)} & \text{when } \alpha < 0 \\ ke^{\alpha\tau} & \text{when } \alpha \geq 0 \end{cases}$$

where $\tau = t/T$, k normalizes the vector sum to satisfy the target detection rate and α is the free parameter for the function family, as illustrated on Figure 5. This particular function family tends to favor speed over accuracy when $\alpha < 0$ since most of its mass is towards the beginning stages, while the opposite is true for $\alpha > 0$.

For a given detection rate D and a value of α, we construct a vector v and use it in the algorithm on Table 2 to find the expected execution time of the corresponding classifier. To find a suitable calibration for a given detection rate and speed, we do a one-dimensional search for the value of α that gives us execution time equal to the target execution time S. The result of calibration is the stage ordering $q(t)$ and the rejection thresholds r_t (provided by the algorithm on Table 2) corresponding to that point on the ROC Surface.

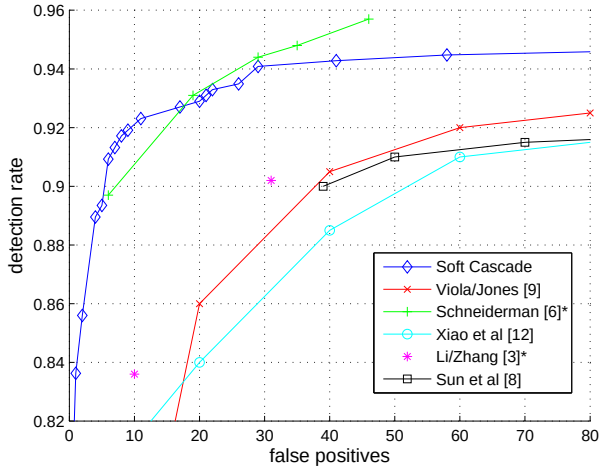

Figure 6: *Comparative detector performance on CMU+MIT face set. (The (*) denotes data with 5 hand-drawn images removed.)*

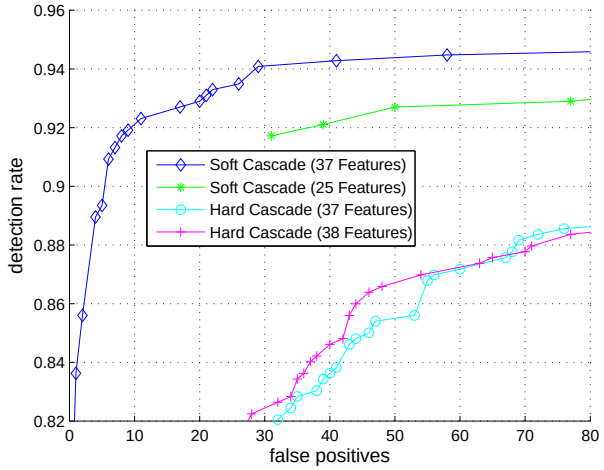

Figure 7: *Comparing ROC curves of the Soft Cascade for different execution speeds on the CMU+MIT face set*

5. Results

A Soft Cascade face detector was trained as described in Section 3. The training set consisted of approximately 17000 frontal upright face images resampled to a canonical 22×22 window size, as well as an initial set of 32000 non-face images. New bootstrapped non-face images were continually added during training. The final set of non-faces included approximately 220000. The result of training was a detector consisting of 4943 thresholded Haar features of the same types as described in [10].

5.1. CMU+MIT Face Set Comparative Test

To get a sense of the accuracy of the classifier in the limit, the uncalibrated version was tested against the CMU+MIT data that consists of 130 images containing 507 faces. Figure 6 depicts the ROC curve for the uncalibrated Soft Cascade classifier when applied to the CMU+MIT face test set. The curve was obtained by fully evaluating the classifier and continuously varying the overall threshold. Comparable curves are depicted for several other recently reported cascade-based classifiers. Of particular interest is the improvement over [9] even though we employ fewer features overall. (The detector in [9] is reported to employ 6061 features.) This improvement can be attributed directly to the advantages of the Soft Cascade, since the two detectors are otherwise very comparable. In addition, the Soft Cascade shows an improvement over [12] despite the fact that [12] utilizes several optimizations to eliminate redundancy in the selected features. The Soft Cascade ROC performs better than [3], although [3] also eliminates feature redundancy through FloatBoost backtracking and employs a much richer set of primitive features. The results reported in [6] are comparable to our system. Nevertheless, a conventional cascade is used in [6] and our conjecture is that using the Soft Cascade could further improve the results. (Note that both [3] and [6] report results based on a reduced set with 5 images containing hand-drawn faces removed.)

Figure 7 explores ROC curves at constant-evaluation times. By setting the rejection thresholds conservatively we were able to obtain an average evaluation time of 37.1 features with virtually no loss of accuracy over the full evaluation. That plot demonstrates the effectiveness of the Soft Cascade and further illustrates the steepness of the execution time dimension near the full evaluation limit. Figure 7 also depicts the ROC for a hard cascade trained in the manner described in [9] using the identical training set, feature set and based on the same boosting algorithm as that of the Soft Cascade. The performance disparity highlights the advantages of the Soft Cascade. Due to the high variation of our training set, we were unable to obtain a hard cascade that executes faster than about 37 features per window on average.

To explore the learning power of our algorithm on a harder classification problem, we successfully trained a 24x32 soft cascade using about 40000 frontal faces with much wider variation of in-plane and out-of-plane rotations. Figure 1 shows examples of faces this classifier can find.

5.2. Exploring the ROC Surface

In this section we use the calibration algorithm to sample the ROC surface, as shown on Figure 8. The surface has been approximated based on discrete data points which are shown as dots in the figure. The vertical axis is the expected evaluation time in terms of number of features and is presented in a logarithmic scale because of the range of times is so great. The top edge of the surface approaches the full

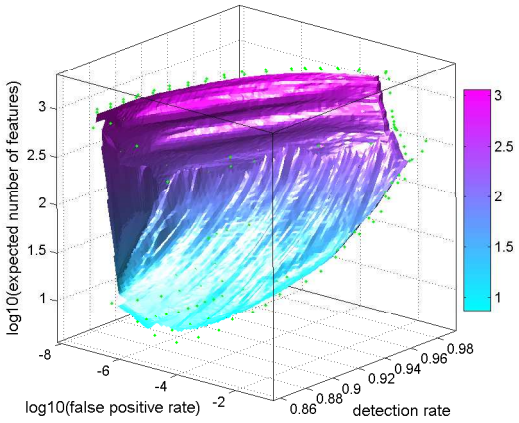

Figure 8: *The ROC surface for the Soft Cascade face detector*

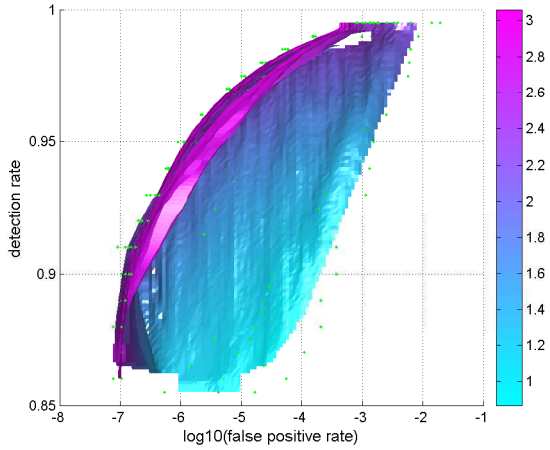

Figure 9: *Overhead view of the ROC surface*

evaluation ROC curve as can be seen more clearly in the overhead view depicted in Figure 9. Note that the maximally conservative calibrated classifier is still considerably faster than the full classifier because it is able to reject all negative sample traces that fall below the minimum positive sample trace without affecting the detection rate.

6. Conclusion

We described a method for generalizing the cascade that addresses problems with the traditional cascade structure, namely, the unnecessary discarding of information at each stage, as well as the burden imposed at each stage to retain nearly all positive samples.

The proposed architecture consists of a single monolithic classifier that is augmented with a rejection threshold function that is tested at each step of the classifier evaluation. We introduced a continuous bootstrapping method that allows for training the classifier against a representative sampling of the non-faces. We introduced a method to calibrate the classifier for a specific detection rate and execution time.

We have demonstrated that our system allows for creating faster and more accurate detectors that are also more compact and therefore easier to train. In addition, the new architecture effectively decouples the speed/accuracy tradeoff from training. Once a classifier is trained, we are able to quickly calibrate it for a given point in the ROC surface.

Acknowledgements

We would like to thank Hailin Jin for his assistance in the preparation of this manuscript, and Gregg Wilensky for his helpful review.

References

[1] R. Féraud, O. Bernier, J. Viallet, and M. Collobert. A fast and accurate face detector based on neural networks. *IEEE Trans. PAMI*, 23(1), January 2001.

[2] B. Heisele, T. Serre, S. Prentice, and T. Poggio. Hierarchical classification and feature reduction for fast face detection with support vector machines. *Pattern Recog.*, 36, 2003.

[3] S. Li and Z. Zhang. Floatboost learning and statistical face detection. *IEEE Trans. PAMI*, 26(9), September 2004.

[4] R. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA*, 2001.

[5] R. Schapire, Y. Freund, P. Bartlett, and W.-S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proc. 14th Int. Conf. Machine Learning*, pages 322–330, 1997.

[6] H. Schneiderman. Feature-centric evaluation for efficient cascaded object detection. In *IEEE Conf. Computer Vision and Pattern Recognition*, 2004.

[7] H. Schneiderman and T. Kanade. Object detection using the statistics of parts. *Int. J. Computer Vision*, 2002.

[8] J. Sun, J. Rehg, and A. Bobick. Automatic cascade training with perturbation bias. In *IEEE Conf. Computer Vision and Pattern Recognition*, 2004.

[9] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, 2001.

[10] P. Viola and M. Jones. Robust real-time object detection. In *IEEE ICCV Workshop on Statistical and Computational Theories of Vision*, 2001.

[11] J. Wu, J. Rehg, and M. Mullin. Learning a rare event detection cascade by direct feature selection. In *NIPS*, 2003.

[12] R. Xiao, L. Zhu, and H.-J. Zhang. Boosting chain learning for object detection. In *ICCV*, pages 709–715, 2003.