On Training Cascade Face Detectors

Brendan McCane
Department of Computer Science
University of Otago, New Zealand
email: mccane@cs.otago.ac.nz

Kevin Novins
Department of Computer Science
University of Auckland, New Zealand
email: novins@cs.auckland.ac.nz

Abstract

In this paper we present two improvements over Viola and Jones [1] training scheme for face detection. The first is 300-fold speed improvement over the training method presented by Viola and Jones [1] with a modest increase in execution time. The second is a principled method for determining a cascade classifier of optimal speed. We present some preliminary results of our methods.

Keywords: face detection, cascade classifiers

1 Introduction

In their excellent paper, Viola and Jones [1] describe an extremely fast method for finding faces in images or video input. The method is very fast (achieving approximately 15 frames/sec on a 700MHz Pentium III), and quite accurate (according to Figure 7 in the paper, achieving greater than 90% correct detection rate for only 0.000053% false positives). These are impressive results. There are three main contributions of their paper: use of an integral image to rapidly evaluate simple features; using the AdaBoost [2] boosting algorithm to improve the effectiveness of simple classifiers on the simple features; using a cascade of AdaBoost classifiers to quickly eliminate most non-faces from consideration.

There are two main limitations to Viola and Jones [1] work. Firstly, the training algorithm they use is extremely slow. To find the best feature at any stage in the cascade, they perform an exhaustive search over every possible feature. In fact, their final classifier makes use of approximately 6000 features, and was trained using approximately $20,000$ training images. Since at each stage they need to search through $180,000$ features for each training image, the training procedure requires $180000 \times 6000 \times 20000 = 2.16 \times 10^{13}$ feature evaluations. Although Viola and Jones [1] do not report training time in their papers, in our implementation, using $126,000$ features and approximately 20000 training images, the method would take on the order of one year on a 2GHz Pentium IV. Such a long training time provides a large impediment to the general adoption of such techniques - as it is extremely difficult to test new theories or algorithms, or even verify Viola and Jones [1] results.

Secondly, their method for determining the target false positive rate at each stage in the cascade is ad hoc. Viola and Jones [1] specify a priori the required false positive rate at each stage of the cascade, alternatively, one could set the maximum number of simple classifiers within each stage. In both cases, it appears the required number is set based on heuristics or experience with the problem at hand. However, there is a clear function to minimise in this case and that is the final speed of the classifier. If we can model the behaviour of the classifier, then we can find an optimal solution, in terms of classifier speed. It is these two aspects of training a cascade classifier which we address in this paper.

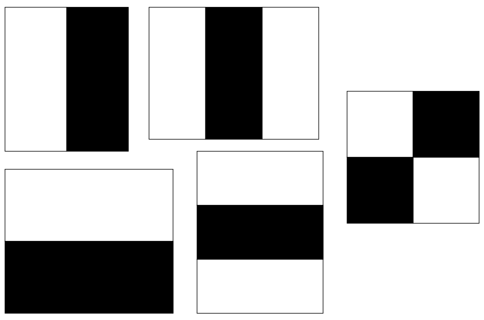
2 Fast(er) Feature Selection

In Viola and Jones [1] there are only 5 basic types of features which are displayed in Figure 1(a). The complete set of features (180,000 according to Viola and Jones [1] , 126,000 according to us), is generated by varying the width, height and starting position of each of these features with respect to a 24x24 window within an image. The feature set can be parameterised by four parameters: width, height, and the feature's offset within the window. These parameters are displayed in Figure 1(b). The feature selection method used by Viola and Jones [1] and adopted here is simply to choose as the next feature to use that which maximally reduces the error rate within our cascade. That is, at stage i in the cascade, set:

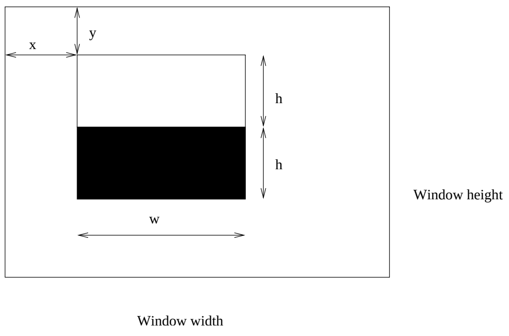
$$\mathfrak{f}_i = \arg\min_{\mathfrak{f}_j \in \mathfrak{F}} E(\mathfrak{f}_j) \qquad (1)$$

where $\mathfrak{F}$ is the set of all possible features, $E(\mathfrak{f}_j) = (n_p + n_n)/N$, and n_p is the number of false positives, n_n is the number of false negatives, and N is the total number of training examples.

In other words, at each stage we are performing an exhaustive search over all possible features - a type of optimisation problem. Since our feature set is parameterisable, and our feature selection method is a form of numerical optimisation - we have a constrained non-linear optimisation function with integer parameters. Such a problem is called a constrained non-linear integer programming problem, for which several solutions

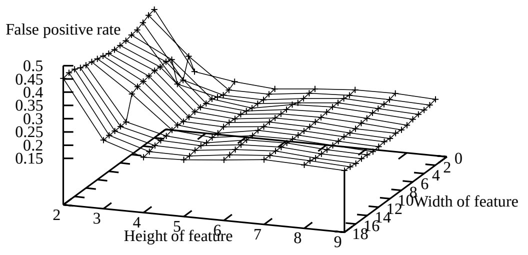


(a) 5 basic features

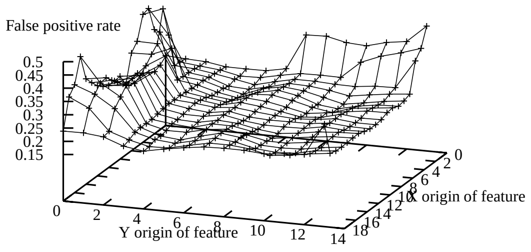


(b) Feature parameters

Figure 1: Simple features



(a) Fixed Origin Slice



(b) Fixed Width and Height Slice

Figure 2: Two-dimensional slices of the error function to be optimised

have been proposed - most notably outer approximation methods [3] or branch and bound methods [4]. However, such methods typically require at least an ability to evaluate the objective function at non-integer points and often assume that the function is of some analytic form such as being convex. It is not clear how to apply these methods to our current problem. Instead, we have developed a simple and fast heuristic for finding a sub-optimal feature.

The error function, $E(\mathfrak{f}_j)$ appears to be well behaved. We have evaluated the function at all integer grid points for each of the five features. This is a four-dimensional function as shown in Figure 1(b). Figure 2 displays two two-dimensional slices of the evaluation function for a two rectangle feature. One in which the origin of the feature is kept fixed, and the other in which the width and height of the feature is kept fixed. In both cases, we see that the function is relatively well behaved and therefore we expect that our heuristic optimisation procedure will find good features in most cases.

Our algorithm is deterministic and very simple and would be amenable to conversion to a non-deterministic algorithm such as simulated annealing which we intend to investigate in the future. The algorithm does a kind of discrete downhill search by evaluating a points' immediate neighbours and then moving to the smallest error point. The scale of the neighbourhood is adjusted to avoid local minima. The algorithm is as follows:

Algorithm 1 (Heuristic Search)

```
while $\neg done$ do
    $done = true$
    $f = \arg\min_{f_j \in \mathfrak{N}_s(f)} E(f_j)$
    if $E(f) < E(f_{best})$
      then $f_{best} = f$
          $s = 1$
          $done = false$
    fi
    if $done$
      then $s = s * 2$
    fi
end
```

Where $\mathfrak{N}_s(f)$ is a neighbourhood based on the size of the parameter s:

$$\mathfrak{N}_s(f) = \{f \pm I_s^x : x \in \mathfrak{D}(f), I_s^x = [0...s...0]^T\} \quad (2)$$

where $\mathfrak{D}(f)$ is the dimensionality of the parameter vector f. In a two-dimensional search space, $s = 1$ would

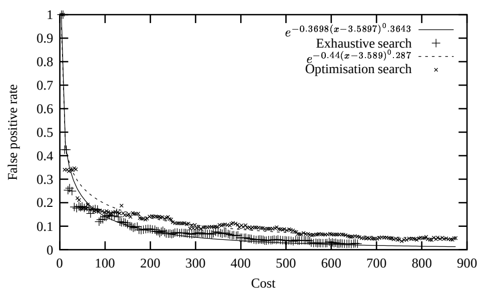


Figure 3: Some results for classifier cost

produce the normal 4-neighbours for example. Regardless of the value of s the neighbourhood function only returns 4 sites - that is, we do not search all possibilities within $\pm s$ from our current position, only those points at the extremities of the bounding box in the axis directions. Our search space is 4-dimensional, so there are 16 neighbours in a given neighbourhood.

Figure 3 shows a comparison of the false positive rate of two AdaBoost classifiers as we add simple classifiers to the ensemble. In one of the classifiers we use the exhaustive search technique of Viola and Jones [1] and in the other we use our optimisation search procedure. The detection rate is maintained at > 0.99, and an independent data set is used to generate the false positive rates. In these examples, the optimisation search strategy offered a 300-400 fold performance increase. Finding the approximate best feature using the optimisation search requires approximately 500 function evaluations, versus more than 100,000 with Viola and Jones [1] method. Our implementation of Viola and Jones [1] method would literally take one year or more of training time. Our optimisation method reduces this time to something more feasible. On the negative side, the exhaustive search strategy produces superior classifiers due to finding better features as shown in Figure 3.

Aside from efficiency reasons, another advantage to parameterising the features is that we can easily define more complex features with only a modest increase in training times. For example, we may want to decouple the feature rectangles so they are not necessarily joined. We can do this by adding an extra two parameters for each rectangle in our feature. Training times with Viola and Jones [1] exhaustive search is exponential in the number of parameters of the features, so adding more parameters is computationally impractical. We have not fully investigated the effects of using more complex features at this stage, but rather note it as a possible advantage of the new method.

3 Optimal Cascade Speeds

In our scheme a 320×240 pixel image has approximately 36,000 sub-windows to be classified, of which only a few contain faces. The central idea of a classifier cascade is to have relatively simple classifiers that reject a large proportion of negative examples early in the cascade. Windows that are classified as not a face need not be considered further, and hence can be rejected very quickly. Windows that pass the first cascade may or may not be a face. The second classifier is then used to reject more of the non-faces, and so on all through the cascade. Only actual faces, need to pass through all layers in the cascade. Most windows can be rejected very early in the cascade. This makes for an efficient classifier on average. Figure 4(a) shows a cascade classifier pictorially.

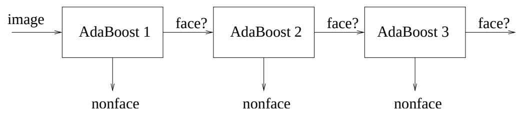
It should be clear that the goal of using a cascade of classifiers is to improve the average rejection speed of the classifier. That is the average speed to classify a given window as a non-face. Since we assume that there are very few face regions in an image compared to non-faces, we can ignore the average speed of classifying faces. Recall that each stage of the cascade rejects a certain proportion of image regions that are passed to it. Let's say that cascade stage i rejects the proportion of $1 - p_i$ of the images passed to it. That is, p_i is the false positive rate of cascade stage i. Associated with each stage is a certain cost of execution, C_i. Stage 1 is always executed. Stage 2 is executed only for those regions that successfully passed through stage 1. That is, the false positive rate of stage 1 (ignoring actual faces which also pass through stage 1). Similarly, stage 3 is only executed for those regions which successfully passed through stages 1 and 2, and so on. This then gives us a clear expression for the average cost of execution of the cascade:

$$\begin{aligned} C_a &= C_1 + C_2 p_1 + C_3 p_1 p_2 + \ldots && (3) \\ &= C_1 + \sum_{i=2}^{N} \left[C_i \prod_{j=1}^{i-1} p_j \right] && (4) \end{aligned}$$

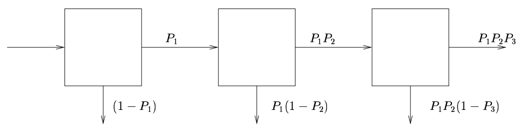
See Figure 4(b) for a pictorial view of how this works. We need only minimise equation 4, subject to relevant constraints, to find the optimal speed classifier.

3.1 Modeling the Component Cost Function

It should be obvious that the cost of executing a cascade stage and the false positive rate of that stage are correlated. We would expect from a good classifier, that as the cost goes up, the false positive rate comes down (ignoring over-training effects due to insufficient test data). In fact, this is approximately the behaviour that we find as shown in Figure 3. It is not exact, because in the scheme of Viola and Jones [1] , AdaBoost is performed and then in a post-processing step the AdaBoost threshold is adjusted to achieve the desired detection



(a) A classifier cascade



(b) Cost of executing the classifier

Figure 4: Classifier cascades

rate on a set of training data. AdaBoost actually optimises for the overall error rate which is not exactly the optimisation we want. What we want is to optimise for the false positive rate subject to the constraint that the required detection rate is achieved. As such, sometimes adding a level to the AdaBoost classifier actually increases the false positive rate. Further, adding new stages to an AdaBoost classifier will eventually have no effect when the classifier improves to its limit based on the training data.

Figure 3 shows a plot of how the cost of execution of a classifier varies with respect to the false positive rate. The cost of the classifier is estimated as the sum of the cost of executing each of the features in the classifier. In our implementation, for integer features the cost is 3.8×10^{-7}, 4.9×10^{-7} and 5.2×10^{-7} seconds for each of 2, 3 or 4 rectangle features respectively on a Pentium III 650MHz machine. Also shown in figure 3 is how closely we can model the cost with an exponential decay function. The exact form of the function we use is:

$$K(p) = (-ln(p)/a)^{1.0/c} + b \qquad (5)$$

where $K(p)$ is the cost required to achieve a false positive rate of p, and the units of the cost function is micro-seconds (10^{-6}s). The constants a, b and c were determined using a non-linear least squares fit. For the exhaustive search, we have $a = 0.3698$, $b = 3.5897$ and $c = 0.3643$. And for the optimisation search, $a = 0.45$ $b = 3.5897$ $c = 0.2869$. The constant b is effectively a minimum penalty term. This reflects the fact that an AdaBoost classifier typically does not reduce the false positive rate until at least two classifiers are boosted. This is due to the post-processing threshold adjustment scheme. As can be seen from the figure, the data is quite noisy, again, mostly due to the post-processing threshold adjustment. However, the trend of the graph is quite clear.

Of course, the cost of each stage of the cascade is equal to the cost of getting from the current false positive rate to the new false positive rate. This cost can be estimated as follows:

$$C(p_i) = K(\prod_{j=1}^{i} p_j) - K(\prod_{j=1}^{i-1} p_j) \qquad (6)$$

where p_i are the component false positive rates of each stage in the cascade.

3.2 Optimising the Cost Function

We are now able to optimise equation 4. Rewriting equation 4, to illustrate the final form of the function gives:

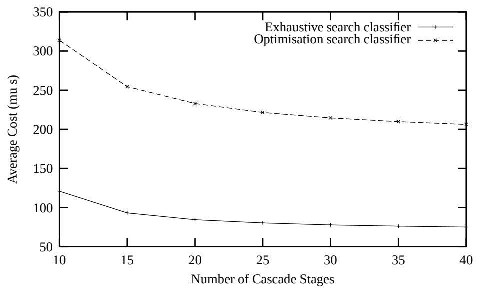
$$C_a(p_1, p_2, ..., p_N) = C(p_1) + \sum_{i=2}^{N} \left[C(p_i) \prod_{j=1}^{i-1} p_j \right] \qquad (7)$$

where $C(p_j)$ is defined in equation 6. Theoretically, we could optimise this function analytically, and this is possible if we wish to limit the size of the cascade. However, in practice, large cascades, which are generally more efficient, contain too many variables in the cost function to be amenable to analytic optimisation. This then leaves us with a constrained numerical non-linear optimisation problem. There are two types of constraints on the problem. The first requires that $0 \leq p_i \leq 1$ since each p_i is a relative frequency measure. In practice, it is impossible to achieve a false positive rate of 0 so it is useful to increase the value of the lower bound to some suitably small number greater than 0. That is, the first constraint becomes: $l \leq p_i \leq 1$, where we have used $l = 0.05$ as an achievable false positive rate. There is one further complication with these bounds. The exponential decay cost function (or its logarithmic inverse) is quite inaccurate at modeling the cost when there are very few features in an AdaBoost classifier. In fact, for the training data we have used, it requires at least 3 features to reduce the false positive rate at all, at which point it typically reduces the false positive rate by between $40 - 50\%$. Further, it can require up to 6 features to reduce the false positive rate significantly more than 50%. For this reason we have set the false positive rate for the first cascade stage to $0.5 \leq p_1 \leq 0.6$.

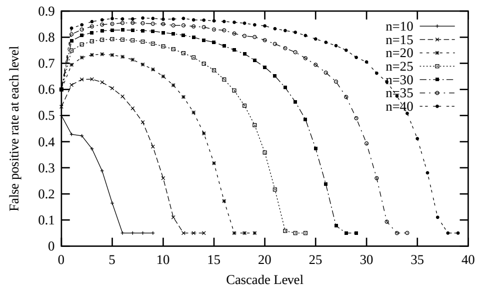
The second constraint is nonlinear, $\prod_{i=1}^{N} p_i \leq k$, where k is the target false positive rate of our whole classifier. To find the optimal speed classifier, we need to optimise over each of the p_i and over N, the number of stages in the cascade. Note we could also fix the required average speed of the classifier and optimise for the false positive rate.

Since the minimisation in this case can be performed offline, we have used an off the shelf minimisation routine. More specifically, we use the `fmincon` function from the Matlab Optimization toolbox. Even so, the optimisation is not straightforward since `fmincon` is a gradient descent based optimisation routine and is not guaranteed to produce an optimal solution. Therefore we perform several restarts to increase the chances of finding the optimal solution.

We have found the optimum parameters with cascades from 10 to 40 stages using both the exhaustive search strategy and the optimisation search strategy and the results are shown in Figure 5(a). The first thing to note is that both curves appear to be asymptotically approaching some minimum average cost, and we would expect that adding more cascade layers would improve the cost a little, but probably not significantly. Secondly, the exhaustive search strategy is approximately three times faster on average than the optimisation search. A reasonable trade-off in our opinion considering training times. Further we can still maintain much of the learning speed advantages of the optimisation search scheme by using the exhaustive search for a few of the earlier cascade stages, and then switching to the optimisation search for the less important later stages - a kind of hybrid learning scheme. We have not yet fully investigated this idea however. Figure 5(b) shows the false positive rate required at each stage of a cascade to achieve an optimal cascade of the given size. The trend here is very clear - most of the hard work should be left for the stages very late in the cascade.



(a) Average cost of optimal cascades of different sizes



(b) Required false positive reduction rates for the optimal cascades (exhaustive search). The false positive rates are achieved independently at each stage of the cascade. That is, the overall false positive rate is the product of the data points shown in the figure.

Figure 5: Some results for different sized cascades.

4 Preliminary Results and Conclusion

Figure 6 shows the results of one of our classifiers on two of the test images. Our current classifier has 23 cascade stages and achieves a recognition rate of 0.71 with a false positive rate of 2.9×10^{-4} on the MIT/CMU test set [5]. Our model predicts a recognition rate of 0.79 with a false positive rate of 1.6×10^{-4} which is in reasonable agreement with the empirical results. Our scanner using our optimisation search runs at approximately 4 frames (320×240) per second (fps) at 10 distinct scales on a 650MHz Pentium III laptop. This is consistent with Viola and Jones [1] who report 15 fps on an 700 MHz Pentium III - approximately 3 times slower as predicted by our theoretical analysis.

In conclusion, we have presented two improvements over Viola and Jones [1] original face detection scheme. We have improved the training time of the face detector by 300-fold, and have developed a principled method for determining the optimal speed cascade.

References

[1] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of somple features. In *Computer Vision and Pattern Recognition*, volume I, pages 511–518. IEEE Computer Society, 2001.

[2] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[3] M.A. Duran and I.E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

[4] Omprakash K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546, 1985.

[5] H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:22–38, 1998.

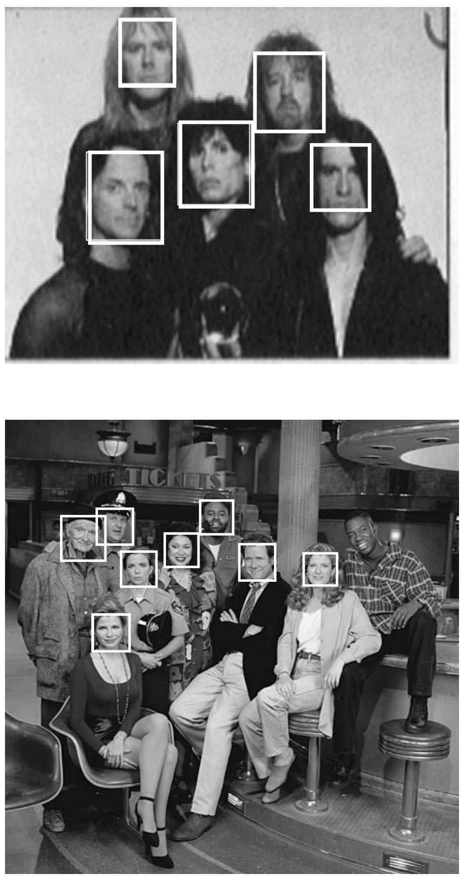


Figure 6: Some example detections