Pose Estimation using Cascade Trees

by

Patrik Sundberg

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Feb 2004

Author ....
Department of [illegible] nputer Science
Jan 30, 2004

Certified by ....
Trevor Darrell
Associate Professor
Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Pose Estimation using Cascade Trees

by

Patrik Sundberg

Submitted to the Department of Electrical Engineering and Computer Science
on Jan 30, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

In this thesis, I implemented and extended a face detector, based on cascades of boosted features, for use in real time systems. The extensions are twofold. First, I designed a way of combining several cascades into a *cascade tree*, and showed how such a tree provides a powerful mechanism for combining detector efficiency and accuracy. When the training data has large variations, the cascade tree yields a faster detector, and when the data has only small variations, there is a distinct detection rate improvement. As a second extension, I designed a system for pose estimation based on an array of cascades. I performed an evaluation of this system and compared to normalized cross-correlation.

Thesis Supervisor: Trevor Darrell
Title: Associate Professor

Acknowledgments

For showing me what really matters, Jo Ann. For advice, Trevor Darrell. For discussions, Louis-Philippe Morency. For random stuff around lab, Mario, Ali, Mike, Naveen, Ellie, Teresa and David. For data, Mike Jones. For guidance, Bob Jaffe and Deb Roy. For unconditional support and trust, my family back home. Thanks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past decade, the digital revolution reached the domain of cameras and pictures. A large fraction of new cameras sold today are digital, and soon the traditional camera will only be used by photo enthusiasts. Many new cell phones come equipped with cameras, and sometimes motion pictures are shot using digital video cameras.

The change in availability of digital imaging devices allows digital systems to come much closer to the individual, and to interact with people in a more direct fashion. This forces us to address the problem of teaching these systems how to recognize when humans are interacting with them, and how to interpret their actions.

One task that needs to be solved is finding the human faces in an image. To a human, this is a very simple task. We do it all the time, with none or little effort. But how do we do it? Consider a sheet of paper with little numbers ranging from 0 to 255, representing pixel intensity values, placed in a grid pattern. How do you decide whether or not you are looking at an image of a face? This is the task the computer must handle. It has no innate knowledge of humans or human faces, and a much less flexible brain than we do. Unless we can tell it exactly how to do the task, it will not be able to do it.

Face detection is a subproblem of the more general task of object detection. However, it is not one of the hardest ones, mostly because faces are rigid objects, with a number of clear features. The eyes, the nose, the outline of the cheeks and chin, and the hairline are all very helpful in detecting faces. In fact, some approaches to face detection [4] focus on finding the parts, and then combining the results to find whole faces. These approaches, although accurate, are usually very slow. Other approaches have used neural networks [7] or wavelet decompositions [8].

Slow methods can be very accurate, and many use techniques that appear very reasonable. However, to be used in real time systems, a detector has to not only be accurate, but also fast. The amount of computation per pixel that can be done while keeping up with video input is very small. On a 2 GHz machine, 30 frames per second of 640 by 480 video leaves time for a few hundred machine instructions per pixel. Translated to more direct terms, that means in the average case, we have a few hundred additions and subtractions to determine whether or not a window of the image, say of size 24 by 24, does or does not contain a face.

In [10], Viola and Jones introduced a method to accomplish this. They use a fast way to evaluate simple rectangular features, and a cascaded detector to quickly reject obvious non-faces, while spending more time on more complicated regions. In Chapter 2, we discuss the ideas behind their detector, and some properties of it. This sets the stage for the rest of the thesis, which will use the fundamental ideas of this approach to do both face detection and pose estimation.

When training several cascades for object classes that are very similar, such as human faces in different poses, there will be redundant information in the cascades. On the other hand, if the appearances of two poses are too different, a single cascade will not be an efficient detector. For example, the rejection rate per node may be too low, because the statistical variation in the training set is too high. This would result in a detector that returns far too many false positives to be useful. In Chapter 3, we introduce an extension to the detector cascade, *cascade trees*, which describes what to do in this situation. Cascade trees offer a flexible way to combine detector efficiency and accuracy.

We then in Chapter 4 consider the task of estimating pose using cascades. We describe how to collect the data needed for the pose estimator, and give the algorithm to actually estimate pose. The evaluation of the algorithm, along with the evaluation of cascade trees, follows in Chapter 5. We describe three different experiments, designed to highlight different features of the cascade trees and evaluate the pose estimator. We finally conclude with a discussion of the approach and results in Chapter 6.

Chapter 2

A Boosted Framework for Detection

In a standard face detection task, the input image to scan may be 640 by 480 pixels, and contain zero, one, or perhaps four human faces. The face detector must find all the faces in the image, independent of scale, orientation, lighting and facial expression. As posed, the search problem is very complex, and it is customary to reduce the task to a pure detection problem.

The reduced task solves a simpler, binary decision problem. Given an image of a predetermined size, say 24 by 24 pixels, is it a face or not? If an algorithm that solves the binary decision problem exists, we can solve the face detection problem by independently considering each possible subwindow of the full input image. To find faces of different sizes, we also need to scan across different scales. In practice, the search across scales could be performed by resizing the image, running the detector again, and repeat until the resized image is too small for the binary decision algorithm.

Let's assume that our search algorithm rescales the images by a factor of 0.8 at each step. Some simple arithmetic shows that for the image dimensions given above, the face detection algorithm must be run on almost 750,000 windows. If we want the total processing time to be a fraction of a second per frame, the binary detector needs to be extremely fast in the average case.

In [10], the authors proposed a detector built to have exactly this property. Their detector is able to reject most non-face subwindows after only a few very simple computations. This chapter explains how that detector works. In a later chapters we will extend the ideas behind it to construct a detector that simultaneously detects faces in multiple poses, and to build a system that can estimate face pose.

2.1 Simple Classifiers

Imagine a face detector that has a detection rate of 95%, and a false positive rate of 80%. On its own, such a classifier is useless. But imagine a voting scheme, where hundreds or thousands of these classifiers are all allowed to consider an input. Figure 2-1 shows how the separability of faces from non-faces increases as the number of simple classifiers increase, assuming all classifiers are independent.

In reality, we cannot find thousands of independent classifiers. In addition, we're not guaranteed to be able to find sufficiently many filters that have a 95% detection rate and not more than 80% false positive rate. However, in [10] the authors showed that we can find classifiers that are approximately independent, with good detection and false positive rates. They also showed that this is enough for our purposes. Following their work, we base our face detector on simple classifiers of the form

$$c(I) = \begin{cases} 1 & \text{if } p_i f_i(I) > t_i; \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

The functions f_i are chosen from a predetermined set of filters F. Each $f \in F$ takes an image subwindow I as a parameter, and returns a scalar.

The filters in F, some of which are shown in Figure 2-2, are specified by two, three or four rectangular regions. At least one region will have positive parity, and at least one will have negative parity. The pixel values in each rectangular region are summed together, and then the resulting sums are combined using their corresponding parities to compute the final filter output. In theory, we could use rectangles of any size, as long as they fit inside the subwindow. In practice, to limit the total number of filters in F, we disallow filters composed of differently sized rectangles. In addition, we

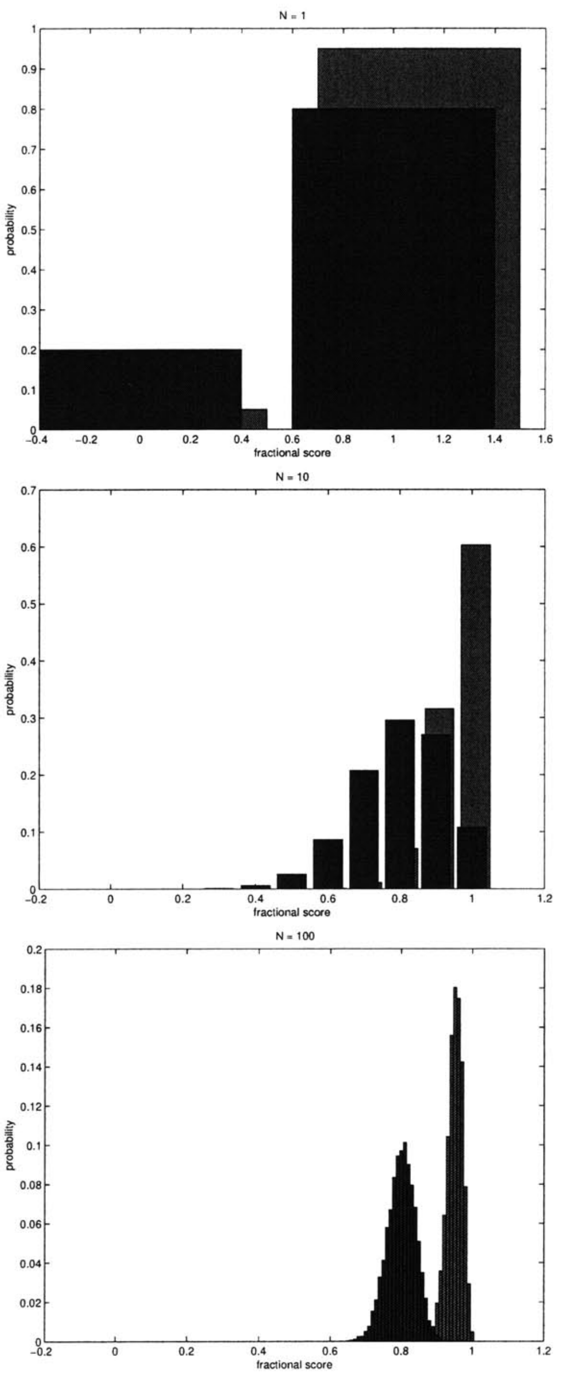

Figure 2-1: Separation of face- and non-face as a function of the number of simple classifiers. Consider an ensemble of independent classifiers that return 1 with 95% probability given that the sample is a face, and 1 with 80% probability given that it is not. This figure shows how the difficulty of separating faces from non-faces decreases dramatically with N, the number of classifiers used.

require the rectangular regions to be connected.

This still leaves a very large number of possible filters to choose from. Using only the types of filters shown in Figure 2-2, we have a pool of more than 138000 unique filters to consider. These filters may look simple, but with the threshold properly set in a frontal face detection task, the best classifier alone can reject more than half of the non-face subwindows, with virtually no missed detections.

2.2 Integral Image

One of the benefits of using rectangular features as basis functions for the simple classifiers is that due to the sheer abundance of available filters, some classifiers are bound to be very accurate. Another key benefit is that they can be evaluated in constant time using *integral images*. An integral image is uniquely specified by its source image, and is generated by replacing each pixel in the normal image by the sum of all the pixels above and to the left of it.

To compute the sum of pixels in any rectangular region, only four references to the integral image is required, as shown in Figure 2-3. Similarly, the difference of the sums of two adjacent rectangles can be computed using only six array references. This allows for very fast evaluation of the simple classifiers, and is key to the success of the Viola-Jones approach.

Creating the Integral Image can be done in time linear in the number of pixels in the image. Consider

$$I_{ab} = \Sigma_{x=0}^{a}\Sigma_{y=0}^{b}\, i_{xy} = \Sigma_{x=0}^{a}\Sigma_{y=0}^{b-1}\, i_{xy} + \Sigma_{x=0}^{a-1}\, i_{xb} + i_{ab} = I_{a,b-1} + I_{a-1,b} - I_{a-1,b-1} + i_ab. \tag{2.2}$$

Eq. (2.2) shows one way of structuring the computation so that it is linear time. Computing the integral image takes only a few machine instructions per pixel, and will not be the most time-consuming step of the face detector, independent of the size of the input image.

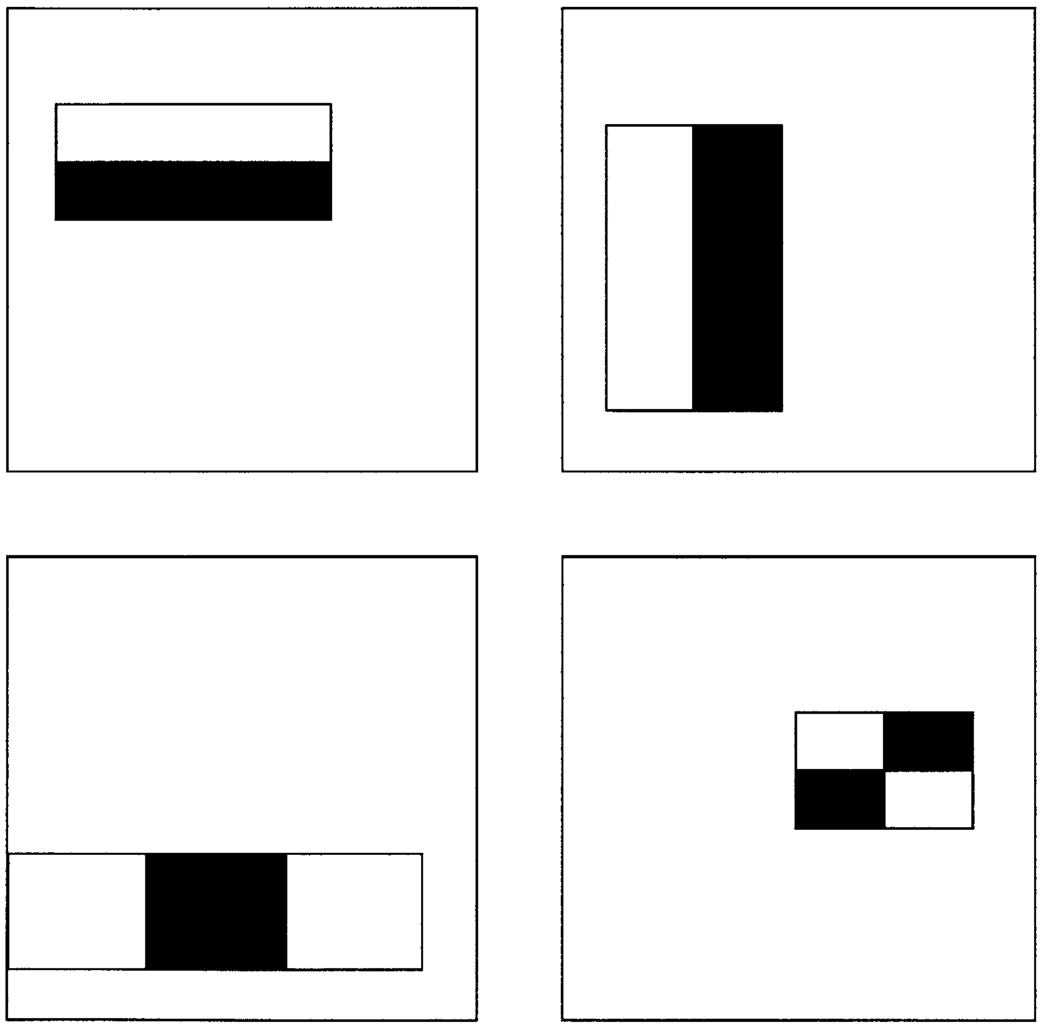

Figure 2-2: The output value of the filters is computed by summing the pixels in the bright rectangles, and subtracting the pixels in the dark rectangles. The figure shows two-, three- and four-rectangle classifiers.

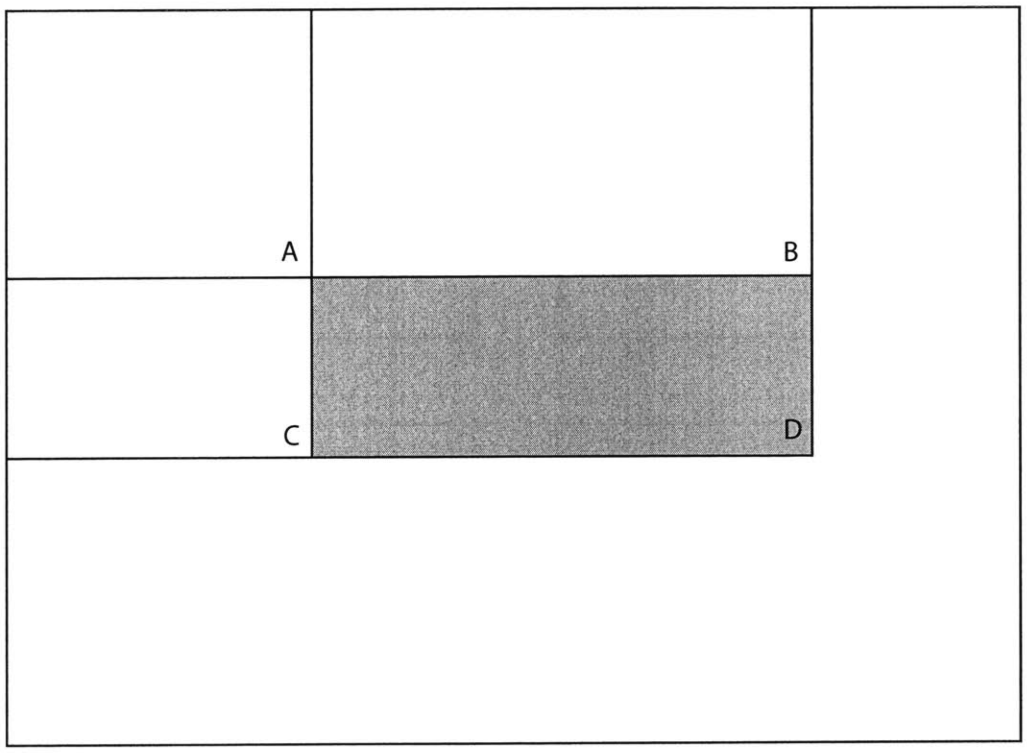

Figure 2-3: The Integral Image allows for summing pixel values in any rectangular region in constant time. The sum of the shaded pixels can be computed by $S = (D) - (C) - (B) + (A)$.

2.3 Cascades

The key to making a detector that is fast is to remove superfluous computations wherever possible. It is natural to use an approach that spends more computational time on "hard" parts of the image, and quickly rules out patches that are obviously not faces. If it is possible to rule out a patch after evaluating only two features, there is no need to evaluate the full ensemble of thousands of features.

Exploiting this fact, the authors of [10] introduced the concept of a cascade. A cascaded detector provides a way of efficiently detecting objects in images, especially in the case where the objects of interest are rare. The cascade consists of a set of nodes, that are evaluated in sequential order for each image subwindow. Only the subwindows that pass the first node are evaluated at the second node, and so one. To be classified as a face, a subwindow needs to pass every single node in the cascade.

Typically, cascades are trained so that each node has a detection rate of at least d, and a false positive rate of at most f. If the cascade has N nodes, the total detection rate is d^N, and the false positive rate is f^N. For a detector with $d = 0.999$, $f = 0.6$, and $N = 40$, the total detection rate is $D = 0.96$, and the false positive rate is $F = 10^{-9}$. In reality, false positive rates between 10^{-6} and 10^{-7} are common, which translates to about one false positive per 640 by 480 image.

To achieve the desired accuracy goals for a given node, more and more features are required. A typical detector might have one or two simple classifiers in the first node, and over 200 per node at the end of the cascade. Since typically less than one window in a thousand pass through the first 10 nodes of the cascades, a large amount of computer time can be spent on the more complicated examples without sacrificing overall speed.

Chapter 3

Cascade Trees

When a set of parallel cascades are detecting visually similar object configurations, considerable performance improvement is possible by merging the independent detector cascades into a tree classification structure. Detectors trained on images of objects in adjacent poses will have cascades with redundant component classifiers. We will show that when we remove these redundancies we can speed up the overall processing time at virtually no cost in detection ratio or false positives. This chapter describes how such a cascade tree is generated from a number of independently trained cascades, by a process of iterative merging. The evaluation of the merged detectors is done in Chapter 5.

3.1 Cascade Merging

When two object classes have similar appearances, it may be the case that their corresponding linear cascades will have similar structure. Figure 3-1 shows the mean face of the 8 different poses used in a set of experiments, along with the first 10 nodes of the corresponding linear cascade. The nodes are illustrated in the figure by the sum of their constituent rectangular simple filters.

The decision of a node on any given image cannot be predicted from this figure, because the individual classifier cutoffs are not shown. It is nonetheless a useful indication of what features were deemed important by the learning algorithm for each

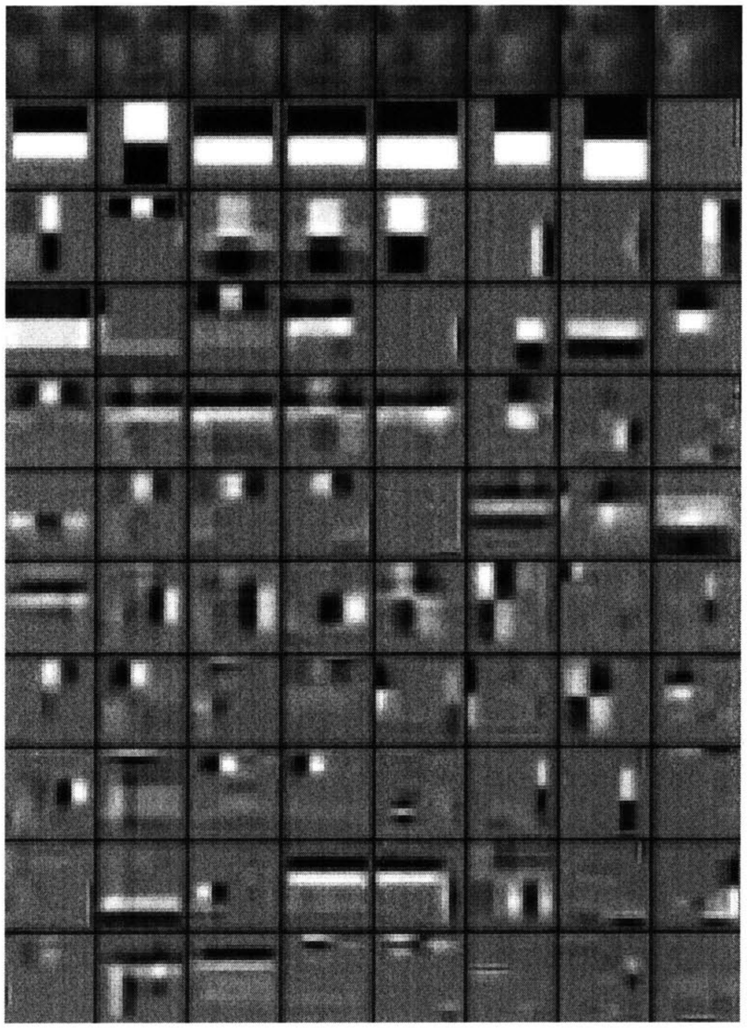

Figure 3-1: Mean faces for the 8 different poses used in a set of experiments, and illustrations of the first 10 nodes of the corresponding cascades. The nodes are illustrated in the figure by the sum of their constituent rectangular simple filters. (The cutoffs for the simple classifiers are not shown.)

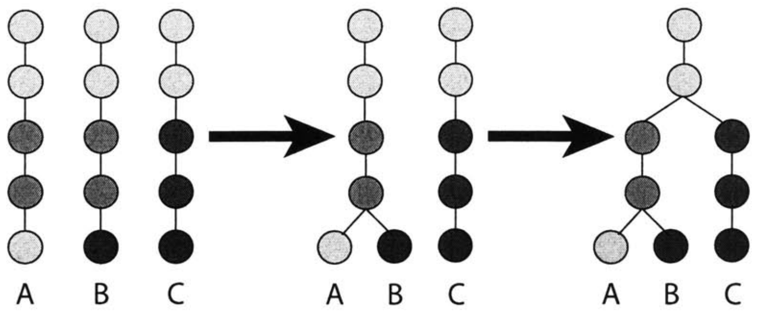

Figure 3-2: A classifier tree is made by iteratively merging cascades.

node. It can be seen from the figure that there is significant redundancy between the cascades. This is exactly the kind of property that can be exploited by merging cascades into a tree.

3.2 Deciding the Tree Structure

In our algorithm, the structure of the tree is determined automatically from the cascades and the training samples in an iterative fashion. We begin by training a linear cascade for each pose in the training set. A linear cascades is a cascade in the Viola-Jones sense, trained to detect only one particular pose, and lacking any branching points.

At each iteration we consider each pair of cascades, and decide at what node they should be merged. We then merge the two cascades that were judged the most similar, up until the node that was decided as the branching point. This process, illustrated in Figure 3-2 is repeated until only one single tree remains.

The similarity measure between a pair of cascades is defined based on how well they perform on their partner's training set. Define two datasets A and B by

$$A = \bigcup_i a_i \tag{3.1}$$

$$B = \bigcup_i b_i, \tag{3.2}$$

where a_i and b_i are samples, and let C_A and C_B be the corresponding cascades.

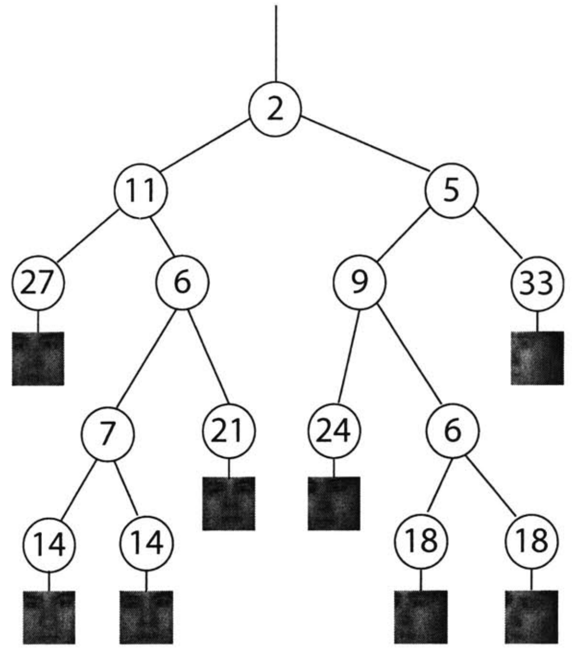

Figure 3-3: The tree designed by the merging algorithm when given 8 cascades of 40 nodes each. The numbers in the circles represent the number of nodes at that point in the tree.

Now define $\ell(C, x)$ to be a function that returns the maximum integer k so that if the cascade C was to be cropped to only depth k, C would accept x as a positive example. Note that a cascade of length 0 accepts anything as a positive example, and removing a node never reduces the detection rate. Both linear cascades and trees can be cropped in this way, since the depth of each node is uniquely defined in each case.

Now we are ready to define the similarity measure between cascades. Let

$$f(A, B) = \frac{1}{2N_A}\sum_{i}^{N_A} \ell(C_B, a_i) + \frac{1}{2N_B}\sum_{i}^{N_B} \ell(C_A, b_i). \tag{3.3}$$

Intuitively, f is the expectation value of how far a training sample of one cascade will make it in the other cascade.

The index of the node to merge the two cascades at is computed as

$$n_{AB} = \lambda f(A, B), \tag{3.4}$$

where λ is a tunable parameter that controls how early in the tree branching occurs. When $\lambda = 0$, all branches occur before the first node, so the resulting tree is equivalent to running each linear cascade in parallel and combining the results. When $\lambda = \infty$ the resulting tree is one single linear cascade.

After the algorithm has decided which two cascades to merge and where, it performs the actual merge. For the purpose of determining the structure of the tree, the merged cascade C_AB consists of either the nodes from C_A or the nodes from C_B up until the branch point, and nodes from both cascades in parallel after it.

3.3 Node Retraining

Once the final structure of the tree is determined, nodes are retrained on the data sets they are meant to recognize. The new nodes are trained to have detection rates on the merged datasets that are as good as the nodes in the linear cascade have one the single-pose dataset. This can be done at the expense of getting more false positives at any given node in the detector.

Taking longer to eliminate false positives in turn slows down the overall system, since more computational effort is required. Note however that the overall false positive rate can be controlled by adding new nodes at the very end of the detector, so the only expense we incur is in the average number of classifiers that need to be evaluated per sample window. Nodes at the end of the detector are evaluated for very few false positives, and hence adding an extra node does not significantly slow down the detector.

When the merging is too weak, more classifiers than necessary are evaluated because similarities between cascades are not being fully exploited. When the merging is too strong, the resulting classifier needs to handle poses that are fundamentally different, which means the false positive rate will decrease only slowly as a function of classifier length. For some intermediate value of λ performance is maximized. In the current implementation we try several different intermediate values and choose the best tree. Figure 3-4 summarizes the algorithm used to design and train the tree.

1. Given N sets of positive examples A_i, a single set of negative examples B, the initial cascade depth k, and the minimum detection rate d_j and maximum false positive rate f_j for $1 \leq j \leq k$.

2. For each i, train a linear cascade C_i using direct feature selection with A_i as the set of positive examples and B as the set of negative examples.

3. While more than one cascade remains, do

   (a) for each pair $\{A_i, C_i\}$, $\{A_j, C_j\}$, compute

   $$c_{ij} = min(\lambda f(A_i, C_i, A_j, C_j), branch_i, branch_j)$$

   where

   $$f(A, C_A, B, C_B) = \frac{1}{2N_A} \sum_i^{N_A} \ell(C_B, a_i) + \frac{1}{2N_A} \sum_i^{N_B} \ell(C_A, b_t).$$

   and $branch_i$ is the index of the first branching node in C_i.

   (b) Find i, j so that c_{ij} is maximized. Define $n = \lfloor c_{ij} \rfloor$.

   (c) Merge the sets of positive examples A_i and A_j into one large sample set A

   (d) Merge the cascades C_i and C_j into a new cascade C. Run C_i and C_j on the dataset A, and keep the first n nodes of the cascade that performs the best. Add a branch point after node n, and paste in C_i and C_j starting from nodes $n + 1$.

   (e) Remove $\{A_i, C_i\}$ and $\{A_j, C_j\}$ from the pool of cascades. Add $\{A, C\}$ to the pool.

4. Finally, retrain each node of the tree using the associated subset of N data, and the subset of B which is not eliminated by earlier nodes in the tree as the negative training data. For a node of depth i, use d_i and f_i as training parameters.

Figure 3-4: Summary of the algorithm for merging detection cascades into a single tree classifier

Chapter 4

Pose Estimation

A face detector is usually used only as the first step of any computerized vision system. The final objective is usually another. For example, face recognition, automatic red-eye reduction and face tracking tasks all include face detection modules. However, most face detectors give a very coarse estimate of the pose of the face, or none at all. In this chapter, we show how to build a pose estimator based the same principles as cascades and cascade trees. The next chapter contains an evaluation of the performance of this approach compared with other approaches for pose estimation.

4.1 Training Data

To train a face detector, all that is needed is a set of example faces in different poses. It is not necessary to label the poses accurately, since that information wouldn't be used anyway. To properly train a pose estimation system, much more care must be taken to get properly labelled data. One way of accomplishing this is to capture data from multiple cameras at the same time. The data from [1] was captured simultaneously by 48 cameras on a grid, and that data is what we used to train the system.

This data was used to train 48 cascades, in the following way. For each pose displayed in Figure 4-1, select that pose and all immediately surrounding poses. This results in 4 selected poses when the original pose is in a corner of the pose grid, but 9 selected poses when it is in the center. All the data from these selected poses are now used as positive examples, and all the data from the other 39 to 44 poses are used as negative examples. The result is 48 cascades trained for 48 different poses. From here on, when we talk about the pose of a cascade, we are referring to the originally selected pose. This is the same as the center among the poses used as positive training data.

4.2 Pose Estimation Algorithm

There are many different ways of doing pose estimation, including complex feature-based methods [5]. Here we've opted for a simple approach, that takes advantage of cascade flexibility. Keeping in mind that a cascade does not necessarily have to be a binary classifier, we modify a standard linear cascade into one that instead of purely rejecting or accepting a sample returns a number signifying how far into the cascade the sample made it.

A good number to return for a sample is the fractional depth of the node that first rejected it, with 1.0 being returned if every node in the cascade accepts it. Since the rejection rate is almost the same at every node, this can be thought of as a score proportional to the logarithm of the probability that the sample is of the same class as the positive samples used to train the cascade.

If a sample in some pose P is evaluated by a detector for the same pose, one would expect the sample to make it deep into the cascade, or even all the way to the end. In reality, this is not always the case. Figure 4-2 shows sample output of the 48 pose detectors on images of a single person in 48 poses. The images were captured by the same array of cameras as the training data, but were held out from the training phase.

The 48 small 8 by 6 subimages represent detector outputs on each of the 48 input images. The top left pixel in every subimage corresponds to the detector for the top left pose showed in Figure 4-2, and so on. It follows that if this was a perfect pose detector, the Nth image of the Mth row would have a black pixel, symbolizing a return value of 1, at position (N, M). In addition, all other pixels would be white,

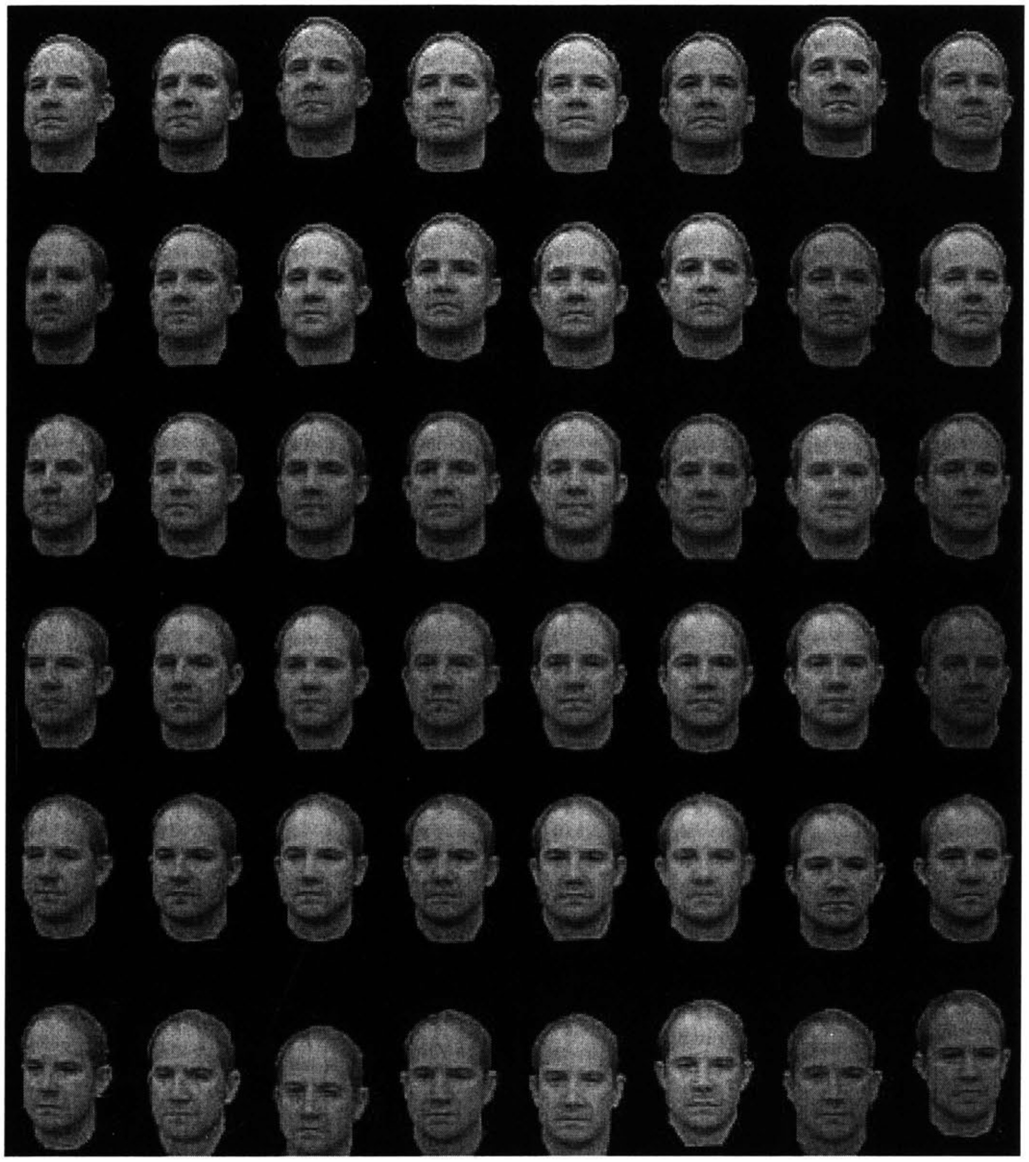

Figure 4-1: The 48 poses used by the pose estimation system

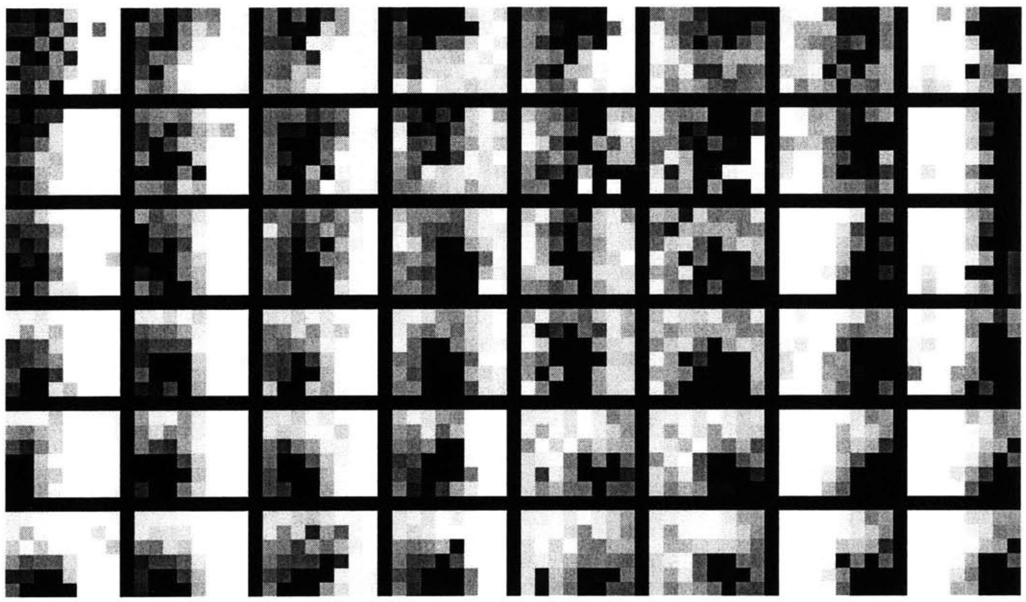

Figure 4-2: Output of the 48 pose estimation cascades on 48 sample input images. Each subimage is the output of the 8 by 6 array of cascades, with one pixel corresponding to each cascade. The input sample used to generate the subimage at position (x, y) was of the same pose as the cascade at position (x, y) in each subimage. Darker pixels correspond to better cascade matches.

symbolizing a cascade return value of 0.

To compute the final pose estimate, we use a two step procces. First, we compute a weighted sum of the poses of the cascades. The weights are the outputs of the cascades on the input sample. Explicitly, the intermediate pose P' is computed by

$$P'(I) = \sum_j C_j(I)p_j, \tag{4.1}$$

where I is the input sample, C_j is a set of cascades, and p_j is the corresponding set of poses. However, this pose estimate tends to be biased towards the center of the pose space, since very few cascades will in general return really low numbers. To compensate for this, we apply a linear transformation to each pose coordinate,

$$P_i(I) = a_i P'_i(I) + b_i. \tag{4.2}$$

The constants a_i and b_i are learned from the training data by minimizing the least-squares error during training. In a combined face detection and pose estimation task, a cascade tree would be used to detect the faces in an image, and then the pose estimator would be run on each detected face. The approach for pose estimation presented in this chapter is general enough to handle both two- and three-dimensional pose estimation, although we will only use it to estimate two pose parameters at a time in this thesis.

Chapter 5

Experiments and Results

Cascade trees present several advantages to traditional linear cascades. In this section, we'll describe a set of experiments conducted to highlight different features of the tree approach. The first experiment uses a training set of 48 almost frontal poses, under one single simple illumination condition. We will show that detection performance can be dramatically increased while keeping the false positive ratio and the speed of the detector fixed.

The second experiment uses the PIE database as training data. This data set is very diverse, with many different poses and lighting conditions, and it is impossible to train a single efficient detector. We show that the tree approach yields almost exactly the same detection performance as 8 independently trained cascades, but with a speedup of more than a factor of 2. In both this and the previous experiment, we used the negative samples from [10]. This set of 9492 files was originally obtained by crawling the web, and randomly selecting images that do not contain faces. Altogether, these images contain a total of 2,440,328,628 subwindows.

In the third and final experiment, we evaluate the pose estimation algorithm introduced in Chapter 4. We use the same data as in the first experiment to train the pose estimator. We compare estimated pose with ground truth and pose computed using normalized cross-correlation.

5.1 Detection Rates

A somewhat surprising benefit of cascade trees is an increase in the detection rate, compared to a single detector. This feature is the most prominent when there are many linear cascades, perhaps because each has a somewhat independent probability of detecting a difficult face in an image. This experiment shows a case where the detection rate increase is significant.

All our experiments use the Direct Feature selection method [11] to train nodes. In the standard AdaBoost-based approach, each node decides whether or not to pass a sample on to the next node based on comparing a linear sum of the votes of a set of simple classifiers to a cutoff. Using the Direct Feature selection method, the weight in front of each simple classifier is 1, and the overall decision is made by majority vote. We used this approach because of the drastic reduction in training time needed per node.

5.1.1 Training Set

The training set for this experiment was collected for [1], using an array of cameras pointing at a subject. The subjects in the training set are shown in Figure 5-1, and the different poses are shown in Figure 4-1. Note that the majority of subjects are Caucasian males, which most likely hampers the ability of the detector to find faces of individuals not in this category. In addition, there is no variation in lighting. The negative training set was the samples from [10].

5.1.2 Performance

This experiment was subdivided into two tasks. The first task is face detection in an office setting, where lighting conditions are similar to the training data. Figures 5-2, 5-3 and 5-4 show detection ratios, false positive rates, and average number of simple features evaluated per subwindow for this office task. Some example images from the first task are shown in Figure 5-5. Note especially that the detection rate is highest for small values of λ, whereas the detector efficiency is highest for large values of λ.

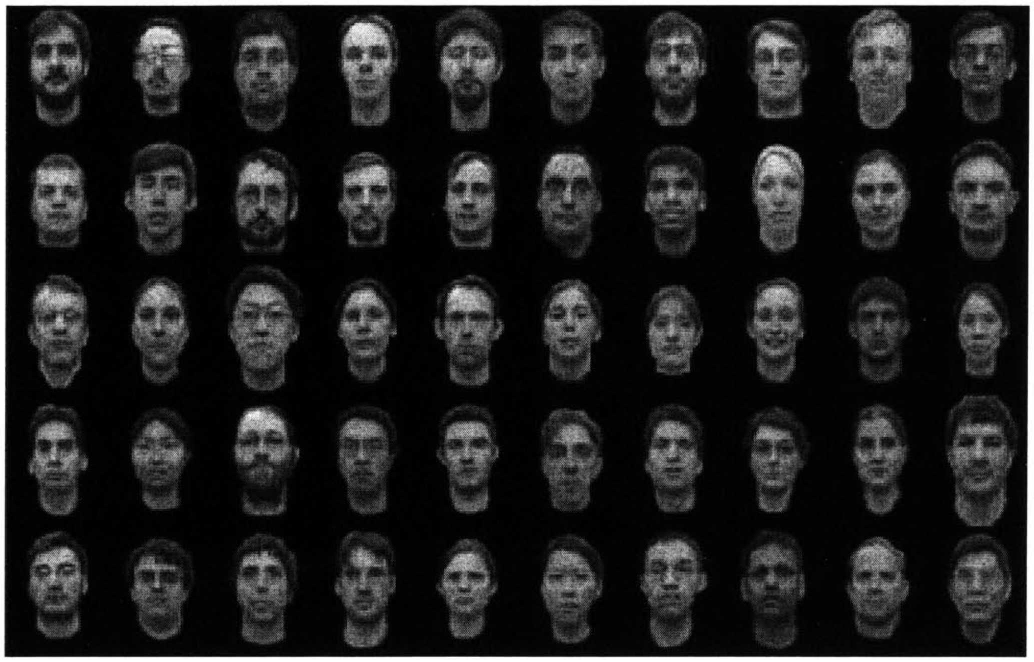
Figure 5-1: The subjects used to train the first set of cascade trees

Around $\lambda = 1.0$, we can get close to maximum detection performance while keeping close to maximum efficiency.

The second task is face detection on images in the MIT+CMU test set[7], which is commonly used to evaluate face detector performance. This test set contains images that are blurry, faces that are not frontal, and various severe illumination conditions. In addition, several test images are included simply because many detectors find false positives in them. Although the performance of our detector on this data set is poor compared to other detectors [6], which can achieve detection rates slightly over 90%, the trend of increased performance as λ decreases is clearly visible. Figure 5-6 shows detector output for two values of λ, with the detection ratios jumping from 61% to 82%, keeping the number of false positives fixed. These two trees were trained on exactly the same data.

The results of a more systematic evaluation is shown in Table 5.1. When the face detectors were evaluated, special care was taken to make sure the number of false positives were kept roughly constant, to make sure that the results did not simply

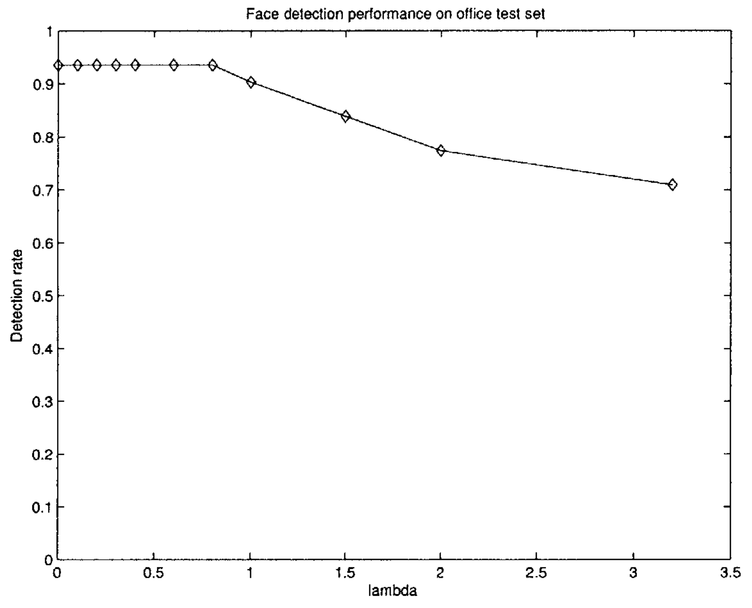

Figure 5-2: Detection ratio for the office task

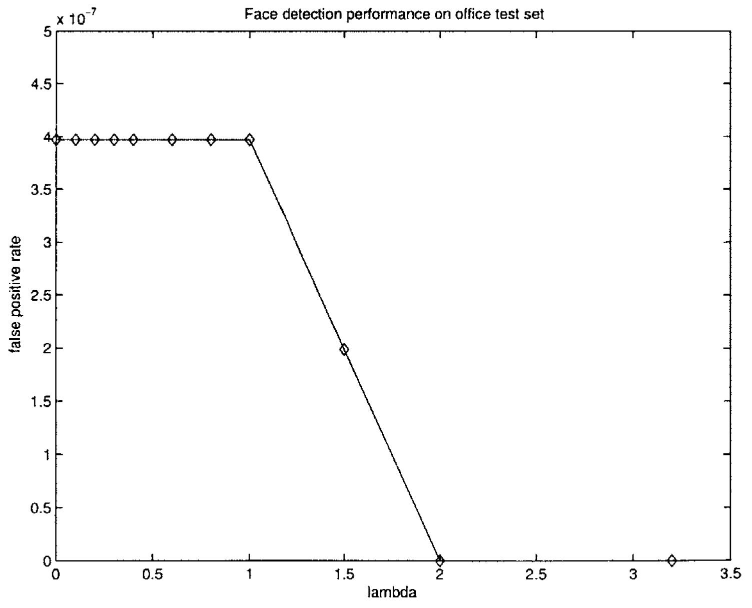

Figure 5-3: False positive rate for the office task

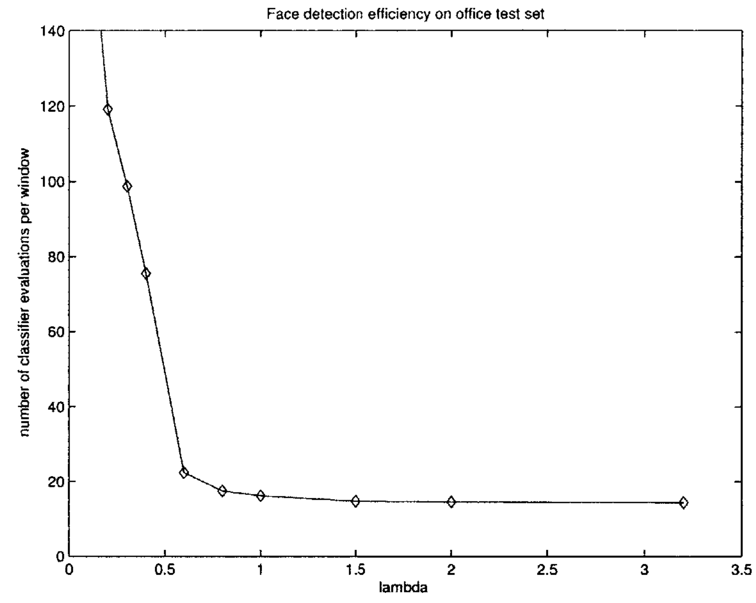

Figure 5-4: Detector efficiency for the office task

λ	**Detection ratio**	**False positives**	**Classifiers evaluated**
0.0	55.2%	82	183.36
0.8	53.6%	73	19.72
1.0	50.5%	64	17.31
1.5	44.6%	62	15.66
∞	37.9%	66	15.46

Table 5.1: Performance of the detector on the MIT+CMU dataset. The dataset contains a total of 130 images with 507 faces, and 68,963,690 subwindows.

portray a shift along the ROC curve of the detectors. This was done by removing nodes at the end of the detector, or at the end of each branch in the case of a tree, until the false positive rate was high enough.

5.2 Face Detector Speed

One of the original motivations for the concept of cascade trees was a desire to remove redundancies between detectors for similar classes of objects. If such redundancies could be removed, the resulting detector should be faster than the original. To gain

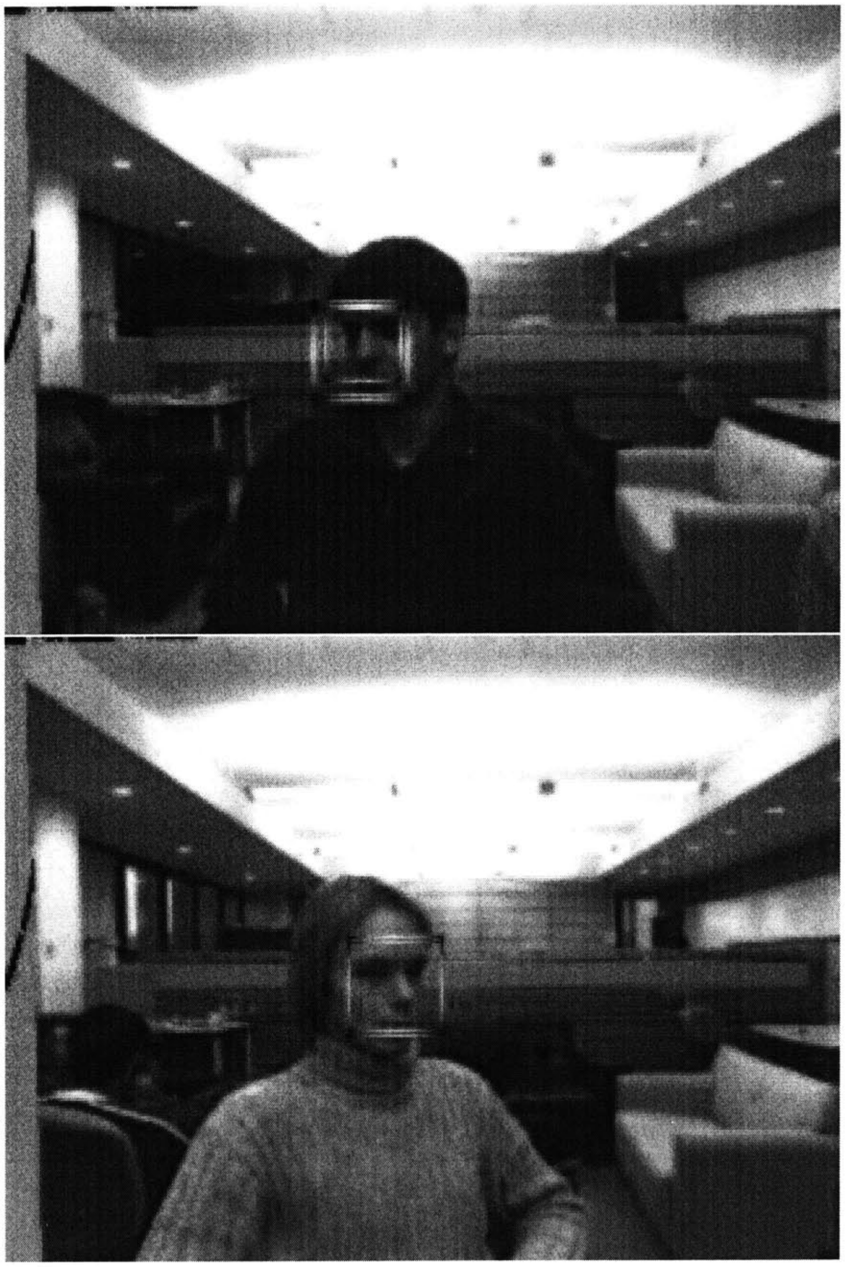

Figure 5-5: Example images from the office task. The subjects shown were not represented in the training set, and the poses detected in these images are outside of the range of the training data.

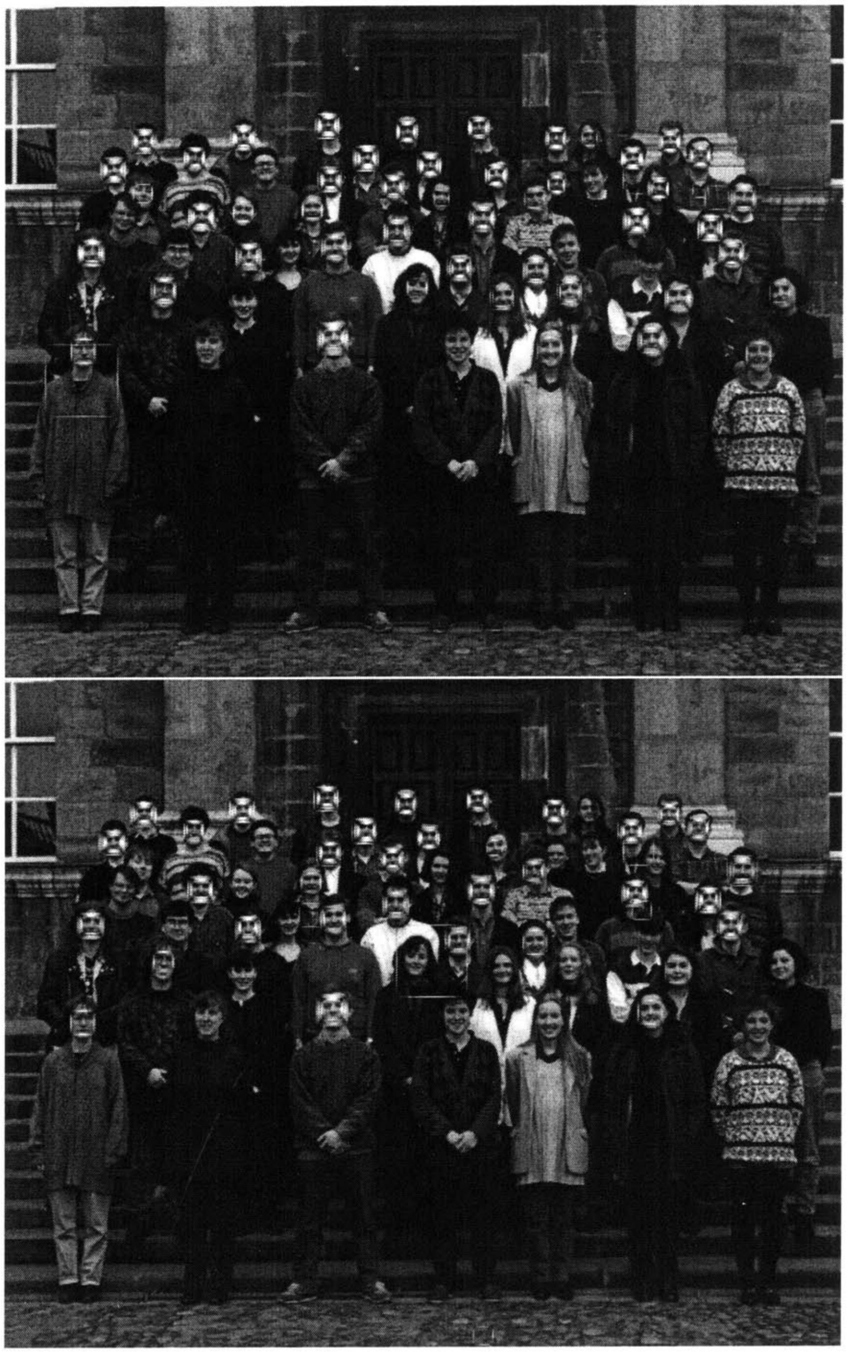

Figure 5-6: Sample output from a detector trained on face data from an array of cameras. The top picture was generated by a tree with $\lambda = 1.0$, and the bottom one by one with $\lambda = \infty$. Note that even though the false positive rates are almost the same in both images, the detection ratio is far higher for the topmost one. These images each contain roughly 3.5 million subwindows.

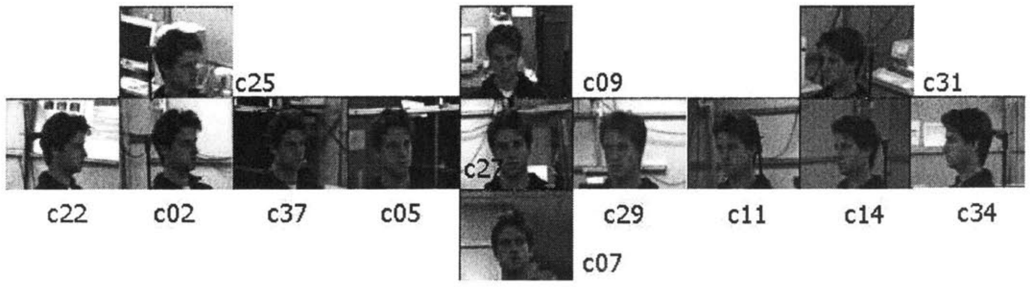

Figure 5-7: The different poses used in the PIE database

the maximum improvement, an exhaustive search over all possible trees is most likely unavoidable. It is, however, also unpractical.

In Section 3, we introduced a way of designing a tree based on a set of already trained linear cascades. We show here that the parameter λ used to tweak the structure of the tree has a natural value of 1.0, at which point detector speed is maximized. In the previous experiment, a higher value of λ would guarantee a faster detector. In this section, however, we show a case where the relationship between λ and detector efficiency is more complex.

5.2.1 The PIE Database

To fully appreciate the results, it is necessary to have some understanding of the PIE[9] database, which was used for training. The database has 68 people in 13 different poses, as shown in figure 5-7. We only used the 8 rightmost poses in our experiments, using the fact that a detector trained for left profiles can be flipped around the vertical axis to detect right profiles.

It is also worth noting that the lighting conditions are severe. In fact, some illuminations were deemed too dark to be useful for training. Figure 5-8 shows one test subject in one pose under all 22 used lighting conditions. Note that once the face is scaled down to 24x24 pixels, it becomes very difficult even for humans to determine whether or not the little image represents a face.

In the interest of fairness, it should be mentioned that the PIE database was not

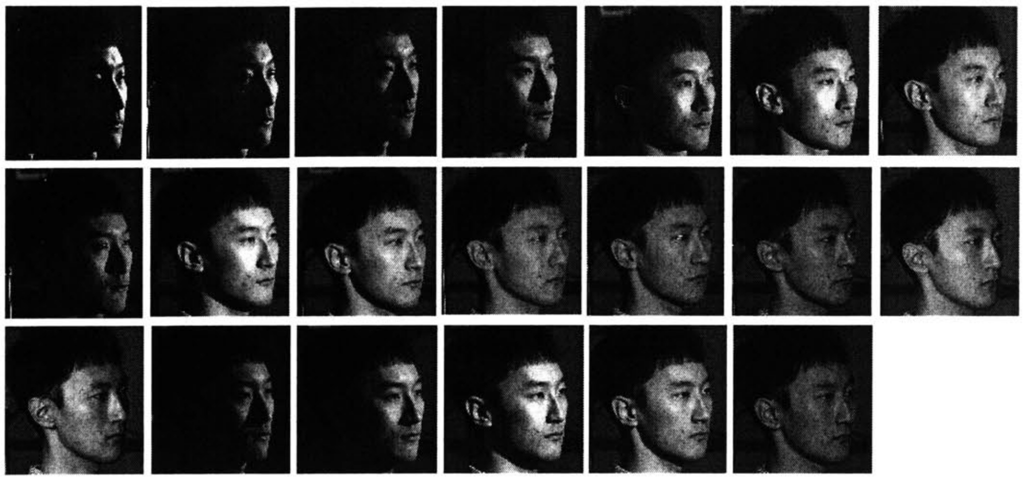

Figure 5-8: The different lighting conditions used in the PIE database

intended to be used as a training set for a face detector, but rather as a test set. Any training data set should mirror the statistical distribution of faces, poses and illumination conditions the detector is expected to see when operating. For the most generic face detector, the ideal data set would be composed of randomly selected images from the world wide web.

5.2.2 Early Rejection and Multiple Cascades

To understand the principles determining the speed of a face detector, it is instructive to look at the first few nodes of some detectors trained on data sets with differing variations. Figure 5-9 shows the performance of three different cascades. The first one was trained on a data set with only interpersonal variation, the second one also had illumination variation, and the third one had both subject, illumination, and pose variation. The rate of false positives remaining after each node is shown below the node, and the average number of classifiers evaluated per subwindow is listed at the very bottom.

The more variation there is in the training data, the harder it is to find good simple classifiers to differentiate positive from negative samples. As a result, a lower percentage of negative samples gets rejected at each node. Since nodes get increasingly

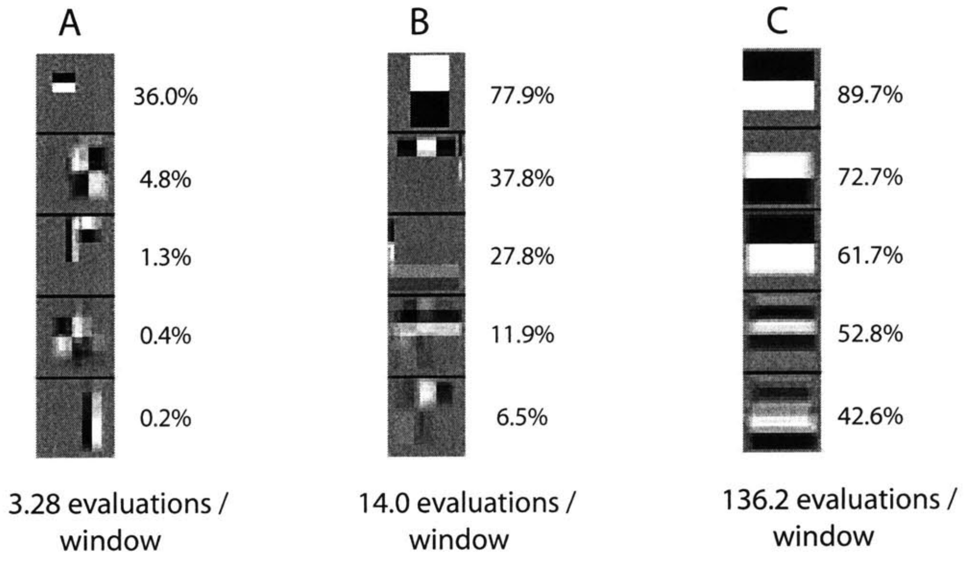
A
36.0%
4.8%
1.3%
0.4%
0.2%
3.28 evaluations / window
B
77.9%
37.8%
27.8%
11.9%
6.5%
14.0 evaluations / window
C
89.7%
72.7%
61.7%
52.8%
42.6%
136.2 evaluations / window

Figure 5-9: Performance of the first five nodes in different detectors. Detector A has subjects in the same pose and illumination, detector B was trained for illumination variations, and detector C for both pose and illumination variations. The number of classifiers per node is fixed at 1, 5, 5, 11 and 11.

complex and expensive to evaluate deeper into the detector cascade, the earlier a sample can be rejected, the faster the cascade is in the average case.

Cascades trained on only a single pose are comparatively good at rejecting false positives. However, to span the entire range of poses, we need multiple cascades, all evaluated separately. This multiplies the average number of features needed to reject a non-face by the number of cascades. To maximize performance, it is necessary to find the right balance between reducing training data variation and minimizing the number of cascades.

5.2.3 Tuning Detector Speed

Tuning λ affects the structure of the tree, which in turn affects the efficiency of the detector. Figure 5-10 shows how detector efficiency, measured as number of classifiers evaluated per subwindow, varies with λ. For this training set, which contains significant pose and lighting variations, the single-cascade $\lambda = \infty$ or the separate-cascade $\lambda = 0$ options are both roughly a factor of 2 slower than the $\lambda = 1$ cascade tree. For this particular training set, $\lambda = \infty$ does not result in a useful detector, even when 40 nodes are allowed. Figure 5-11 shows sample detection performance for several values of λ on the MIT+CMU test set.

5.3 Pose Estimation

A cascade tree is not necessarily a binary classifier. Each terminal node can be given an independent ID, which would be returned by the detection algorithm if that node in the tree were to be reached. If several end nodes were to be reached, a list of IDs might be returned. Unfortunately, ordinary cascades, trained with only non-faces as negative examples, are not good at pose discrimination. This section describes the results of an evaluation of the pose estimator described in Chapter 4. We also compare the results to normalized cross-correlation.

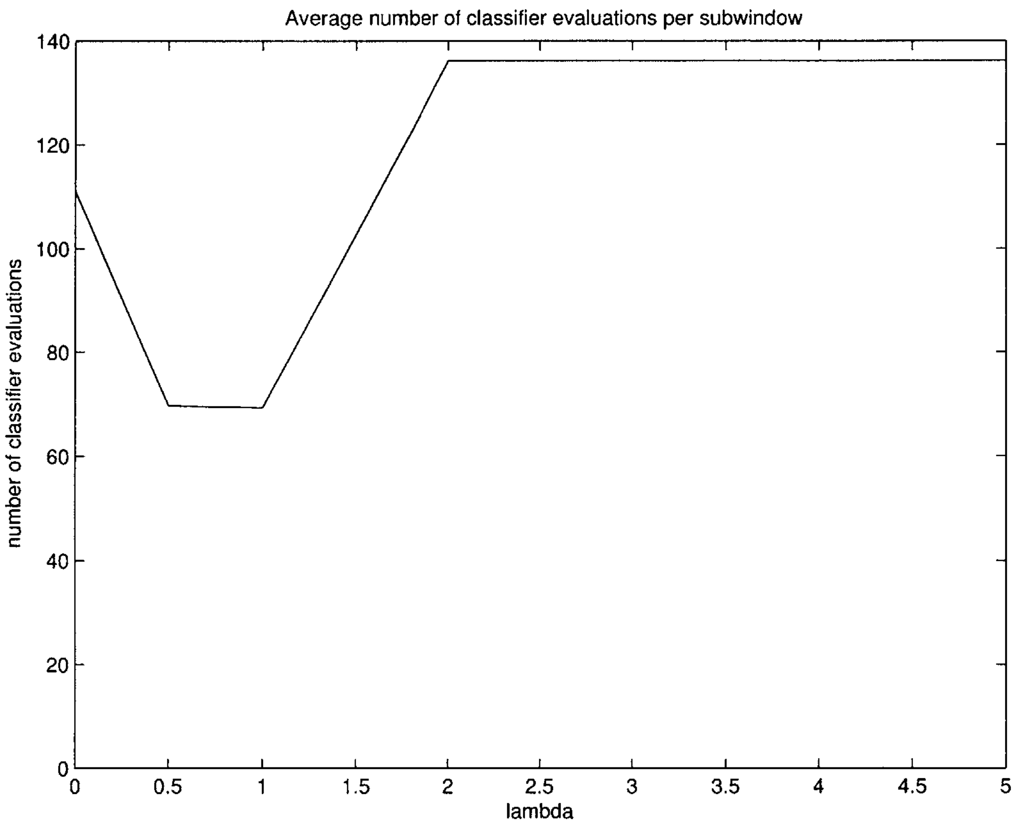

Figure 5-10: Detector efficiency, measured in the average number of classifiers needed to reject a non-face, for various values of λ. The detectors are adjusted to have the same detection rates and false positive ratios. Around $\lambda = 1.0$, efficiency is maximized.

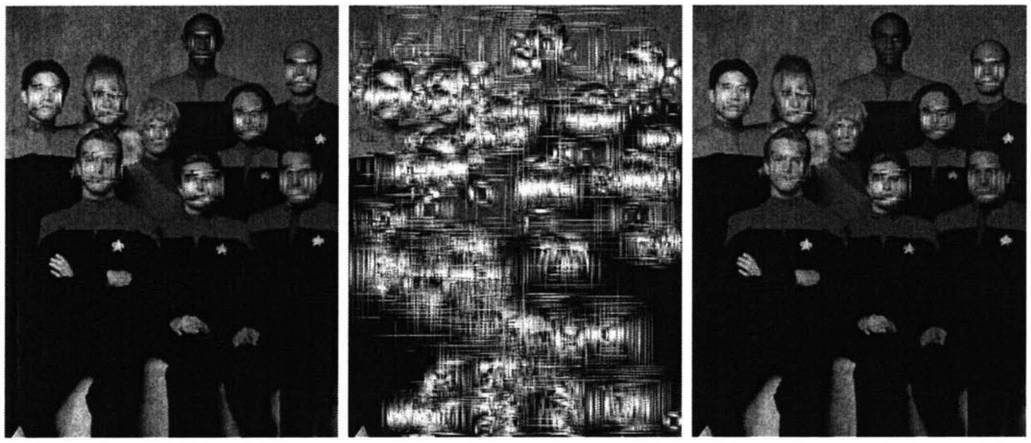

Figure 5-11: Results of multi-pose detectors on image with unseen faces. The left image shows baseline result from set of independent cascade classifiers, equivalent to $\lambda = 0.0$. The center image shows result from the extreme case of merging all cascades into a single chain, or $\lambda = \infty$. The right image shows the result with a cascade tree designed by our algorithm, with $\lambda = 1.0$ The performance with the cascade tree is qualitatively similar despite considerable savings in computational cost. Note, however, the one lost detection at the top of the image.

	Horizontal	Vertical
a	1.62	2.70
b	-1.17	-16.40

Table 5.2: Parameters for the pose estimation transformation.

5.3.1 Task Description

In this experiment, we again used the data from [1] to train 48 cascades. As positive data for a given pose, we used the sample images from that pose and all immediately surrounding poses. As negative data, we used the remaining sample images. Note that the negative data set from [10] is not used. A test set of 192 images of 4 individuals were held out from the training set, and the pose of those images is then estimated by the system. As a comparison, we also implemented and evaluated a pose estimation system based on normalized cross-correlation. The comparison uses the mean face of each of the 48 poses as templates, and the best template match becomes the **base pose**. The final pose estimate is an adjustment of the base pose, computed by fitting a second-degree polynomial to the correlation scores of the three closest poses in the x and y directions, and finding the maximum of that polynomial.

5.3.2 Results

Table 5.2 lists the parameters used for the linear pose estimate transformation, as described in Eq. 4.2. These parameters were determined using only the training data. As expected, they encode for a stretching of the pose space, cancelling the bias to return a pose in the central region of the field.

Figure 5-12 shows the final results of the pose estimation task. It shows that the cascade approach is fairly robust, with no pose estimates being wrong by more than 10 degrees. Note that most pose estimates are in fact within 5 degrees from the true pose. In the training data, poses were also 5 degrees apart, so this error corresponds to an error of one discrete pose. Table 5.3 summarizes the mean and variance of the pose estimation error for the two approaches.

Method	Horizontal error	Vertical Error
Array of Cascades	-1.78 ± 3.37	-0.31 ± 2.66
Normalized Correlation	-5.42 ± 6.21	-4.20 ± 7.88

Table 5.3: Mean and variance of the pose estimation error for the two approaches

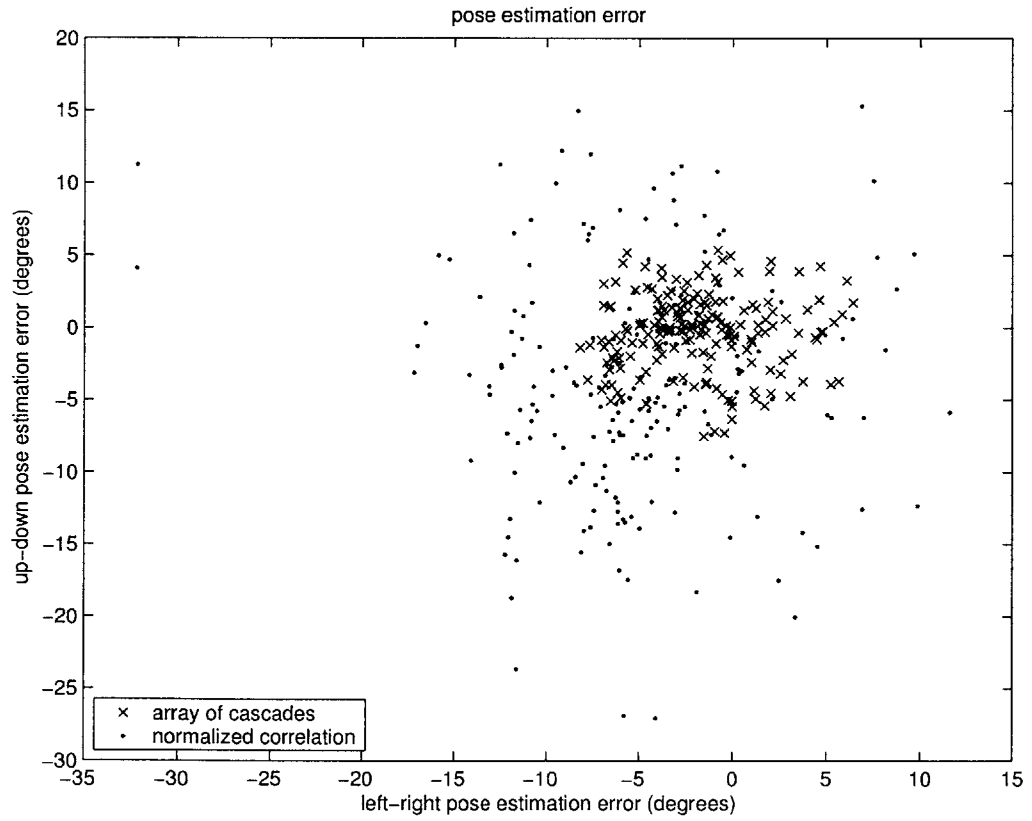

Figure 5-12: Pose estimation error for the array of cascades approach and the normalized cross-correlation approach. A pose error of 5 degrees corresponds to the difference between two neighboring poses in the training grid.

Chapter 6

Discussion

Creating face detectors by boosting simple classifiers is a fascinating topic. During my experiments, I've come to realize the power of this approach. State-of-the-art face detectors such as [6] can scan video data in real time, with amazing accuracy. This has opened up a sea of other possibilities. Tasks that need face detection as a preprocessing step can now get complete, pre-built modules that use up very little computing time.

In my own experiments, I've consistently seen excellent performance. In particular, the performance on the training data is always remarkable, both when discriminating faces from non-faces and when discrimination faces of different poses from each other. The real moment of truth comes when the detector is applied to a test set. If the training set was well balanced, performance on the test set will sometimes not be worse than performance on the training set.

It's also amazing that one single feature evaluation is able to rule out 40 to 60 percent of all non-face subwindows. As was hinted at in [11], the power of this approach most likely comes from the sheer number of possible filters used in the classifiers. All filters will have some distribution of outputs for faces and non-faces, and not all distributions can be random or uniform. In fact, statistically it would be strange if some were not useful for classification. What kind of boosting is used, be it Adaboost[2], Direct Feature Selection[11], GentleBoost[3] or something else, matters less.

The main difficulty is finding a data set with the right kinds of variation. We showed in Section 5 that large lighting variations are not useful and can even be harmful. On the other hand, even a training set consisting mostly of Caucasian people not wearing glasses yields a detector that half the time detects African Americans wearing sunglasses. Another somewhat surprising result is that a detector trained for an office environment with data from only 60 people, is capable of finding virtually any face in the same lighting condition and pose range. Even at 10 degrees outside the training pose range, detection rates are still around 60 to 70 percent.

In this thesis, we have shown that cascade trees have two benefits. First, if the training set is too diverse to make a single detector efficient, using a cascade tree will speed up the overall detector without sacrificing detection performance. Even if the data set is not diverse, using a cascade tree will not hurt the efficiency of the detector. The speedup is accomplished by exploiting whatever statistical similarities there are between the different object classes. When those redundancies are exhausted, the tree will branch off into different cascades. The new, more narrowly defined cascades are more efficient at rejecting non-face samples, since the statistical variations in their training sets are lower. The resulting detector can be several times more efficient than a simple linear cascade.

The second benefit of cascade trees is that a tree may have a significantly higher detection rate than a linear cascade trained on the same data. The cause of the performance increase is the randomness inherent in the cascade training. If the training set densely samples the pose space, a linear cascade trained on one pose will do well on data that is meant to be handled by its immediate neighbors in the pose space. Sometimes it will do just as well as the cascades which are meant to handle the data. There's always a little randomness in detecting a face. If two detectors are completely independent with 90 percent accuracy, and overlap in the samples they can handle, the combined detector might be up to 99 percent accurate. In reality, the improvement will not be that big, but the results indicate that the accuracy might be around 94 or 95 percent. In either case, having a set of different detectors will minimize random rejections.

We have also shown that cascades are useful for pose estimation. However, to fully take advantage of their power, the cascades must be trained with data from other poses as negative examples. When simple non-faces are used as negative examples, there is no penalty for a cascade to detect faces in other poses, and on occasion it will. This is the very cause of the increased detection rate of cascade trees; the set of recognized faces for each branch of the tree overlap slightly, and even if one detector misses a face, another might catch it. But when doing pose estimation, we need more predictable cascades. The results presented in this thesis show that using data from other poses as negative examples during training gives us the reliability we need. When averaged over a set of cascades, the resulting detector is both robust and accurate.

In conclusion, we have in this thesis explained the concept of boosted cascades of simple features, and why that is a very appropriate method for face detection and other rare event detection tasks. We have also shown how to build a cascade tree from a set of cascades, and how such a tree can improve both the efficiency and the accuracy of an overall detector. We have also shown how to build a pose estimator using cascade technology, and evaluated a system based on our algorithm on a real pose estimation task. The boosted cascade is a very powerful and versatile tool, and it is a concept important enough for any student of computer vision should be familiar with it.

Bibliography

[1] C. M. Christoudias, L. Morency, and T. Darrell. Lightfield appearance manifold. In *ECCV*, 2004.

[2] Y. Freund and R. E. Shapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory: Eurocolt*, 1995.

[3] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 28(2):337–374, 2000.

[4] B. Heisele, T. Serre, M. Pontil, and T. Poggio. Component-based face detection. In *CVPR*, 2001.

[5] T. Horprasen, Y Yacoob, and L. S. Davis. Computing 3-d head orientation from a monocular image sequence. In *IEEE Conference on Automatic Face and Gesture Recognition*, 1996.

[6] M. Jones and P. Viola. Fast multi-view face detection. In *MERL Technical report TR2003-96*, 2003.

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

[8] H. Schneiderman and T. Kanade. Object detection using the statistics of parts. *International Journal of Computer Vision*, 2002.

[9] T. Sim, S. Baker, and M. Bsat. The cmu pose, illumination, and expression (pie) database. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(12):1615–1618, December 2003.

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

[11] J. Wu, J. M. Rehg, and M. D. Mullin. Learning a rare event detection cascade by direct feature selection. In *SCTV, in conjunction with ICCV*, 2003.

6075-55