Chapter 8

Detection of sparse models: Counting

The decomposable graphs presented in the previous chapter have been partially successful in automatic anatomy detection in medical imaging. They rely heavily on finding instances of all the local features at the correct location on the object, and are sensitive to occlusion or other sorts of noise. Moreover when searching for objects at a large range of scales in large images, dynamic programming becomes very slow. Finally if several instances of an object can be present in the image dynamic programming becomes significantly more complicated.

The solution involves a coarser model. After detecting all instances of the local features in the image, instead of directly trying to find arrangements satisfying the complex geometrical constraints, try to find arrangements satisfying a simpler set of constraints which is consistent with the original one. Then check if the more complex constraints are satisfied. The idea is illustrated in figure 8.1. The original model involves constraints on relations between pairs or larger subsets of the points. Various admissible configurations are presented in the top three panels. Now replace this with a simple model which only constrains the locations of the features relative to a central point, independently of each other, as illustrated in the bottom panel of figure 8.1. A point in the image becomes a candidate 'center' if a sufficient number of the features are found in their corresponding regions.

At each such candidate center search through the instantiations in $\Theta^{(0)}$, and pick the one for which the most features are found at their expected location. If this number is below a threshold, rule out the candidate center. For those candidate centers which remain, the locations of the features which were found together with the *expected* locations of those which were not found provide an instantiation of the model.

Here again we have a coarse to fine procedure. First we use a model with a much coarser set of constraints to find candidate centers. At this stage we expect to hit all correct locations together with a number of false positives. Then a more detailed instantiation is computed both in order to filter out false positives from the previous stage, and to provide the full description of the detection. Subsequently an even more refined process can be implemented such as one of the deformable models described in earlier chapters with the detected instantiation serving as an initial point.

In order to detect the object at a wide range of scales, say 4:1, reprocess the image using the exact same algorithm at a number of smaller resolutions. For example if the original range of scales of the detection scheme is $\pm 25\%$, subsample the object at scales $0.75, 0.56, 0.42, 0.32, 0.24$, and rerun the detection algorithm at each new resolution. Thus the model is given for the smallest range of scales at which the object is detected. The method described here is very efficient and tpyically takes under .5 second for processing a standard 240×320 image, at all resolutions, on a PENTIUM III, 700 Mhz.

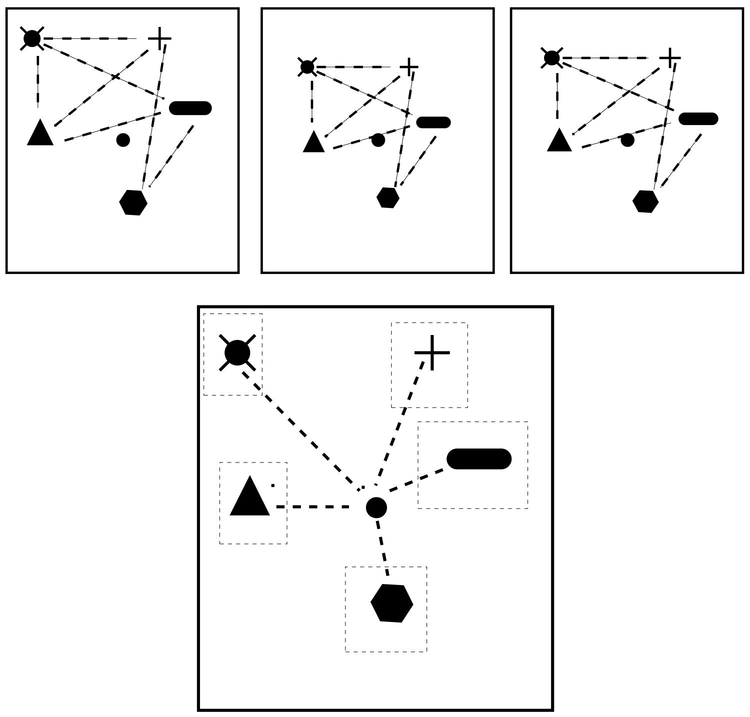

Figure 8.1: Top: Three feature arrangements consistent with complex set of constraints. Bottom: Coarse model. Features can be found anywhere in their respective regions relative to a center point.

8.1 Detecting candidate centers

Here we will use the instantiation set Θ defined by 6.1 and take a uniform prior on Θ. The on object probability p_o is assumed less than one although much larger than p_b. Maximizing equation 6.5 involves finding values of θ for which $n_y(\theta)$ is greater than some threshold τ. Specifically, find pairs (x_c, θ) with $x_c \in L$ and $\theta \in \Theta^{(0)}$ such that

(D) There exist indices $i_1, \dots, i_\tau$ for which $X_{i_j}(x_c + \theta_{i_j}) = 1$ for $j = 1, \dots, \tau$.

There are several ways to find such instantiations, all of which involve counting the number of features of a specific type present at a specific location in the image.

The brute force approach involves looping through all $x_c \in L, \theta \in \Theta^{(0)}$, counting how many indices satisfy $X_i(x_c + \theta_i) = 1$, and recording those with a count greater than τ. This is clearly a massive loop over all locations in the image, and for each location a large number of instantiations at that location. An alternative is the Hough transform, (Hough (1962) and Grimson (1990),) which consists of a reordering of the loops.

Create an array H indexed by all elements in Θ. Find all instances $x_{i,j}, j = 1, \dots, J_i$ of each of the local features $X_i, i = 1, \dots, n$ in the image.

1. For each $i = 1, \dots, n$,

   For each $j = 1, \dots, J_i$, loop through elements Θ. For each θ such that $x_{i,j} = \theta_i$ add 1 to the entry corresponding to θ in H.

2. Find all entries in H with value greater than τ.

If the local features are of low density in generic images, it is computationally more efficient to index into admissible configurations through the locations of the features, rather than carrying out the brute force loop. This algorithm is feasible provided there is an efficient way to loop over elements in Θ, or some analytic indexing mechanism to find those elements in Θ for which $x_{i,j} = \theta_i$.

A more efficient way to solve this detection problem is to pursue a coarse to fine procedure. Find a product set $\Theta_p^{(0)} \subset L^n$ which contains $\Theta^{(0)}$. By product set we mean a set of the form $\{\theta : \theta_i \in B_i,\ i = 1, \dots, n\}$ where $B_i \subset L$. If for example $\Theta^{(0)}$ is defined through a set of transformations Υ then take $\Theta_p^{(0)} = \prod_{i=1}^n B_i$ with

$$B_i = \Upsilon z_i = \{\upsilon z_i, \upsilon \in \Upsilon\},\ i = 1, \dots, n. \tag{8.1}$$

With the larger set of instantiations $\Theta_p^{(0)}$ detect *candidate centers* namely points x_c satisfying

(DP) There exists some $\theta \in \Theta_p^{(0)}$ and τ indices $i_1, \dots, i_\tau$ for which $X_{i_j}(x_c + \theta_{i_j}) = 1$.

Any location x_c for which there exists $\theta \in \Theta^{(0)}$ such that the pair (x_c, θ) satisfies condition (D) above, will *necessarily* be identified as a candidate center satisfying condition (DP); simply because $\Theta^{(0)} \subset \Theta_p^{(0)}$. In all experiments below we use $\Theta^{(0)}$ defined in equation 6.1, with $\mathcal{A} = \Upsilon$ some subset of linear maps covering a range of scales of $\pm 25\%$. Thus $B_i = \mathcal{A} z_i$.

In a brute force search for candidate centers each location $x \in L$ would be visited, counting how many of the regions $x + B_i$ contain an instance of the corresponding feature X_i, i.e. $\max_{y \in x + B_i} X_i(x) = 1$. In some cases, one can search over a coarse subgrid of the image grid L and the brute force method becomes efficient. See for example the work in Fleuret & Geman (2001) and Fleuret (2000). In our context it will be more efficient to reverse the loops and use the Hough transform, which in this case is easy to implement, since $\Theta_p^{(0)}$ is a product set.

Algorithm 8.1: Sparse model: counting detector, step I

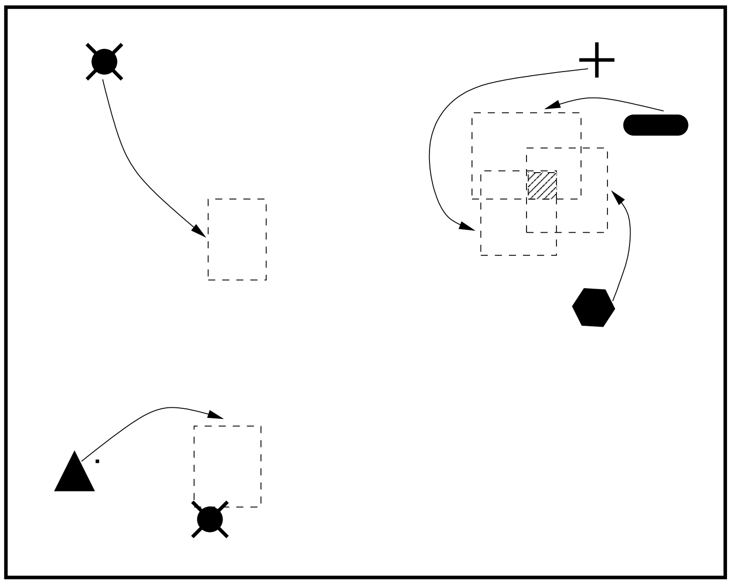

Figure 8.2: An illustration of the voting procedure of the Hough transform.

```
1. For $i = 1, \ldots, n$ find all instances of $X_i$, at $x_{i,j}, j = 1, \ldots, J_i$.
2. Initialize to 0 an array $C$ the size of the image.
3. For $i = 1, \ldots, n$, do
    (a) Initialize to 0 an array $D$ the size of the image.
    (b) For $j = 1, \ldots, J_i$
        Set $D(y) = 1$ for all $y \in x_{i,j} - B_i$.
    (c) Set $C(x) = C(x) + D(x), x \in L$.
4. Find those locations $x$ in $C$ for which $C(x) \geq \tau$.
```

The idea behind this loop is illustrated in figure 8.2 in terms of the coarse model shown in figure 8.1. Each instance of feature X_i detected at location x, 'votes' for a region $x - B_i$ of candidate centers. Here $n = 5$ and $\tau = 3$. In the upper left hand corner a location has received three votes and becomes a candidate center.

The calculation done in item 1 is efficient and only requires lookups in small subregions of the lattice determined by the subregions defining the local features. The calculations in item 2 are very efficient because the loops are only over locations of local features, which are rare, and over the regions $x_{i,j} - B_i$. The size of these regions depends on the distance of the point z_i from the origin and the size of the set $\mathcal{A}$. As indicated we restrict the range of scales covered by $\mathcal{A}$ to $\pm 25\%$ and ± 15 degrees rotation. The size of the reference grid is approximately 30×30, so that typically the regions B_i are on the order of several tens to one hundred pixels.

In figure 8.3 we show four of the 20 stages of the implementation of the Hough transform for step I of the counting detector, with the face model shown in figure 6.9. In the first two stages we show an image with the locations of the corresponding local features (denoted as +') and the distribution of the counts

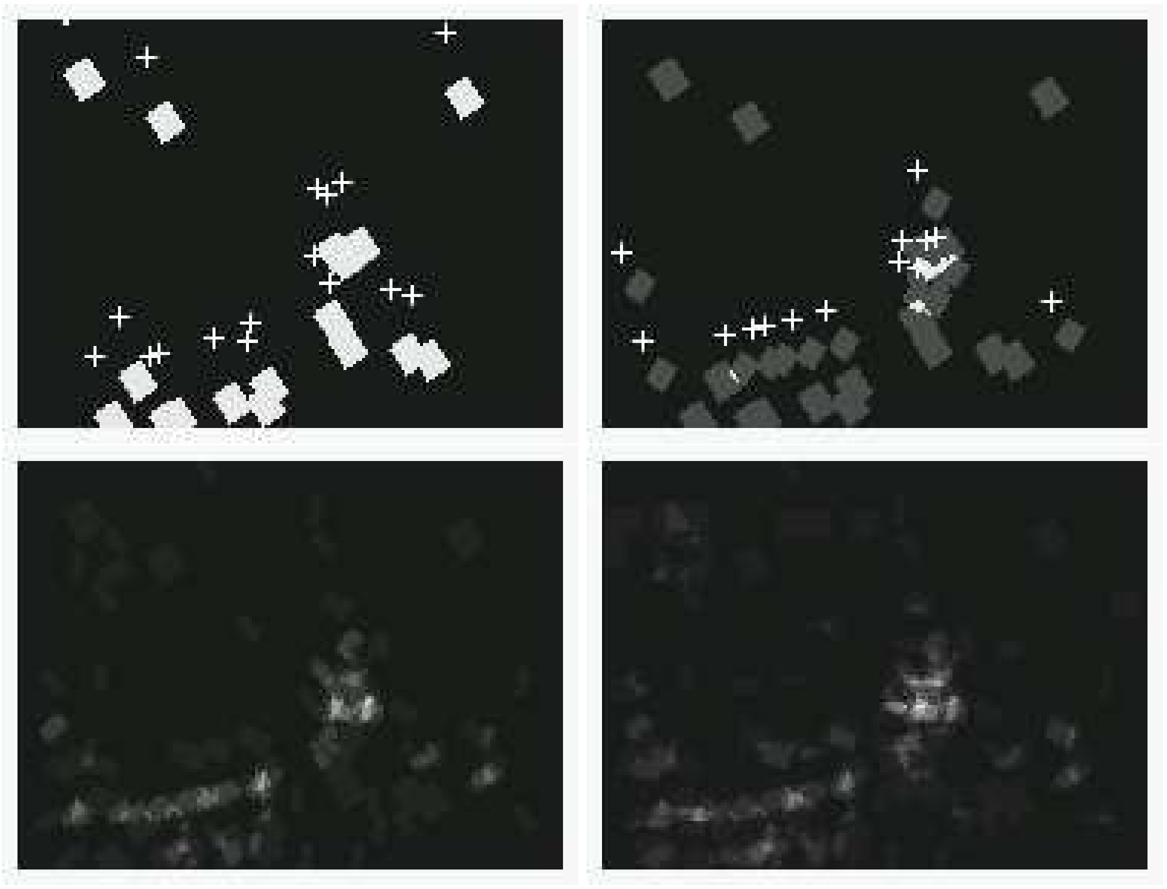

Figure 8.3: Four stages of the Hough transform. Top: First two local features. Locations marked as '+' and count maps (0-black, 1-gray, 2- white). Bottom: Count maps after detection of two other local features later on in the process. Candidate centers for faces emerge as bright regions.

recorded in the array C used in item 3c above. For the other two stages we show only the distribution of counts. At each of these stages it is possible to make out the shape of the corresponding region B_i. Note how the distribution of counts gets more and more peaked at the face. The image being processed here is shown, with the results of the entire counting algorithm, in figure 8.5 on the right.

8.2 Computing pose and instantiation parameters

At each candidate center, x_c, identified by first step of the counting detector we need to recover information regarding the instantiation. First we estimate pose parameters such as scale and rotation. The most straight forward method, used in the examples below, involves brute force search through the range of allowable linear maps. Add a small range of translation to the set of allowable linear maps $\mathcal{A}$. Let C be a small 3×3 or 5×5 neighborhood of the origin. Let $\tau_a = \tau$, the threshold used in condition (D).

Algorithm 8.2: Sparse model: counting algorithm, step II

1. At candidate center x_c, for each $\mathbf{a} \in \mathcal{A}$.
2. Loop over $i = 1, \ldots, n$. If there is an instance of X_i at some point x in the region $x_c + \mathbf{a}z_i + C_i$, add one to a counter $N_{\mathbf{a}}$. Also record the location of x in a list, $\mathbf{X}_{\mathbf{a}}(i) = x$.
3. Let $\mathbf{a}^* = argmax_{\mathbf{a} \in \mathcal{A}} N_{\mathbf{a}}$, be the map with highest count. If $N_{\mathbf{a}^*} < \tau_a$ the location x_c is discarded. Otherwise the detection is assigned the affine map $\mathbf{a}$ defined through $\mathbf{a}(z) = x_c + \mathbf{a}^* z$.

Steps I and II together represent the full counting algorithm. The main computational cost is in implementing step I since it involves image wide detection of the local features. Among the few locations which it detects it is very fast to implement step II to obtain the instantiation.

Edge based model for step II

Step II is described in terms of the features used in the original object model. It yields locations satisfying condition (D). But in practice the features used for this step do not have to be the same features used for the first step. One of the important properties of the more complex features, employed in the initial detection step, is their low density in the background. This is essential for computational efficiency and to keep a tab on the number of false positives. However once a location x_c is selected these issues are irrelevant. Indeed in step II, one can use the simpler edge features which have already been extracted in the process of finding the edge arrangements. These are stored and used for step II.

Training of the edge based model used for this step, would proceed as described in section 6.4. There are much larger numbers of high probability edges on the reference grid, than edge arrangements.

Determining thresholds

Now there are two thresholds to determine, τ for step I, and τ_a for step II. We thus use the following modification of the threshold training procedure given in section 6.4.

- Find the largest value of τ for which at least $(1 - r)T$ of the training images has a candidate center, resulting from step I, within D pixels of $x_{c,t}$ - the mean of the three anchor points for image t.
- Find the largest value of τ_a for which at least $(1-r)T$ of the training images, has at least one candidate center resulting from steps I (using τ determined above) and II, within D pixels of $x_{c,t}$.

From pose to instantiation

The outcome of this stage is a list of affine maps $\mathbf{a}_k, k = 1, \ldots, K$ of the reference grid into the image, corresponding to detected instantiations of the object. These are represented through a *detection triangle* $\mathbf{a}_k z_1, \mathbf{a}_k z_2, \mathbf{a}_k z_3$. In addition, for each such map, there is a list $X_{\mathbf{a}_k}$ of the locations of those model points which were detected in the correct regions: $\mathbf{a}_k z_i + C$. For those model points z_i which were not detected, it is possible to 'extrapolate' their location by taking $X_{\mathbf{a}_k}(i) = \mathbf{a}_k z_i$. In this way we recover an entire instantiation element $\theta \in \Theta$ for each detection.. Note that this instantiation is not simply given by $\mathbf{a}_k z_1, \ldots, \mathbf{a}_k z_n$. For those features that were found in step II, their original location in the region $\mathbf{a}_k z_i + C$ is recorded. The larger the neighborhood C, the more flexibility there is in terms of the final instantiation, at the price of more false positives.

Clustering

The local features occur in clusters. Since they are only used for the crude initial detection of candidate locations they can be clustered without significant loss of information. Typically we find all features of the same type in disjoint 5×5 blocks and replace them by one such feature at the mean location.

Further efficiency can be gained if instead of recording the local features on the original lattice and then clustering they are directly recorded on a coarse sub-lattice and the Hough transform is performed using a coarse version of the B_i sets on the coarser lattice. Candidate centers are then properly injected into the original lattice where the full instantiation information is then recovered in step II.

Detected instantiations will also occur in clusters. Given that only one instance of the object is present in a certain region it is necessary to pick one detection from the cluster. This can be done in a variety of ways. In the examples below the detections are clustered according to the distance between the detection triangles. In each cluster the map $\mathbf{a}$ with highest count $N_{\mathbf{a}}$ of detected features, is chosen.

Invariance

The detection scheme described here covers the range of poses determined by the set $\mathcal{A}$ as well as a range of non-linear deformations which is difficult to quantify. It is also quite robust to non-smooth variations such as occlusion of part of the object and changes in internal structures. The examples below will illustrate these points. The invariance to the range of poses in $\mathcal{A}$ is explicitly obtained through the definition of the sets B_i defined in equation 8.1, as well as in the implementation of Step II. Invariance to other deformations and degradations comes from two sources. The first is again the large degree of slack introduced in the counting detector. The decoupling of the constraints resulting from the use of the product set $\Theta_p^{(0)}$ allows for rather large non-linear deformations. In addition, the fact that in both Step I and II a detection survives if the number of counts is above some threshold, allows for certain parts of the object to change quite drastically or be entirely occluded.

8.3 Density of candidate centers and false positives.

In section 6.5 we studied the false positive density of the sparse models, disregarding the means by which these models are detected. Some additional statistics are relevant at this point which bear upon the computation time and shed some more light on the false positive density. Recall that originally the model simply involved a configuration of local features of predetermined complexity n_r. However now we have split the detection into two steps. The first detects a coarser model and the second recovers the pose and instantiation parameters using the basic edges.

The density of the candidate centers detected by step I will now be higher than the density of false positives presented in table 6.2, see for example column 4 of table 8.1. In the former an instance of the feature could be anywhere in the 5×5 neighborhood of the expected feature location $\mathbf{a}z_i$, just to accommodate small local variations. However now an instance can be anywhere in the region B_{z_i} which is much larger and may contain close to 100 pixels. Still for the various values of n_r the number of candidate centers in a 240×320 image varies between several tens to several hundreds. As long as the number is on the order of tens, subsequent computations at these candidate centers do not amount to a large fraction of the computing time. However as this number grows, Step II starts dominating in terms of computation time. It is then necessary to devise more efficient algorithms for Step II. We emphasize that the statistics shown here are obtained from outdoor images which have a very large density of edges. On indoor scenes the numbers are much lower.

In this context we see why a model solely based on edge features would be problematic. Edge densities range between .03 to .06 edges per pixel, for each edge type. Even for a restricted set of admissible transformations $\mathcal{A}$ the minimal size of the B neighborhoods would be on the order of 30-40 pixels. But then the

n_r	τ 5% Fn.	Cand. Centers	τ_a 5% Fn.	False Pos. I+II
1	29	$9 \cdot 10^{-4}$	58	$8 \cdot 10^{-5}$
2	25	$2 \cdot 10^{-4}$	58	$3 \cdot 10^{-5}$
3	21	$5 \cdot 10^{-5}$	53	$3 \cdot 10^{-5}$
4	15	$8 \cdot 10^{-5}$	58	$2 \cdot 10^{-5}$
5	8	$2 \cdot 10^{-4}$	58	$5 \cdot 10^{-5}$
6	5	$3 \cdot 10^{-4}$	57	$7 \cdot 10^{-5}$

Table 8.1: τ and τ_a, candidate location and false positive rates for false negative rates under 5%.

expected number of edges of any one type in any of the regions becomes very close to 1 if not greater. This implies that the number of false positives is on the order of the number of pixels in the image and nothing has been gained in this step.

Taking into account step II of the algorithm which is based on edge features the density of false positive turns out to be even lower than that recorded in table 6.2 in section 6.5. This is shown in table 8.1. The thresholds τ and τ_a were estimated with the procedure outlined above for each value of n_r. The values of τ are larger here than in table 6.2 because the constraints in step I are now defined in terms of $\Theta_p^{(0)}$ which is much larger than $\Theta^{(0)}$. These thresholds are determined with a small set of 100 faces which were not used for creating the model. For different values of n_r the final false positive rates are quite similar due to the fact that the same edge based model is used in step II. However the density of candidate centers detected in step I varies quite significantly. This has an important affect on the the final computation time, in terms of the time required to compute step II.

8.4 Further analysis of a detection

After an instantiation θ is detected a map from the reference grid into the image can be defined. This map could be the affine map $\mathbf{a}$ or a more complex map obtained through some form of interpolation of the correspondence between z_i and θ_i for $i = 1, \ldots, n$. The best known method involves thin plate splines, see Bookstein (1991), a much more computationally intensive alternative is proposed in Joshi (1997) where the interpolated map is guaranteed to be a homeomorphism, an more efficient computation of such homeomorphisms can be found in Camion & Younes (2001). In the examples below we restrict ourselves to the affine map. The region of interest (ROI) $\mathbf{a}G$, is obtained by applying the map $\mathbf{a}$ to each point of the reference grid. The data $\hat{I}(x), x \in \mathbf{a}G$ can be registered back to the reference grid to produce,

$$\hat{I}_r(z) = \hat{I}(\mathbf{a}z), z \in G, \tag{8.2}$$

and can then be further analyzed using any of the deformable models described in earlier chapters. These can serve not only to provide a more detailed instantiation but also to filter out false detections, by keeping only those detections which yield a fit of the template below some cost level, determined using training data.

Detection as classification

The detection scheme provides a classification at each possible instantiation θ, between the two classes 'object' and 'non-object'. Note that in the first two steps of the algorithm there was hardly any use of 'non-object' or background images, in terms of training. The reason is that the background image population is so large and diverse that a very large sample would be needed to yield reliable estimates for a global classifier

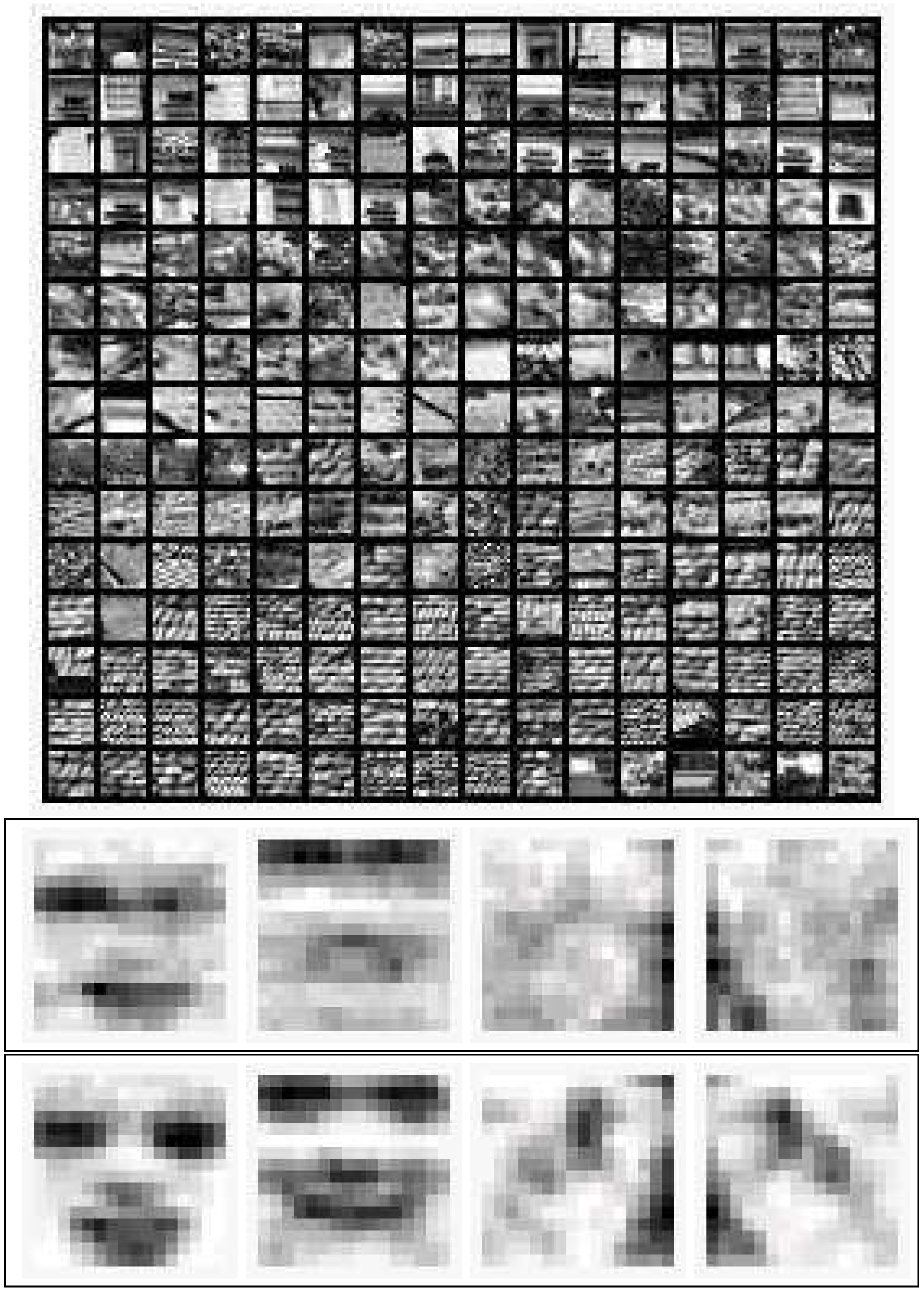

Figure 8.4: Top: Registered ROI's of false positives from the face detector. Middle: Edge maps for this population. Bottom: Edge maps for face population. Reference grid is reduced to a 16×16 lattice covering the immediate region around the eyes and mouth.

between object and non-object images. Large data sets of many thousands of images have been used for this purpose in Rowley et al. (1998), Sung & Poggio (1998). The sparse object models described here can be trained successfully with several tens of examples. The only information regarding background in the counting algorithm involves the densities of the local features, studied in Chapter 6. Only a crude estimate of these is needed simply to ensure the feasibility of the first step of the algorithm. However once a detector is produced, the population of *false positives* is much more homogeneous and constrained.

Using the original training sample of the object, together with a sample of false positives, a classifier can be trained. Run the detector on all the images of the object in the training set, and register the pixel intensities, or the edge data, or the edge arrangement data, from the ROI defined by the detection, to the reference grid, as specified in equation 8.2. The same is done on images containing no object, each of which may yield a number of registered detections, to produce a population of false positives. This data is then used to train classifiers of the type described in Chapter 9. In figure 8.4 we show a sample of ROI's of false positives for the face detector. The reference grid has been reduced to a 16×16 lattice immediately surrounding the area around the eyes and mouth. Below in the middle panel we show the histogram maps of four edge types (two vertical and two horizontal) on this population, compared to the maps for the face population shown in the bottom panel. The similarity is quite striking, indicating that this false positive population is quite restricted. The edge histograms for randomly selected 16×16 images in the background population are *uniform.* And still for the human eye there is no question that most of the false positives are *not* faces.

8.5 Examples

In this section we illustrate some aspects of the counting detector for a variety of of objects: faces, randomly perturbed LaTeX symbols in cluttered scenes, the axial MRI scans and an example of a 3d object at a limited range of viewpoints. In addition to detections of the sparse model we show examples of how these detections can provide automatic initialization for the deformable models described in Chapters 3, 4, and 5. Specifically we will use the maps **a** estimated in Step II, to initialize the templates. Recall that the sparse model represents the smallest range of scales at which the object is detected, and the algorithm is applied at 6 different resolutions to cover a range of scales of 4:1.

The edge arrangements in the examples are defined with the 16 smaller wedges shown in the top row of image 6.7. The complexity of the arrangements is 3, i.e. $n_r = 3$, and apart from the MRI example we always pick $n = 20$ arrangements for the model with lower bound $\rho = .5$ (see algorithm 6.4).

Faces

The face model shown in figure 6.9 was trained from 300 face images of the Olivetti data set. Much smaller datasets yield very similar results. The original images were downscaled to 44×36. The centers of the two eyes and the mouth were manually marked on each image and used as the three anchor points. The mean locations of each of these three landmarks are used for the three reference points p_1, p_2, p_3 on the reference grid, as defined in section 6.4. At this scale the mean distance between the center of the eyes is 14 pixels and the distance between the center of the mouth and the middle point between the two eyes is also 14 pixels. Edges are extracted on each downscaled image, after which their locations are registered to the reference grid using the affine map taking the anchor points to the reference points. The edge statistics are graphically represented in figure 6.8, and in terms of these, an edge model for Step II was derived with 110 edge-type/location pairs. The 20 edge arrangements of complexity $n_r = 3$ obtained from training are shown in figure 6.9. A model using $n_r = 1$ is shown in Chapter 11. The range $\mathcal{A}$ of linear maps at which we expect to detect a face at a given resolution covers $\pm 25\%$ scaling and ± 15 degrees of rotation. The smallest scale at which a face is detected is approximately 10 pixels between the two eyes. Faces at much larger scales are detected by sub-sampling the image and rerunning the detection algorithm.

The results of Step I of the detection algorithm using this model, and processing 6 resolutions covering a range of scales of 4:1, are shown in the top row of figures 8.5 and 8.6. In the middle row are shown all detected affine maps obtained from Step II. The affine maps are represented using the detection triangle showing the locations of the two eyes and mouth.

A final classifier was produced for face versus non-face using randomized classification trees (see chapter 9). These trees were trained using the registered edge data of the face training set against registered edge data from false detections obtained on a collection of random images, as shown in figure 8.4. The results are shown on the bottom row of the two figures. Some additional detections are shown in figure 8.7. The training set for this classifier is quite small - 300 face images of 30 dfiferent people (10 per person) and a similar number of false detections. This is therefore the least stable component of the algorithm and increases the number of false negatives.

Deformable models initialized at detections of sparse model

In figure 8.8 we show a close-up on the four faces of the previous figures and the locations of the points in the instantiation obtained from step II. The white dots show edge features detected at the estimated pose and the black dots show the extrapolated features which were not detected. If the features in the model are labeled according to the components of the face to which they belong, we obtain an estimate not only of where the centers of the two eyes and mouth are but other parts of the face as well. For example the location of other parts of the eyes, or part of the nose, the hairline etc.

One can also initialize a deformable image algorithm at the detected pose . Due to the high variability in lighting and illumination of faces we implement the Bernoulli data model from section 5.4. This data model, being based on probabilities of edges which are very robust to illumination changes, inherits some of these properties. On the left in figure 8.9 we show the location of a set of points chosen on the reference grid mapped into the image by the detected pose. In the middle we show the outcome of a global search on a range of scale and location parameters for updating the pose. On the right is the outcome of the deformable image algorithm using a small number of basis coefficients. Note how the outlines of the eyes have been adjusted as well as the hairline which in the initial instantiation was outside the face. The first example of this procedure was shown in figure 5.4 in Chapter 5, together with the edge maps and the edge data used to drive the algorithm.

Randomized LaTeX symbols

Displays with randomized deformations of LaTeX symbols are useful for investigating properties of the sparse models. The background has objects with many similarities to the one being detected, making the problem of particular interest. We use 32 randomly perturbed versions of the prototype as shown in figure 8.10 for training. The random perturbations involve random rotation of up to 15 degrees, and $\pm 25\%$ scaling independently in each coordinate. The 20 local features identified in the training step are given in the bottom panel of figure 8.10. The reference grid is approximately 30×30.

Here it is possible to produce a detector from the prototype alone, by simply picking some existing arrangement of edges extracted from the prototype, in each region of the reference grid. Such a procedure may pick an edge or a particular arrangement that is very unstable and only present in the particular image of the prototype. This is avoided by producing a small randomly perturbed training set as above, in which case unstable arrangements are eliminated.

Some detections in randomly generated scenes containing a large number of randomly deformed LaTeX symbols are shown in figure 8.11. Again these are obtained from six different resolutions. There are at most

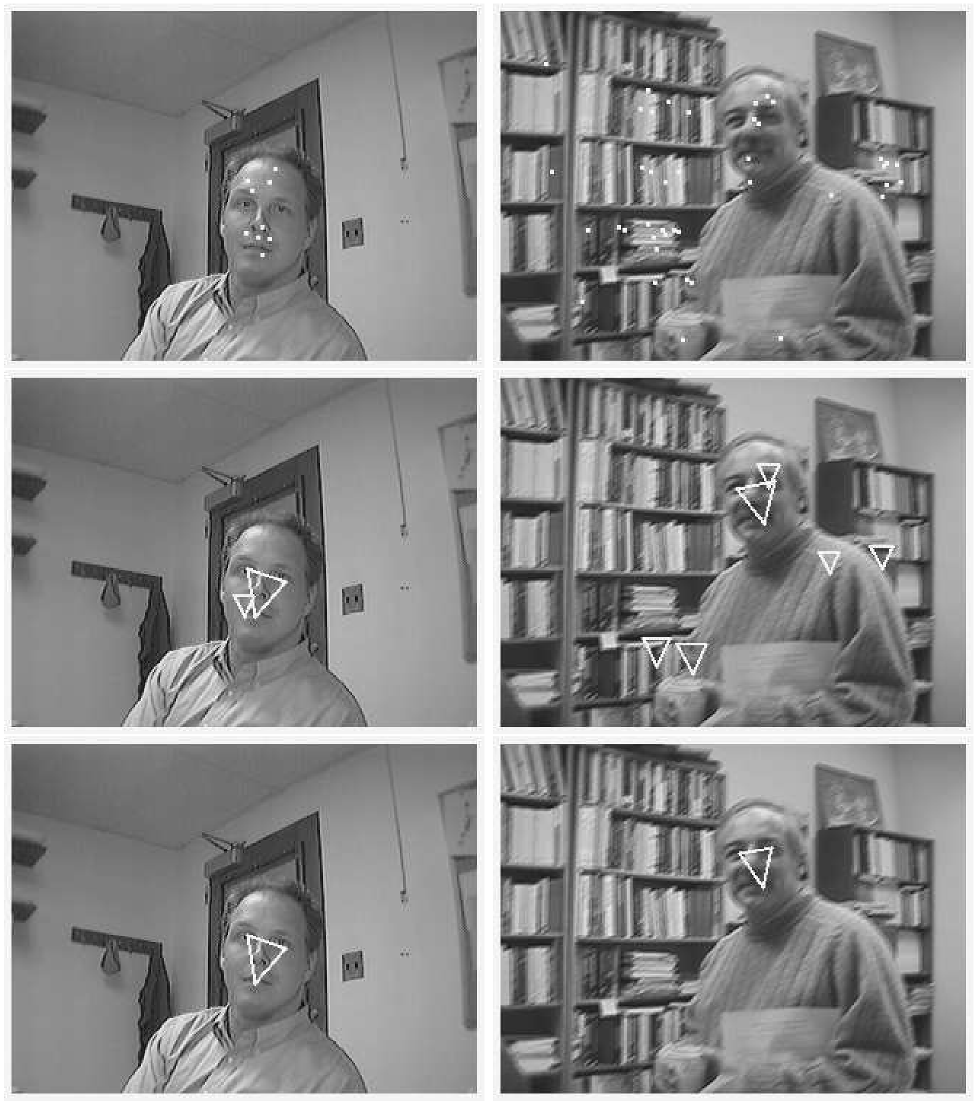

Figure 8.5: Top: Detections from step I. Middle: Detections from step II. Bottom: Detections remaining after final classifier. Detection triangles - mapping of the reference points into the image by **a**.

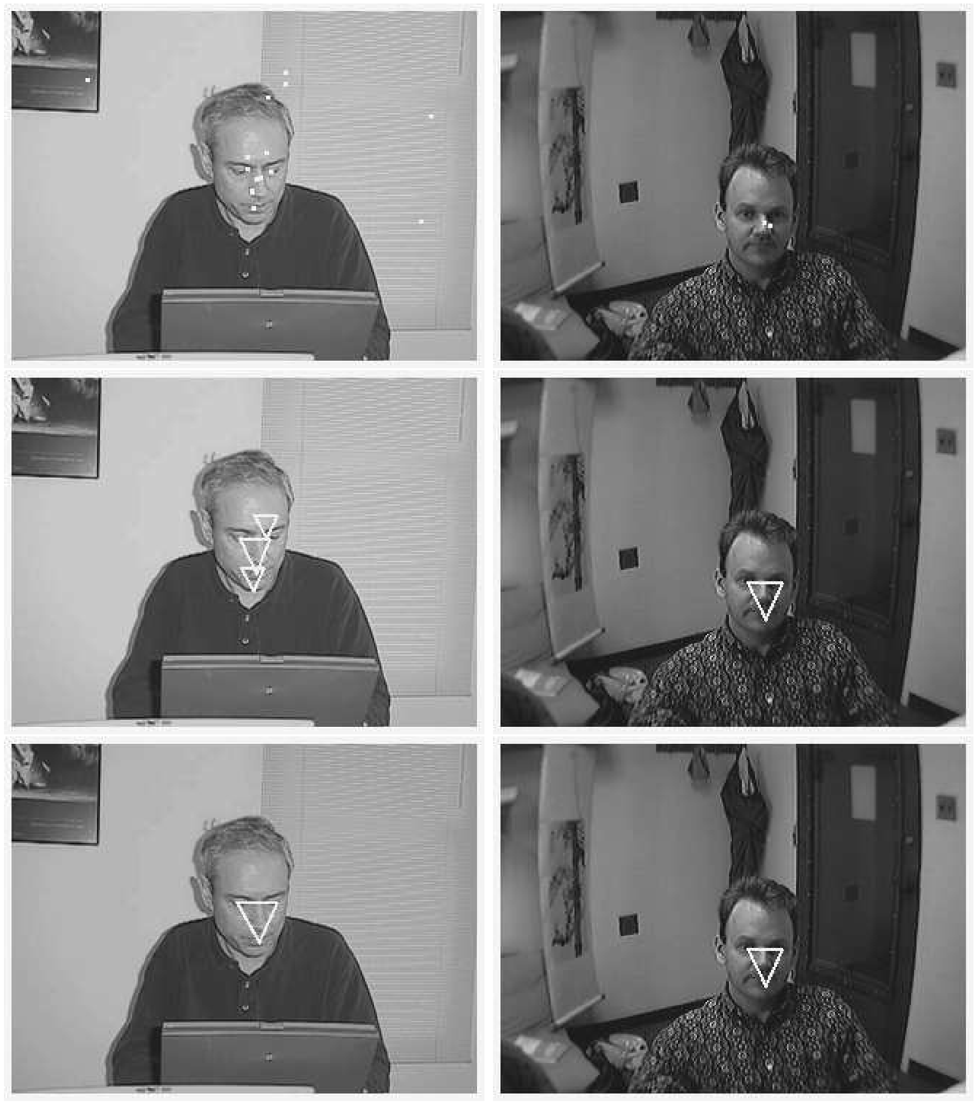

Figure 8.6: Top: Detections from step I. Middle: Detections from step II. Bottom: Detections remaining after final classifier. Detection triangles - mapping of the reference points into the image by **a**.

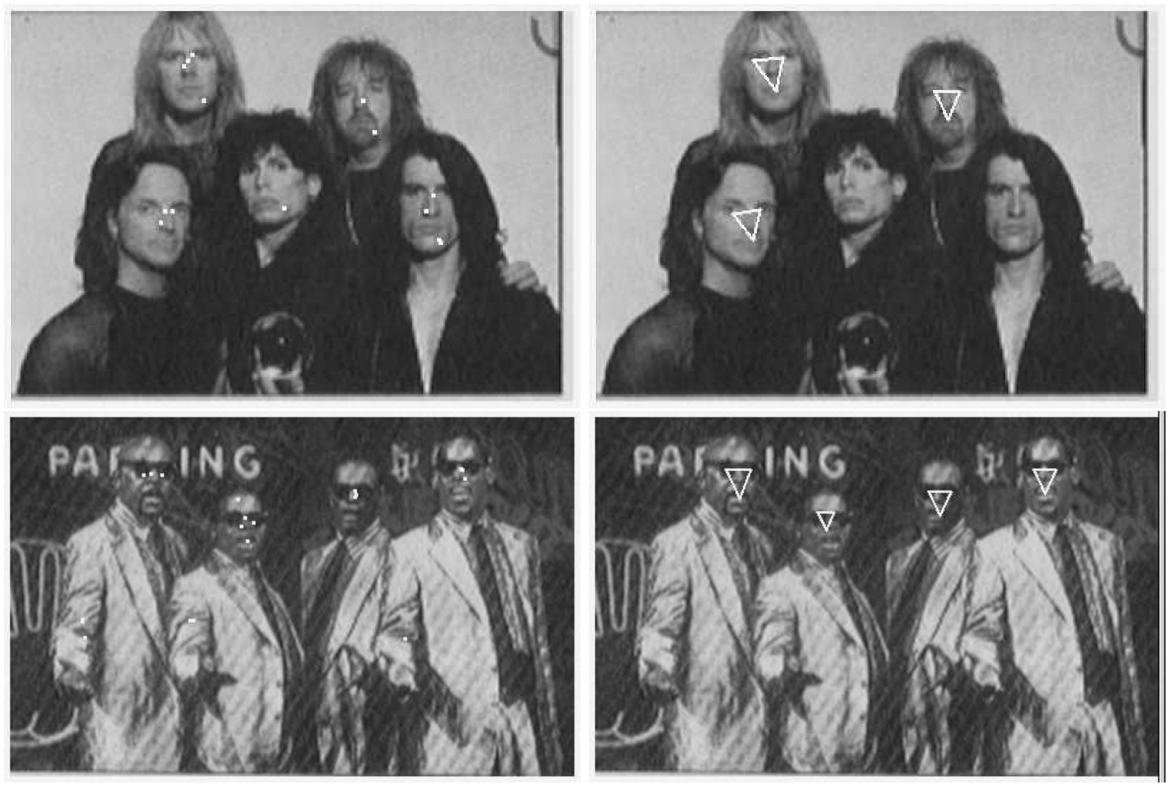

Figure 8.7: Some additional examples of face detections: steps I and II

three instances of an $\mathcal{E}$ in each image. The top panels show the outcome of steps I and II, and in the bottom panels the outcome of running the classifier of object versus false positives, based on registered edges. In the bottom panels the triangle vertices are shown, to enhance the view of the symbols underlying symbols. The use of a classifier among different object classes, detected by the same model, is discussed in Chapter 10. In the top row of figure 8.12 is an example of a deformable contour initialized with the scale and location of the detection, then the final detection of the deformable contour algorithm. In the bottom row we show a similar experiment with the deformable curve algorithm.

Very similar results are obtained for any of the symbols in the data base. However certain symbols are more 'generic' in their shape and share many features with a substantial number of other symbols. The best example being the digit 0 which has much in common with many round symbols. The representations trained for these symbols usually produce more false positives. Although a drawback from the point of view of accurate detection we turn this vice into a virtue in Chapter 10 where we discuss ways to integrate detection and recognition.

The random LaTeX displays are artificial and do not represent all the complexities of real scenes. However in many respects these scenes force us to deal with some important issues such as clutter, confusing classes, and detection at a variety of poses. The background clutter in these images contains many components and parts of the detected object itself. This reinforces the point that detection can not rely on individual local features, and forces us to think about detecting configurations. The existence of objects very similar to the one we want to detect forces us to deal with recognition as well as detection. It seems there is still much to be explored in this controlled synthetic context.

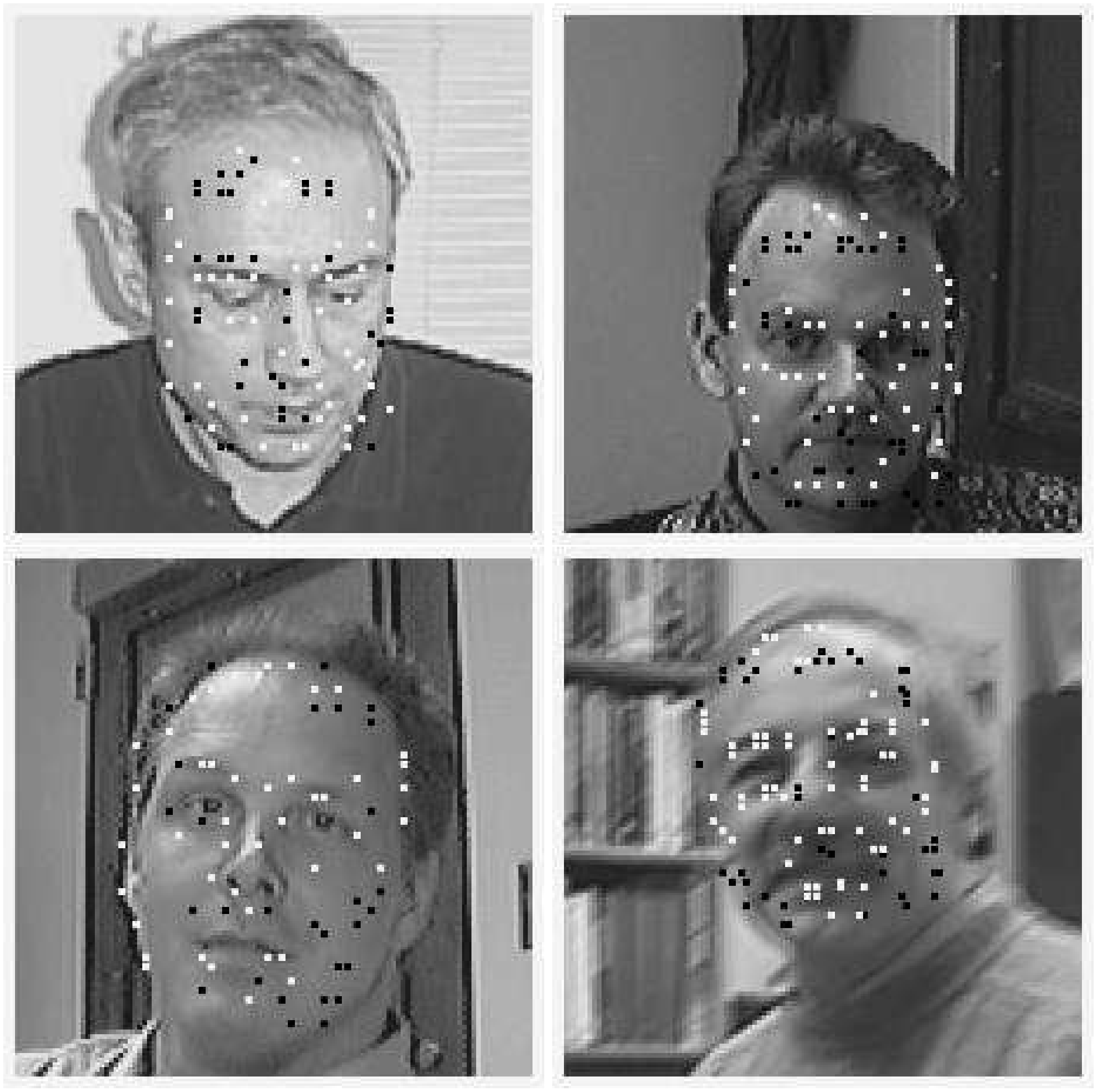

Figure 8.8: Estimated instantiation θ on the four faces detected above. White dots show detected edge features, black dots show extrapolated features.

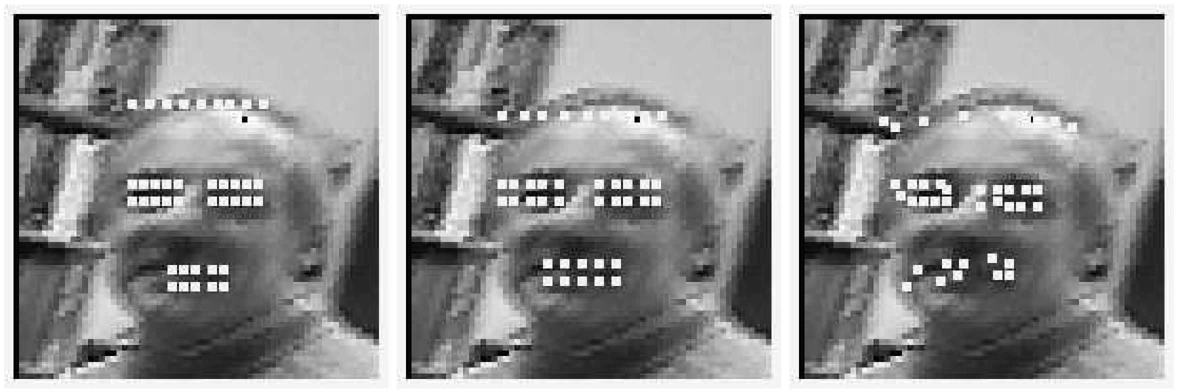

Figure 8.9: Pose detected on right hand image in figure 8.5 initializes Bernoulli model for deformable images.

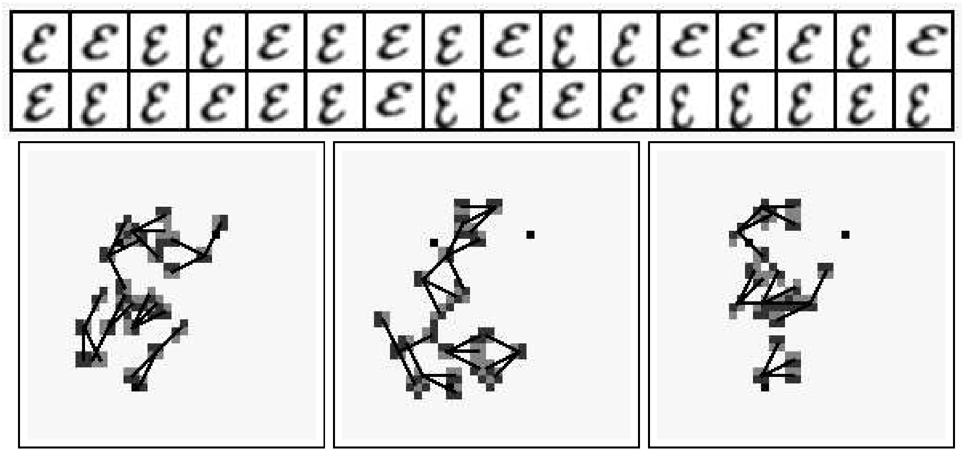

Figure 8.10: Top: 32 randomly perturbed $\mathcal{E}$'s used for training. Bottom: The 20 features of the model represented on the reference grid. The three reference points are shown in black.

3d object - range of viewing angles

Detecting a rigid 3d object from multiple views is a complex problem. Some of the original work in the field attempted to construct 3d models and match them to data. Many of these models are described in Grimson (1990) and Haralick & Shapiro (1992). We take the 'view based' approach in which significantly different views of the object are to be considered as 'different' 2d objects which are linked together symbolically. Much as if in the LaTeX scene we set out to detect a *subset* of symbols as opposed to a particular one. Thus detection involves a sequence of searches for each of these possible objects or views.

Here we focus on developing a sparse model to detect an object from a limited range of views. The problem then reduces to the detection of a *non-rigid* two-dimensional object and can be addressed with the tools we have developed. The object model is trained as the LaTeX symbols using only one prototype image which is deformed 32 times with random affine transformations. A sample of these deformed images is show in figure 8.13, together with the model which is constructed of 20 local features of complexity $n_r = 3$ on a reference grid of approximately 40×40. Note that the local features describe a rectangle of certain aspect ratio to which are attached, on the top and the bottom, two additional wire-like rectangular structures.

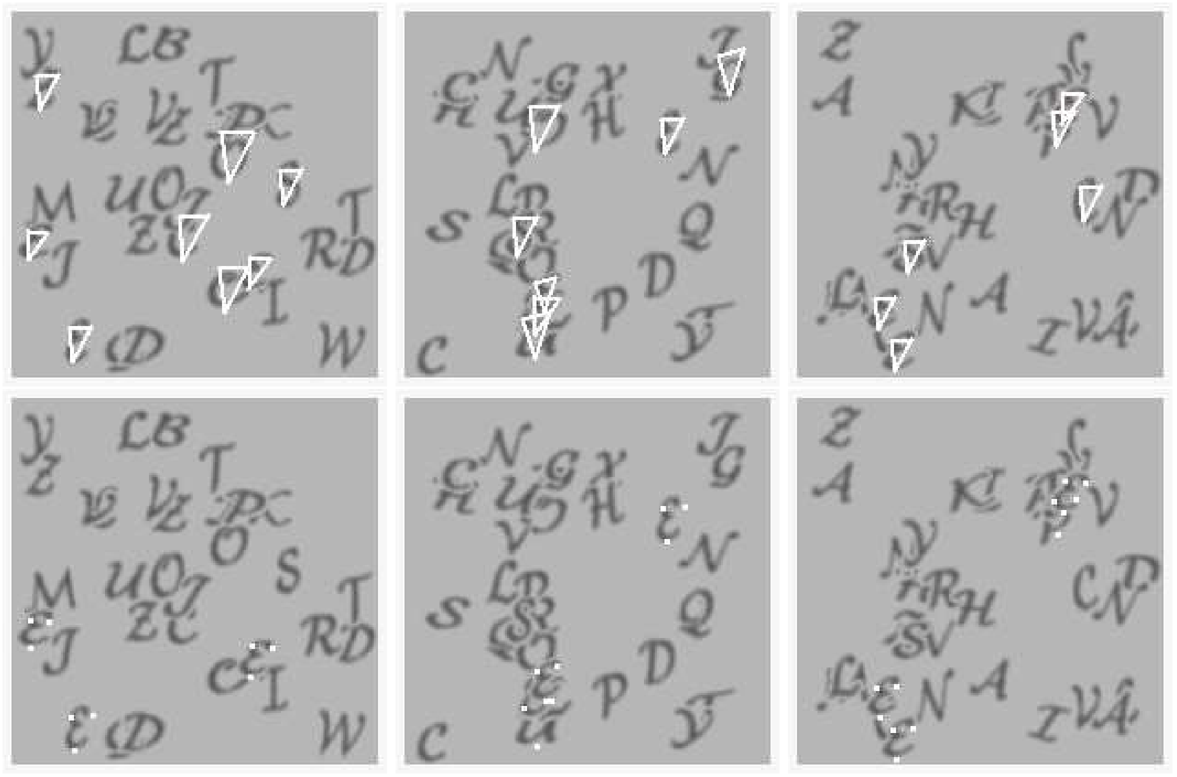

Figure 8.11: Top: Three examples of detections of the $\mathcal{E}$ model, steps I and II. Detection triangles are shown. Bottom: Results after applying the classifier of object vs. false positives. Triangles shown as 3 points for better viewing.

The outcome of the counting detector are shown on images in figure 8.14. No final classifier has been implemented. The algorithm is run at the six resolutions. The detections appear to be invariant to changes in viewing angle around the vertical axis, clutter, occlusion, variable lighting, and articulation of the handles, despite the fact that no such examples were shown in training. Here we see the advantage of treating a rigid object in the same way as deformable objects. In the discussion section and in chapter 10 we offer some preliminary ideas on the idea of detecting 3d objects as unions of their 2d views. How can this be integrated into a more general scheme of object detection and recognition, and how to avoid the combinatoric explosion of representing multiple objects at multiple views.

MRI images

The MRI images are trained on 18 examples of more or less the same axial slice. The reference grid in this case is larger because we have attempted to create a model for the entire structure and reducing to a small 40×40 grid destroys many of the important components. Thus the reference grid is 128×128, and 100 features are identified. These are shown in figure 8.15. In this case the same features are used for step II as well. This experiment demonstrates the applicability of the counting detector for highly deformable objects.

In figure 8.16 we show the outcome of steps I and II in terms of the estimated instantiation. Note how

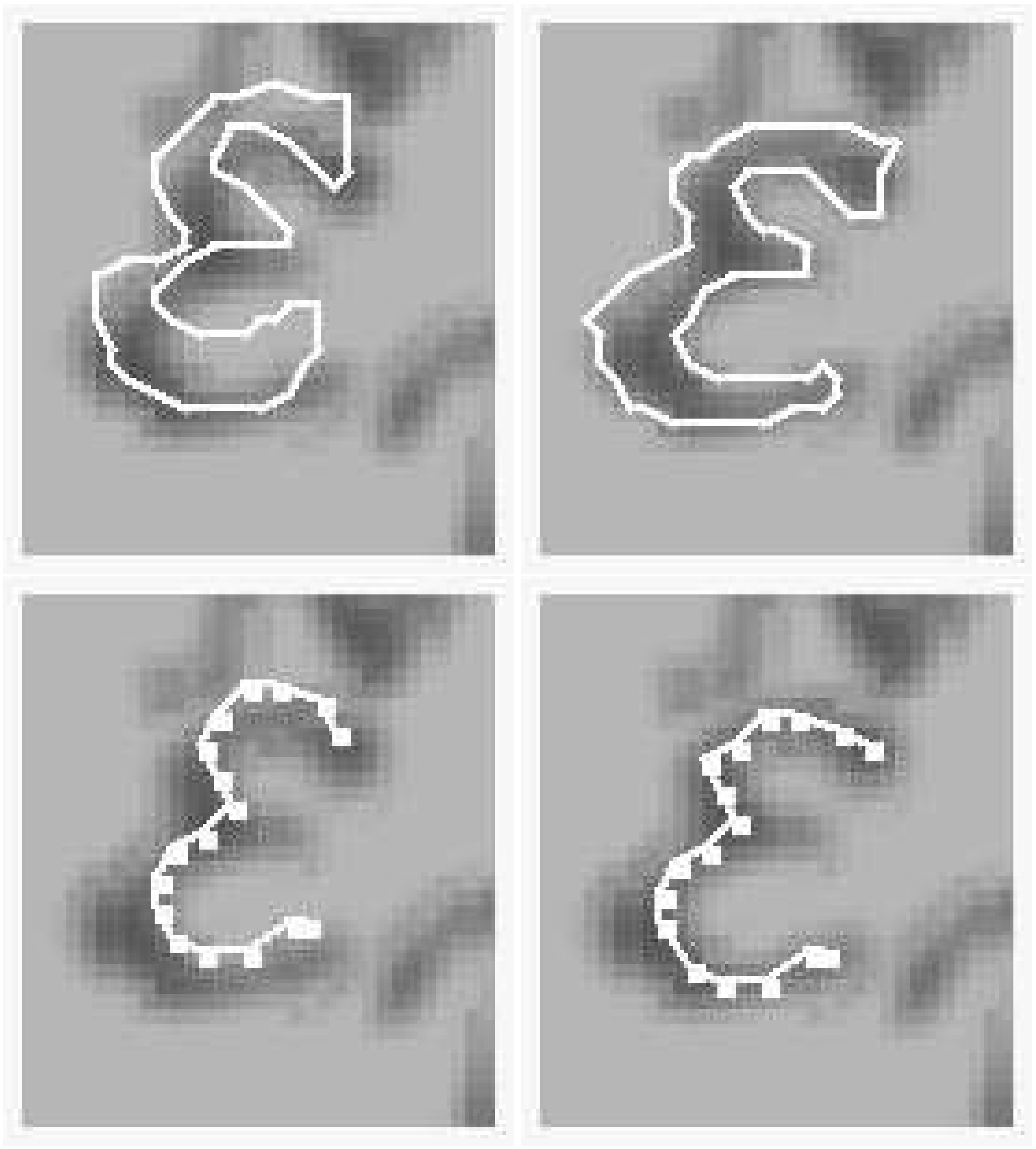

Figure 8.12: Top row: A deformable contour algorithm initialized by a detection in the middle panel of figure 8.11. On the right is the initial contour. On the left the final contour. Bottom row: The result of the deformable curve algorithm initialized the same way. .

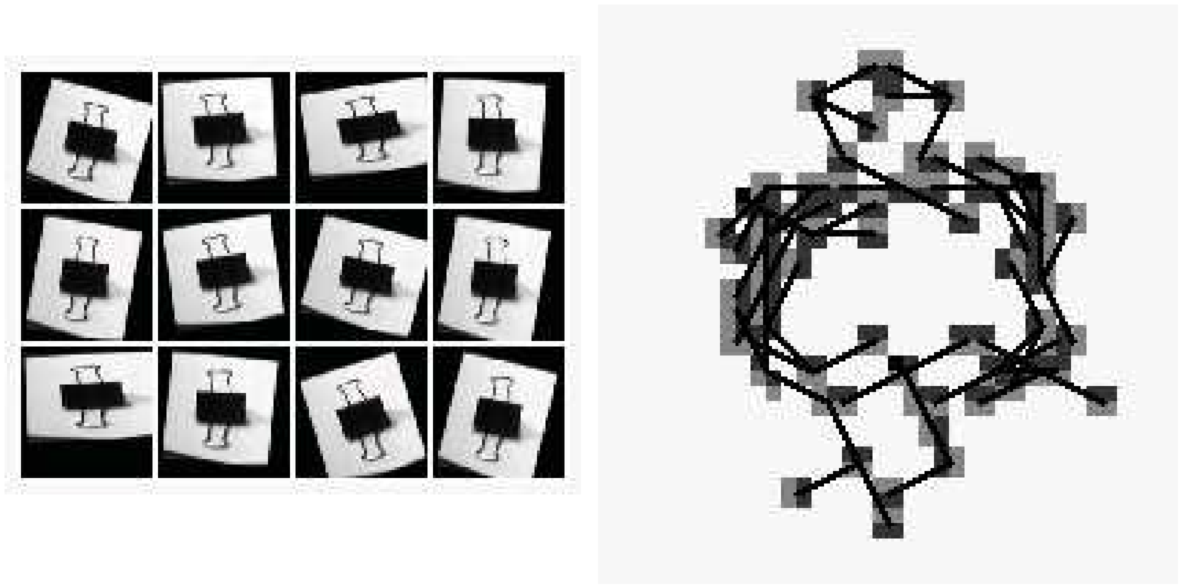

Figure 8.13: A sample of the synthetically generated training images for the clip, and the 20 features identified in training.

the multiplicity of features introduces a large degree of flexibility. Even if parts of the internal structure are missing or significantly deformed the detection succeeds. We show only the close-ups on the ventricle region for those images which actually contained such structures. On those parts which are present or are not severely deformed, the detected instantiation has local features found in the data. For the rest we show the extrapolated location of the undetected edge features. In order to illustrate the correspondence between the detected features a number is attached to each point. [p] Subsequent to detection a deformable contour algorithm is implemented, initialized at the expected location of the anterior ventricles, determined automatically relative to the detected instantiation . The initial contour has a shape similar to the outline of these ventricles, see figure 8.17. Four points have been marked on the contour to show the correspondences derived from the contour match. A more efficient approach, that has not been implemented here, would initialize the elastic contour using the local features detected in step II which are know to lie on the outline of the anterior ventricles. The important fact to remember here is that the entire process is completely automatic, no user initialization is required.

8.6 Bibliographical notes and discussion

The material in this Chapter is based on Amit et al. (1998), Amit & Geman (1999), and Amit (2000) where it is shown how the counting algorithm can be implemented in a neural network architecture. The network can detect any model evoked in memory using a priming mechanism. This architecture is described in Chapter 11.

Object detection with arrangements of local features

The idea of using local feature arrangements for object detection both for faces and other objects can also be found in Burl et al. (1995), Burl et al. (1998), Van Rullen et al. (1998), Wiskott et al. (1997),

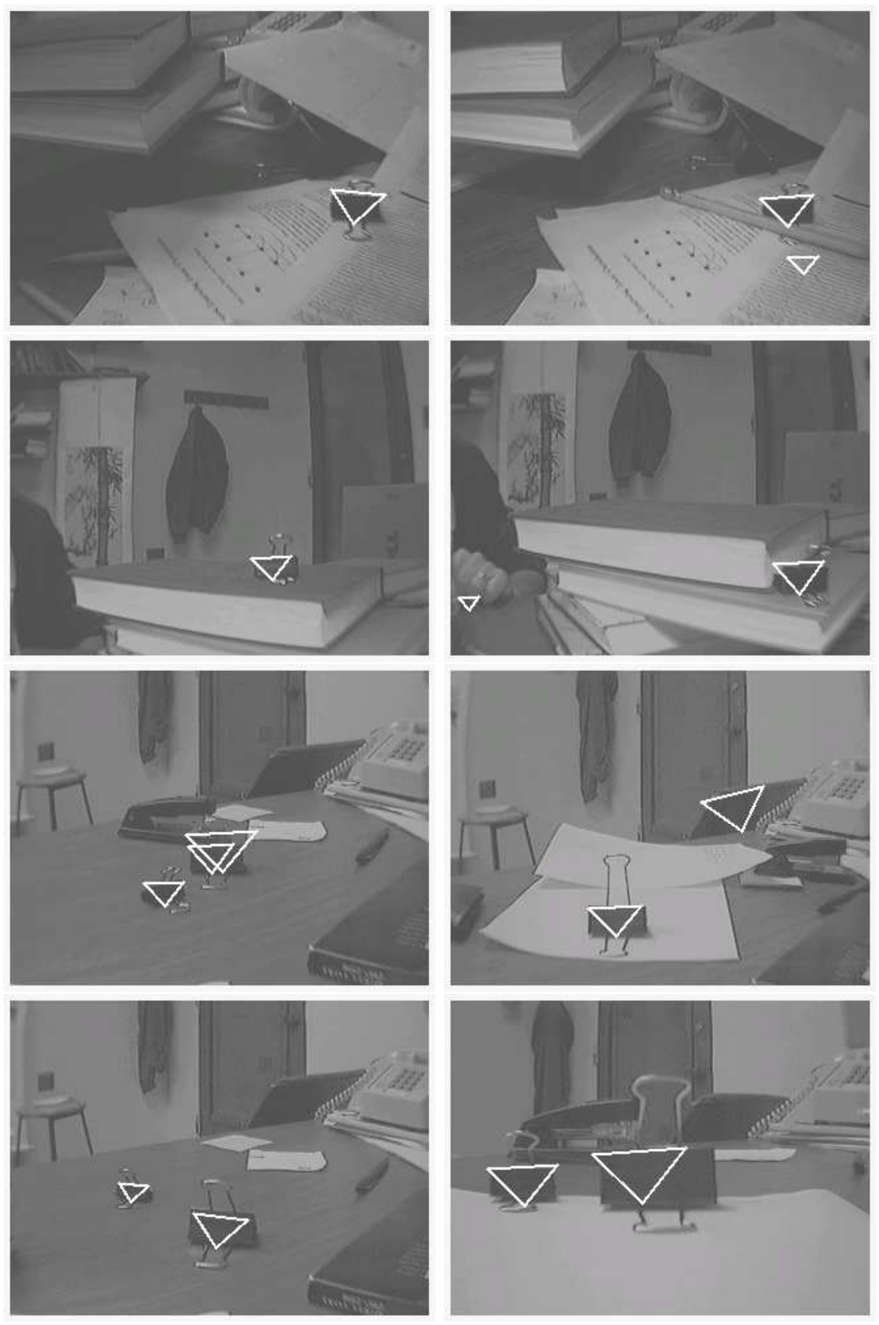
Figure 8.14: Eight examples of detections of steps I and II combined.

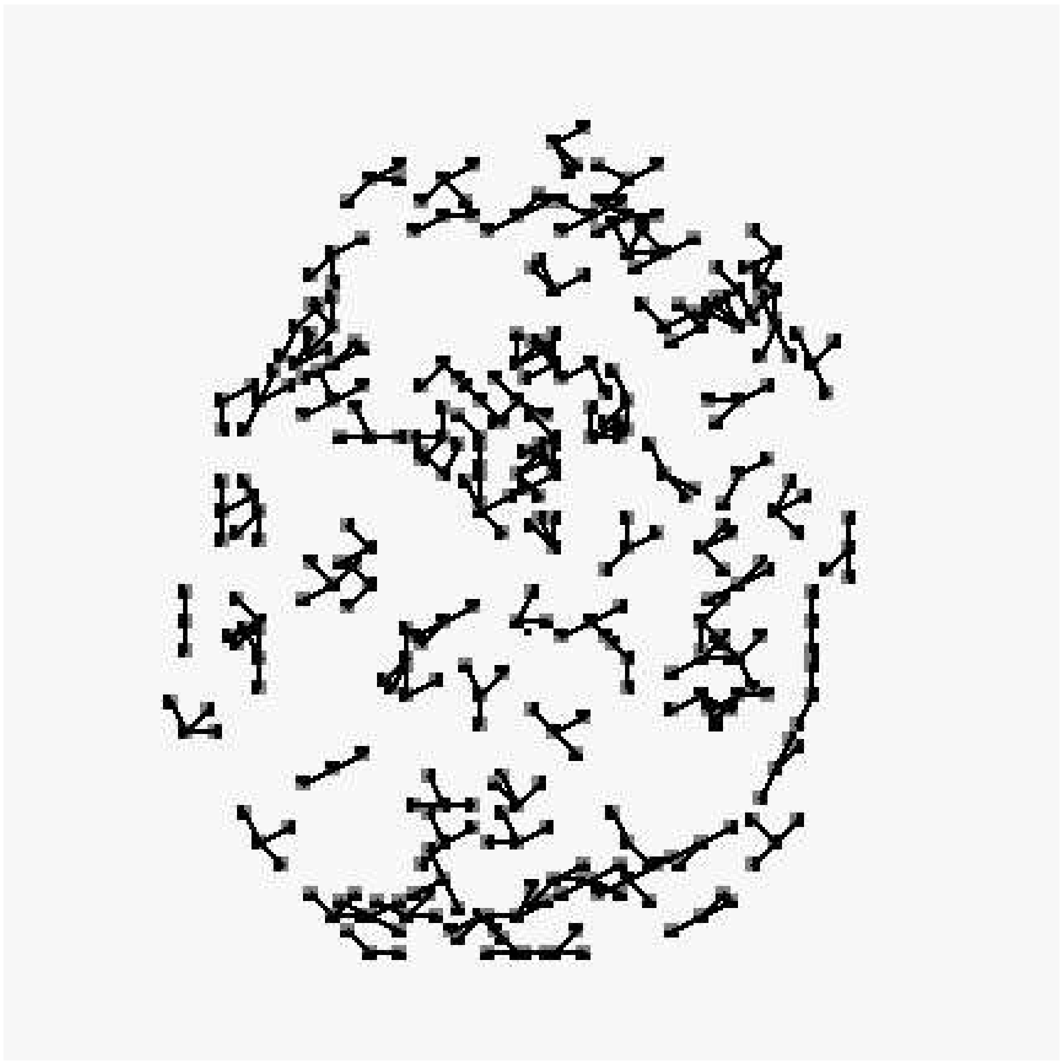
Figure 8.15: The model obtained for axial MRI images.

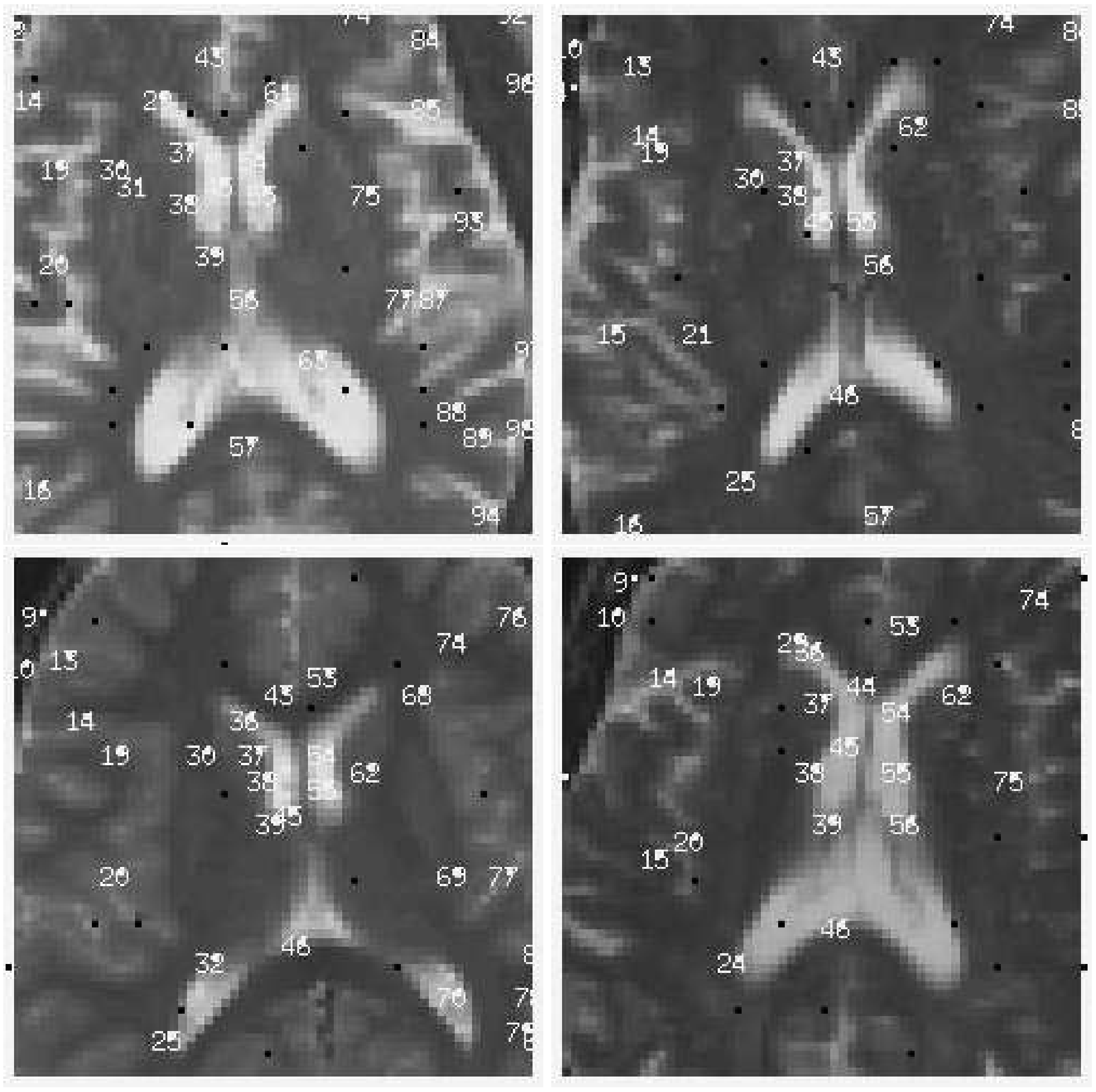

Figure 8.16: Instantiations in four axial MRI brain images. Ordinal numbers are shown for detected features. A black dot denotes an undetected feature at location $\mathbf{a}z_i$.

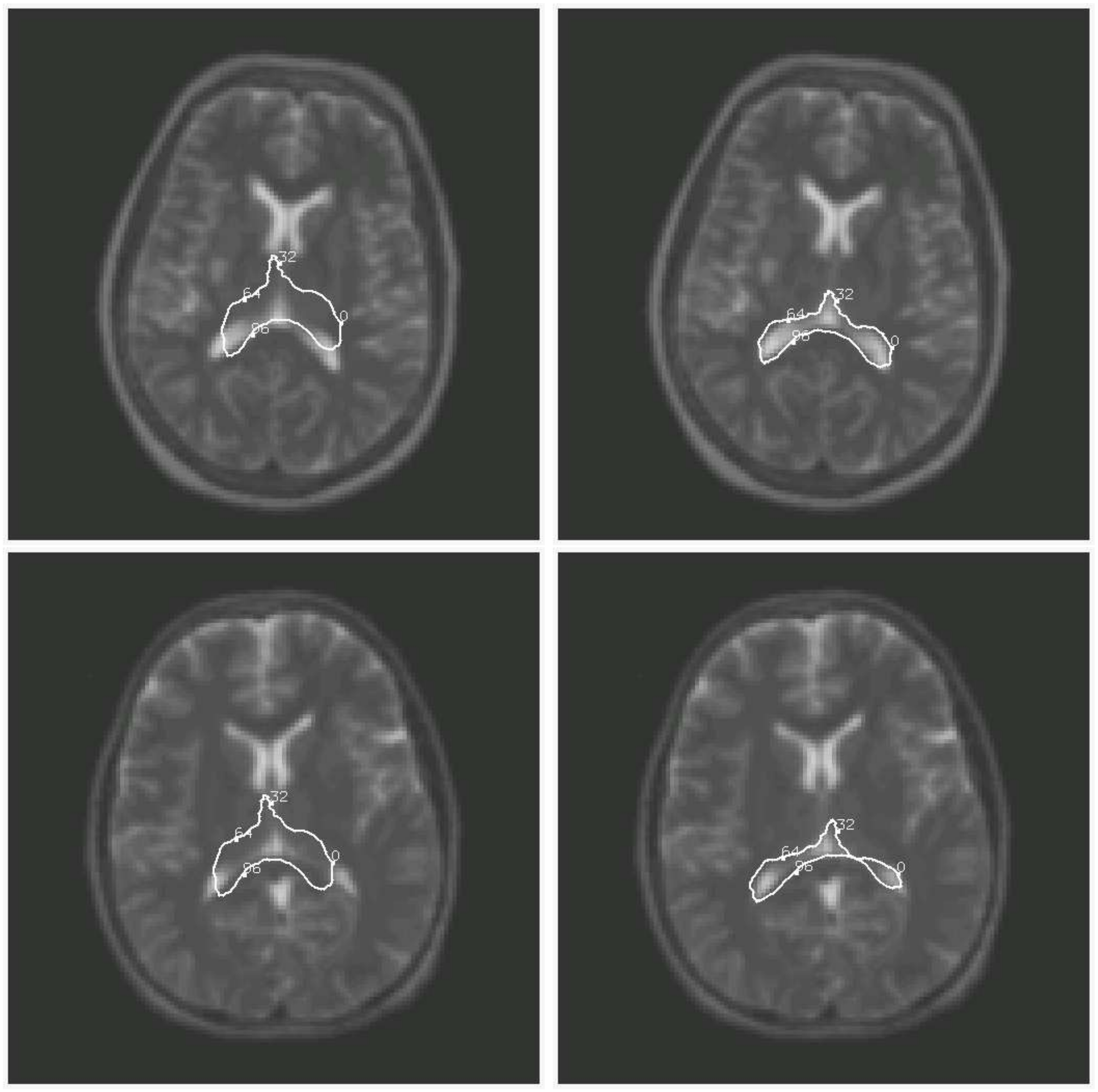

Figure 8.17: Two examples of a deformable contour initialized at the location of the anterior ventricles and the final state after running the elastic deformation algorithm.

Cootes & Taylor (1996) and most recently in Viola & Jones (2002). In these approaches the features, or certain relevant parameters, are also identified through training. However the models presented here make use of a hierarchy of binary features with hardwired invariances, and employs a very simple form spatial arrangement for the object representation. This leads to an efficient implementation of the detection algorithm. The representations described in the other papers make use of various types of linear filters with a rather large support. The invariance properties of the filters are not clear, neither with respect to photometric transformations nor with respect to deformations. In the models described in Burl et al. (1995) and Van Rullen et al. (1998) only a small number of features are used which appear to be more complex than the ones suggested here, and for faces are typically concentrated around the eyes and mouth. One problem with over-dedicating resources to only a small number of locations is the issue of noise and occlusion which may eliminate some locations. Again in the context of faces, if the goal is to detect faces on a large range of scales from 50-60 pixels between the eyes down to 10-14 pixels between the eyes, local information can be quite ambiguous.

In the work by Viola & Jones (2002) and interesting collection of binary features is defined by comparing mean intensities in rather large rectangular regions. These appear very robust to illumination changes, and are computed extremely fast. Using cumulative sums of the image intensities computed once at the start, the sum of the values in a given rectangle can be obtained with a simple operation of the cumulative sums at each of the corners.

Cutting the invariance pie

The Hough transform has been extensively used in object detection, see Ballard (1981), Grimson (1990), Rojer & Schwartz (1992). However in contrast to most previous implementations, the local features used here are more complex, and are *identified through training.* On the other hand in many of the implementations of the Hough transform referred to in Grimson (1990) there is an attempt to obtain full rotation invariance in one shot. Our implementation is intentionally *not* fully rotation invariant. It allows the use of much richer features and hence rarer in the background, reducing false positives and computation time.

This brings up the recurrent question of how to cut up the 'invariance pie' which came up in the discussion section of Chapter 5. Should the detection of a face be done at once for all rotations, including upside down faces. Or should we define ranges of pose parameters for which detection is done with one procedure, but using different procedures for different ranges, or the same procedure at rotated versions of the image. Should features be photometric invariant or should different detection models deal with different lighting conditions. For example none of the models used in this Chapter can detect the object if the polarity is flipped and the object becomes lighter than the background. The edge features on which the entire model is based are sensitive to the direction of the gradient modulo 2π. To detect the objects at both polarities one can either incorporate polarity invariance by taking oriented edges modulo π, detect even faces which are half darker and half lighter than the background at the price of more false positives; or run the face detector twice with the edge polarities flipped.

View based detection of 3d objects

This issue also comes up regarding the detection of 3d objects. The use of full 3d models, as suggested in Marr & Nishihara (1978) and found in much of the work described in Grimson (1990), is an extreme approach which claims the model should be invariant to all viewing angles. On the other hand the view based approach recently being proposed in Riesenhuber & Poggio (2000), Ullman (1996) and Tarr & Bülthoff (1998) requires only invariance within a range of viewing angles which represent smooth changes in the image of the object as we have done for the clip, or for chess pieces in Chapter 10. This implies that even rigid 3d objects should be modeled as deformable objects since their 2d views are not rigid at all. Moreover working with sparse models for non-rigid objects allows us to cover quite a wide range of viewing angles with one detector. Working in this context makes the understanding of 2d non-rigid object detection and recognition even more important since all the analysis has been reduced to the two dimensional domain.

On the other hand significant changes in viewing angle which completely change the resulting image, for example viewing the clip from the side or from the front, lead to separate models. Indeed significantly different views can be considered as *different 2d objects*.

Parts

In Biederman (1995) there is a strong emphasis on the role of generic 3d parts, called 'geons', as the building blocks for 3d object representations. These objects which consist of blocks, cylinders and other simple geometric structures, are detected in a view-based manner, but the idea is that not too many views are required to model them, and that the entire library of 3d objects can than be modeled quite simply in terms of rather coarse relationships between the geons. We have not experimented with this idea, but in terms of the models used in this Chapter, it implies introducing an intermediate level between the local features and the object models. Sparse models using local features would be constructed for the geons, and the complex object models would be constructed in terms of very coarse geometric relationships between the geons. If the geon detectors have very low false positive rates, such as for example the 'clip' detector, then there is no need for refined definition of the relationships between the geons, since very few spurious elements will be detected. We return to related ideas in Chapter 10.

Hierarchy of detection models

Comparing the various models described in this manuscript in terms of the set Z of points defining the template, we see that the smallest set was used in the sparse models whereas the largest sets were used for the deformable image models. In between lie the deformable contour and deformable curve models. But the size of the template in the sparse models is really a matter of choice. The number of stable local features can be rather large, in particular if the probability threshold ρ used in training is lowered. Indeed taking more and more local features and representing them as in figure 6.9 we would get a very dense configuration of edges, see for example face models shown in Chapter 11.

In most detection experiments in this Chapter we limited ourselves to 20 local features. However in training for edge arrangements on faces for example, close to 100 were found, with above threshold probability. There may be two reasons not to use all 100 local features. The first is that the main computational burden of the algorithm, for $n_r = 3$ is in Step I, in particular the calculation of the locations of all the local features. The second is that some of these features may be highly correlated on background. Ignoring the computational cost, and assuming that the correlations are not strong - it would be preferable to use more local features from the point of view of false positive and false negative probabilities.

Following a detection, one could smoothly deform the model to best fit the transformed data - i.e. the local features extracted from the image. This is precisely what is done by the Bernoulli model for deformable images. In other words feature based image deformation is a natural algorithmic sequel to the feature based detection using sparse models. It is a refinement of the detection process. On the other hand the 'dense' Bernoulli model can be viewed as the 'true' model, which is difficult to compute if pose is not predetermined. This is obtained using a sparse subset which enables efficient detection of coarse instantiation information, providing an initialization to the smooth deformable model which fine tunes the instantiation in terms of the full model. The one-dimensional models can be viewed as intermediate approximations which identify continuous curves given the initial matching of the sparse set of local features. So rather than a collection of different models we can embed all these models as successive approximations to a 'dense' two dimensional model.

A related but different approach to coarse to fine modeling and computation is presented in Fleuret & Geman (2001). Pose space is recursively partitioned and a detector is trained for each cell of pose space using examples with poses randomly sampled from the cell. In contrast to the approach here where training is performed at a reference pose. There is then a systematic method of exploring the image first in terms of detecting the object at one of the coarse poses and subsequently narrowing down to more refined poses.

The use of only eight edges types to define the local arrangements is quite limited. This family of elementary features should be extended to include texture information, and color and motion information when available. Texture discontinuities may define important transitions which are not well captured using the simple edge detectors. However once an appropriate set of elementary features is defined, the same mechanism can be used to construct the more complex local features, and finally the global models. The models described in the experiments above were all trained with the same parameters. Some small choices had to be made regarding the size of the reference grid and locations of the reference points, but no particular information we might have regarding the object is used. Other parameters produce similar results perhaps at some computational cost. In this sense we have here a generic method of producing object models from very primitive elementary features, using very small training sets.

Chapter 9

Object recognition

Recognition refers to the classification of an object or shape present in an image. Classification is meaningful only when the image under investigation contains a single object. Thus it must be preceded by some procedure whereby a subimage of the entire scene is selected, in which hopefully only one object is present. This is still quite vague since the subimage could for example contain only one whole object, but parts of many others. These may not necessarily pass as objects however they are part of the subimage and need to be ignored in the classification. In other words one would want to identify on-object pixels, and use only those to classify. This is precisely the segmentation problem discussed in the Introduction. Some preliminary ideas for 'segmentation' based on top-down flow of information, as opposed to a purely bottom-up approach, are explored in Chapter 10. There detection is used to identify instantiations of *coarse* models, which detect *object clusters* with elements from several object classes, and following detection a classification among these classes is performed.

In this Chapter we proceed as though some mechanism is able to identify a region of interest in the large image, which contains an object and not much else. The goal is to classify which object is present. We can even assume that the region is registered to a fixed size reference grid. Either a bounding rectangle is provided and the ratio between the largest dimension (width or height) and the dimension of the reference grid determines a scaling. Alternatively a pose has been estimated, determining a region of interest in the image, which is mapped into the reference grid.

We work in the context of characters and symbols since it is easy to produce data, and study various aspects of the classifiers. The two examples used in this chapter are the LaTeX database which we have already encountered and some handwritten digit databases. In the first case it is possible to control the degree of variability in each class and study the behavior of the classifiers in different situations. Moreover it is interesting to study how classifiers behave in the presence of hundreds of classes. Ultimately one would hope to be able to recognize among the thousands of objects the human is familiar with (see Biederman (1995)). Figure 9.1 shows the 293 prototypes, and figure 9.2 shows 20 samples of one deformed symbol, as well as some examples of deformations of a random collection of other symbols. The handwritten digit databases are used as a reality check to see how well one can perform on real data. We use the NIST Special Database, Garris & Wilkinson (1996), and parts of the USPS database produced and distributed by CEDAR, SUNY Buffalo. One could also experiment with recognizing views of 3d objects placed against homogeneous backgrounds, but then there is hardly any difference from recognizing shapes and characters, where more abundant data bases are available. Object recognition in more complex gray level images is explored in Chapter 10.

It is important to note that the particular domain of optical character recognition (OCR) has many interesting dedicated solutions, some of the most successful being Simard et al. (2000) and LeCun et al.

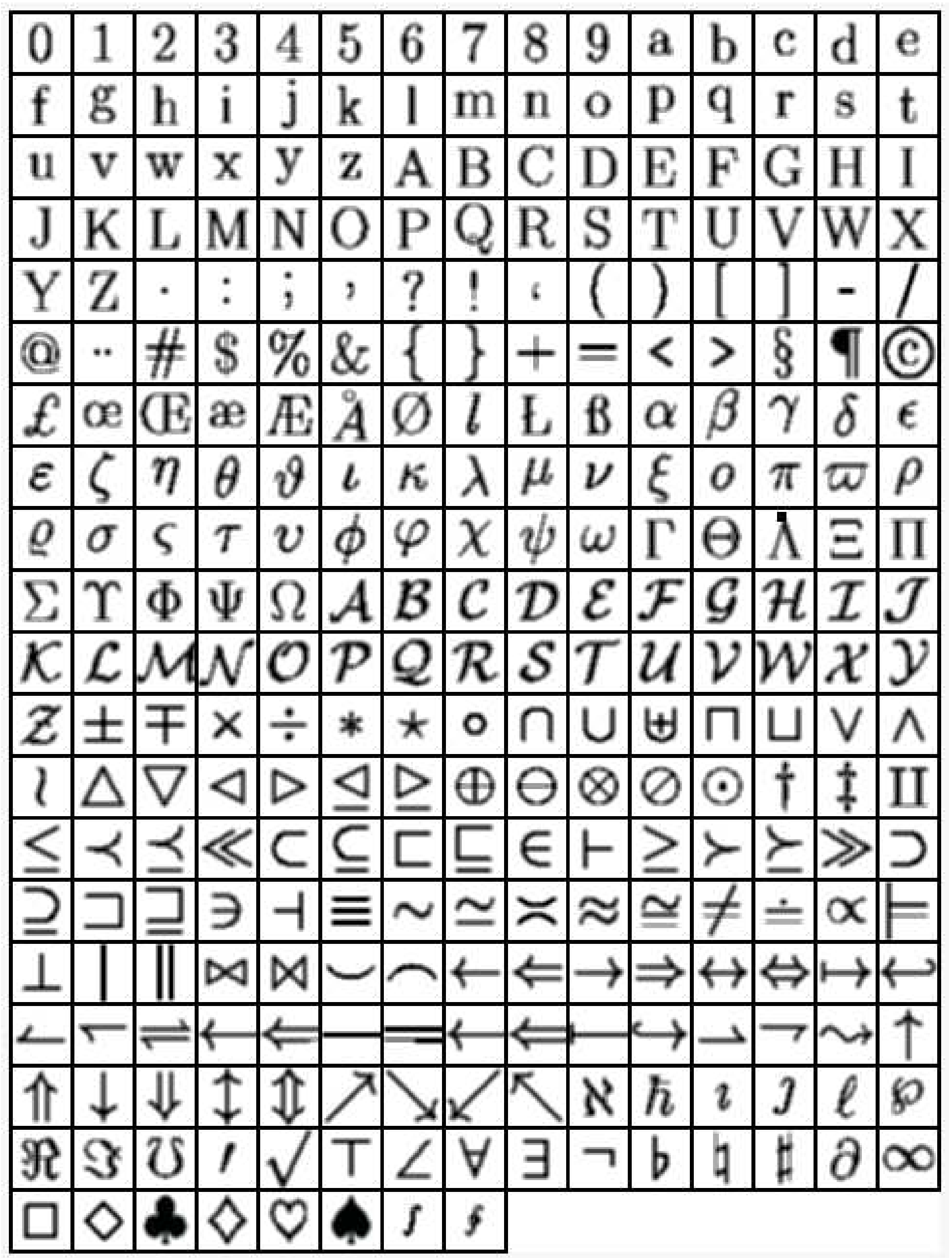

Figure 9.1: The 293 LaTeX symbols

(1998), which are discussed in the last section. Extensive reviews can also be found in Nagy (2000) and Plamondon & Srihari (2000). Although we will be working with binary character data, we want to be able to extend the methods to objects in gray level images. For that reason we will also experiment with the photometric invariant binary features introduced in earlier sections, which have proved to be powerful for detection in gray level images.

Shape classification with arrangements of local features

In the preceding chapters we have seen that objects and shapes can be detected in terms of sparse models defined as flexible arrangements of local image features. The simplest model encountered corresponds to the first step of the counting detector and has the form of a *star* type arrangement where the features are constrained to lie in regions defined relative to a center, see figure 8.1. In this chapter we consider the problem of classification among several shapes using similar ingredients. Arrangements of local features are used to discriminate among objects. Geometric and photometric invariance are explicitly incorporated in the flexibility introduced in the arrangements, as well as in the definition of the local features.

Two alternatives to defining the local features are explored. The first employs a form of local image coding which is adapted to the particular family of images being analyzed, in this case binary images. The alternative to such dedicated local features is to employ the same edges and edge arrangements used for detection in the previous chapters. These are generic and not adapted to a particular data set, but have proven to have the desired invariance properties. As we will see, the results with the two families of features are comparable. Two approaches to describing the local feature arrangements are explored.

- **Absolute**. An arrangement is a list of feature/region pairs denoting which features are expected to be found in which region of the reference grid. The larger the regions the more invariant is the arrangement to various linear and non-linear deformations but then the arrangement is less discriminating. Since information is expressed in terms of absolute regions in the reference grid, pre-registration to the reference grid is assumed here.

- **Relational**. Arrangements are described in terms of coarse constraints on the *relative angles* between pairs of features. There are no fixed regions or locations in the definition of these arrangements. The information is not coded in terms of absolute locations. This means that there is no need for pre-registration of the image to the reference grid. This approach will allow us to classify images which are of different dimensions. The arrangements are similar to those employed in the graphical template algorithm in Chapter 7, and in the object detection algorithm in Chapter 8.

9.1 Classification trees

Most literature on classification methods assumes that a fixed size predictor vector, $X_f, f \in \mathcal{F}$, is precomputed for each data point. The main focus is then which type of classifier to employ, how to train and how to predict test error. A large variety of classifiers have been proposed over the years, the most popular ones being Fisher's linear discriminant analysis and variations thereof, discussed in Duda & Hart (1973), and Ripley (1994), feed-forward neural networks, extensively covered in Bishop (1995), and Ripley (1994), support vector machines, see Vapnik (1995), and binary classification trees, see Breiman et al. (1984).

In this chapter recognition will be achieved using binary classification trees, which are especially well suited for tapping into the information provided by the feature arrangements, in particular the relational arrangements. A binary query is associated to each node in the tree. The query at the top node is applied to a data point. If the answer is 'yes' the data point moves to the right hand child node, if it is 'no' the data point moves on to the left hand child node. At the new node the associated query is applied. Again

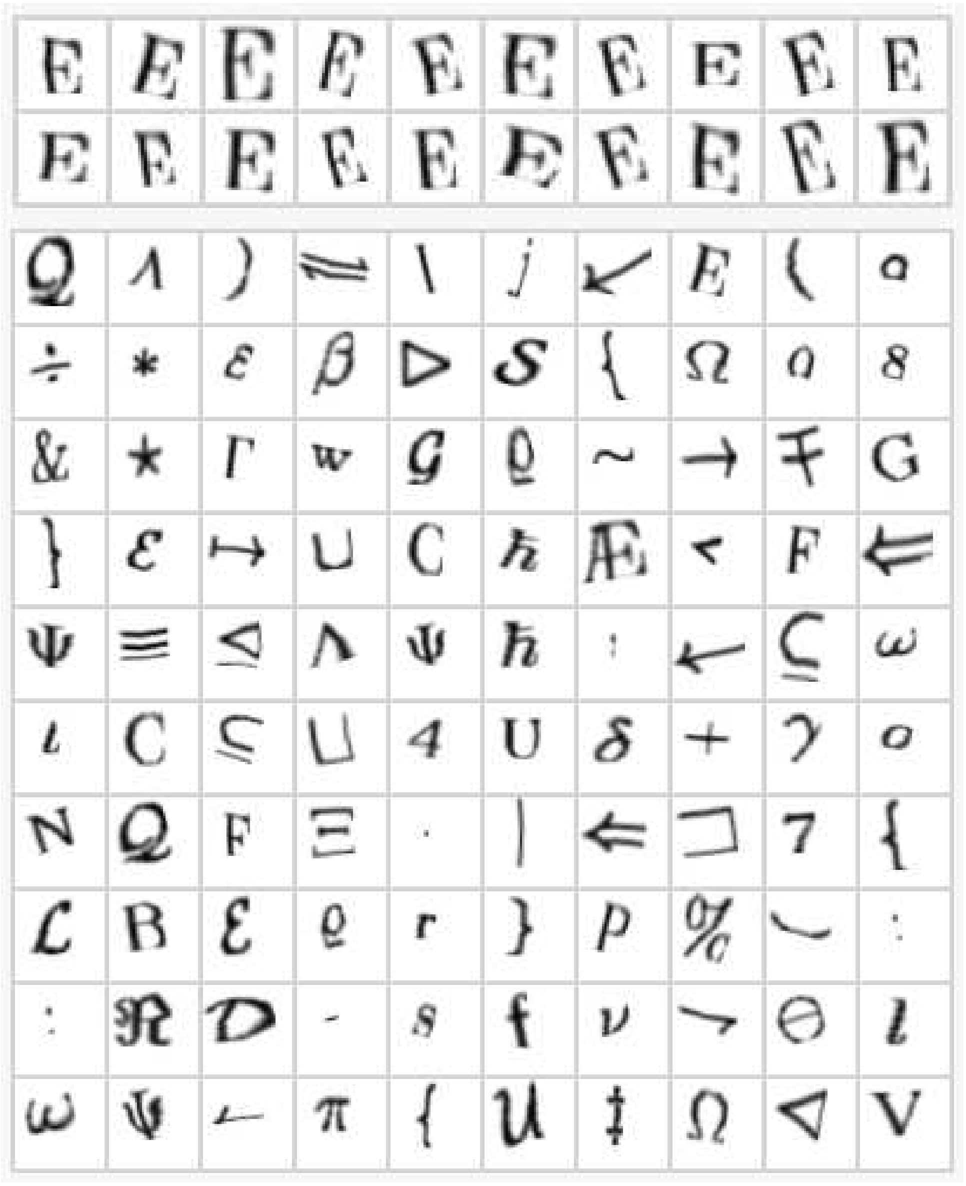

Figure 9.2: Top: Deformed images of one symbol. Bottom: One deformed sample from a collection of symbols

according to whether the answer was positive or negative the data point proceeds right or left. When the point reaches a terminal node it reads out a label associated to that node. The queries attached to the nodes are found during training. The goal is to split the training data at a node so that the distribution on class in the children nodes will be as peaked as possible.

As we will see it is crucial to produce *multiple* classification trees, from the same training set. Relying on one classifier is unstable even with rather large training sets. Using a randomization procedure classification trees are produced, which are significantly different from each other. They classify the data from 'different points of view'. Combining the output of such collections of classifiers yields drastic reductions in error rates. Randomization also allows us to grow a tree trained on tens of thousands of images in a few seconds. Typically other classifiers take much more time to train. Moreover there are very few parameters to set in growing the tree and it appears that the results are not too sensitive to the setting of these parameters.

Recognition with trees is also fast. A data point simply proceeds down the tree and at each node the appropriate query is applied. The number of operations is thus determined by the depth of the tree. This is typically no more than 10 on average with the large datasets employed here. We start by describing how a classification tree is grown.

Training

Let $\mathcal{L}$ be the training data set with data from K classes. For an element $\omega \in \mathcal{L}$ let $1 \leq Y(\omega) \leq K$ denote the class label and let $X(\omega) = \{X_f\}_{f \in \mathcal{F}}$ be the vector of predictors, assumed to be binary. Let $\mathcal{L}_t$ be the data at node t, let $\hat{P}_t$ be the empirical distribution determined by the data points at node t, i.e. the uniform distribution on $\mathcal{L}_t$, and let F_t be some subset of the full set of predictors $\mathcal{F}$. For each $f \in F_t$ calculate the conditional entropy on class given X_f, i.e.

$$\begin{aligned} C(f) &= H_t(Y|X_f) \\ &= -\hat{P}_t(X_f = 1) \sum \hat{P}_t(Y = c|X_f = 1) \log \hat{P}_t(Y = c|X_f = 1) \\ &\quad -\hat{P}_t(X_f = 0) \sum \hat{P}_t(Y = c|X_f = 0) \log \hat{P}_t(Y = c|X_f = 0). \end{aligned} \tag{9.1}$$

This is a special case of the conditional entropy defined in equation 4.19 since X_f is binary.

The function $C(f)$ is a measure of the 'purity' of the split induced by X_f. The aim is to produce children nodes where the data is more concentrated around one class or a small number of classes. Other purity measures have been used to determine node purity and are described in Breiman et al. (1984). The overall performance is typically not affected by choice of purity measure. Pick f_t which minimizes $C(f)$, over $f \in F_t$. The associated predictor X_{f_t} most reduces the entropy or uncertainty regarding class, and f_t becomes the query associated to node t in the tree.

All training data points in $\mathcal{L}_t$ for which $X_{f_t} = 1$ proceed to the 'yes' node t_y and form the set $\mathcal{L}_{t_y}$ and all those for which $X_{f_t} = 0$ proceed to the 'no' node t_n and form the set $\mathcal{L}_{t_n}$. Note that f_t is chosen based on the training data at node t, which determines $\hat{P}_t$. All these data points have answered the same way to all queries along the path from the root node to t. Thus using $\hat{P}_t$ is equivalent to using the original empirical distribution at the root node, but conditioning on the sequence of answers to the queries along the path.

The set of predictors F_t inspected at node t depends on the context. In all applications reported here it is a random subset of $\mathcal{F}$. The nodes of the tree are split depth first starting from the root node. Various criteria can be used to stop the splitting. For example a lower bound on the number of data points at a node, or on the number of data points in the top two classes, or on the entropy on class. Also if none of the candidate predictors further reduces entropy, the node is not split and becomes a terminal node.

Once the tree T is grown, at each *terminal* node t we denote the conditional distribution on class given that node by $P(Y = c|t)$. This distribution is estimated using the training data through

$$\mu_T(c) \doteq \hat{P}(Y = c|t) = \frac{|\mathcal{L}_t^{(c)}|}{|\mathcal{L}_t|},$$

where $\mathcal{L}_t^{(c)}$ is the set of training points of class c at node t. These estimates are not necessarily reliable, and depend on the number of training data points that reached the node. Finally the terminal node t is labeled by the mode of μ, i.e.

$$\hat{Y}_t = argmax_{1 \leq c \leq K} \mu_t(c).$$

Testing and error

A test data point ω can now be 'dropped' down the tree. At each node t the point goes left or right according to whether the value of $X_{f_t}(\omega) = 0$ or 1. The data point reaches some terminal node denoted $T(\omega)$, and is classified according to the label $\hat{Y}_{T(\omega)}$ attached to that node.

It is possible to grow the tree until pure nodes are reached, meaning that the error rate on the training set is 0. This of course does not imply a 0 error rate on the test set, since the estimated 'pure' conditional distributions in the terminal nodes are unreliable. More reliable estimates are obtained with shallower trees where more training data is present in each terminal node. Then the error rate on the training set is non-zero however the difference between training error rate and testing error rate may be smaller. There have been many attempts to deal with this payoff by growing pure trees and then pruning them backwards, see Breiman et al. (1984). This more refined processing becomes unnecessary when multiple trees are grown.

Note that if the depth of the tree is say 10, a data point dropped down the tree has been classified with only 10 predictors having been evaluated. In our case there may be thousands of predictors, and obviously a very small number has been used to classify the data point. One way to access this large pool of predictors is to grow very deep trees, but even with datasets of several tens of thousands one runs out of data quite quickly. Only very few terminal nodes will be of depth greater than, say, 20. The alternative is to grow multiple trees instead of only one and then somehow combine the information.

Unsupervised trees

There are situations when data is provided without labels and we seek to find clusters of similar shapes, which can subsequently be labeled. This is called unsupervised learning, also known as clustering or vector quantization. For a broad range of methods see Duda & Hart (1973), Gersho & Gray (1992), Ripley (1994), and Bishop (1995). Here we describe one way to achieve this using binary trees.

Since no labels are provided all data points with exactly the same values of *all* the predictors are a 'class'. Thus there are $2^{|\mathcal{F}|}$ classes, where $|\mathcal{F}|$ denotes the number of predictors, and the 'class' variable Y and the predictor vector X uniquely determine each other. Recall that at node t we denote by $\hat{P}_t$ the empirical distribution on the data at that node, and H_t is the entropy with respect to that distribution. Using the same cost function introduced in equation 9.1 for the case of labelled data, and the relation between conditional entropy and joint entropy of equation 4.20, we write

$$\begin{aligned} H_t(Y|X_f) &= H_t(Y, X_f) - H_t(X_f) \\ &= H_t(Y) - H_t(X_f). \end{aligned} \tag{9.2}$$

The second equality follows from the fact that Y completely determines X and in particular X_f, so that the joint entropy of Y and X_f is the same as the entropy of Y. Since $H_t(Y)$ does not depend on f,

$$argmin_f H_t(Y|X_f) = argmax_f H_t(X_f). \tag{9.3}$$

Since X_f is binary, setting $p_f = \hat{P}_t(X_f = 1)$ we have

$$H_t(X_f) = -p_f \log p_f - (1 - p_f) \log(1 - p_f).$$

The function $-p \log p - (1-p) \log(1-p)$ is concave and maximized at the value 1/2, so that the maximization in 9.3 reduces to searching for that predictor f which is closest to splitting the data at the node *in half*, or

$$C(f) = |\hat{P}_t(X_f = 1) - 1/2|.$$

A stopping criterion based on depth or number of data in a node can be used.

Such trees provide some form of partitioning of the population of images. Typically images falling into the same terminal node will be similar in shape since they have in common the answers to the sequence of queries on the path from the root of the tree to the terminal node. Furthermore if class labels are assigned to some collection of images, possibly different images from those used to produce the unsupervised tree, it is possible to drop these down the tree and obtain class distributions at the terminal nodes. The tree can then be used as a classification tree.

Multiple trees

Randomization

In the description of the basic tree growing protocol we already mentioned the possibility of choosing the set F_t randomly from all possible predictors $\mathcal{F}$. This could be a rather small subset, indeed in the experiments reported below we sometimes sample several tens from a collection of hundreds of thousands of possible predictors. This randomization allows us to generate different trees from the same training data set. A data point dropped down two such trees will have had different predictors evaluated along the paths from root to terminal nodes.

There are two ingredients in randomization. The most important is choosing the sets F_t of candidate predictors at each node as a random sample from the collection of all possible predictors. The second is estimating the optimal predictor using a random sample of the data points at each node. The sizes of these two random samples are parameters which need to be set. In most experiments below both are on the order of a few hundred or less.

Aggregation

After N trees $T_1, ..., T_N$ have been produced, the optimal way to aggregate the information is to maximize the posterior given the N random variables defined by the trees. Namely classify a test point ω as

$$\hat{Y}(\omega) = argmax_{1 \leq c \leq K} P(Y = c | T_1(\omega), ..., T_N(\omega)),$$

where $T_n(\omega)$ is the terminal node reached by ω in tree T_n. This posterior is impossible to estimate for much the same reason it was impossible to grow a very deep tree. There is not enough data.

However if the random variables T_n, defined by the trees, were *conditionally independent* given Y, i.e.

$$P(T_1 = \tau_1, \ldots, T_N = \tau_N | Y = c) = \prod_{n=1}^{N} P(T_i = \tau_i | Y = c),$$

the posterior on class given the N trees could be given analytically in terms of the individual posteriors. Specifically given a sample point ω let $\tau_i = T_i(\omega), i = 1, \ldots, N$. If the prior on class is uniform, $P(Y = c) =$

$1/K$, then using Bayes' rule twice, and the fact that the trees are conditionally independent we have

$$\begin{aligned} P(Y=c|T_1=\tau_1,\ldots,T_N=\tau_N) &= P(T_1=\tau_1,\ldots,T_N=\tau_N|Y=c)\cdot\frac{C_1}{K} \\ &= C_2\prod_{n=1}^{N}P(T_n=\tau_n|Y=c) \\ &= C_3\prod_{n=1}^{N}P(Y=c|T_n=\tau_n)P(T_n=\tau_n) \\ &= C_4\prod_{n=1}^{N}P(Y=c|T_n=\tau_n), \end{aligned}$$

where C_1, C_2, C_3, C_4 do not depend on the class c. Thus maximizing the posterior given the output of all the trees is equivalent to maximizing

$$\sum_{n=1}^{N}\log P(Y=c|T_n=\tau_n),$$

over $c=1,\ldots,K$. Using estimates of these conditional probabilities the classifier would be

$$\hat{Y}(\omega)=argmax_c\sum_{n=1}^{N}\log\mu_{T_n(\omega)}.$$

This assumption of conditional independence is of course very strong and unrealistic. Also using logarithms may be somewhat unstable. The simplest alternative is to directly average the terminal distributions, namely to classify by maximizing

$$\begin{aligned} \hat{Y}_A(\omega) &= argmax_{1\le c\le K}A_N(c,\omega), \text{ where} \\ A_N(c,\omega) &= \frac{1}{N}\sum_{n=1}^{N}\mu_{T_n(\omega)}(c), \quad c=1,\ldots,K. \end{aligned} \tag{9.4}$$

We call $A^{(N)}$ the aggregate distribution and $\hat{Y}_A$ the *aggregate classifier*. As will be shown below aggregating multiple trees leads to drastic decreases in error rates relative to the best achievable individual tree in the context of shape recognition. It provides very robust classification even with quite small data sets and is quite insensitive to parameter settings. In section 9.5 we provide some additional pointers as to why this aggregation produces significant improvements.

Boosting

Thus far we have described a mechanism for generating different trees using randomization. Each tree is produced disregarding all the others according to the same protocol, determined by the number of random queries sampled at each node, and the stopping criterion on splitting nodes. Another related approach to producing multiple trees is known as *boosting*, as described in Freund & Shapire (1997), Schapire, Freund, Bartlett & Lee (1998), where the trees or more generally the classifiers are produced with some dependence on the previously generated classifiers. The idea is that more effort should be dedicated to the 'hard' examples, namely those examples in the training set which have been misclassified by the trees already grown. This is done by using a weight vector on the training data, with the weight on a data point increasing every time it is misclassified and decreasing if it is correctly classified.

In the simplest two class case this is done by setting an initial uniform weight vector W_1 on the training set $\mathcal{L}$. A tree T_1 is grown with the training data using some chosen protocol. The training error $e_1 = \hat{P}(Y \neq \hat{Y}_{T_1})$ of the tree is obtained, where we recall that $\hat{P}$ is the empirical distribution and $\hat{Y}_t$ is the class label attached to the node t. The misclassified points in the training set have their weight increased by a factor of $(1-e_1)/e_1$. The weights on the training set are then renormalized to produce the new weight vector W_2. After $n-1$ trees $T_1, \ldots, T_{n-1}$ have been produced and given an updated weight vector W_n, a new tree T_n is grown with the weighted training sample. *All probabilities and entropies are calculated using the empirical distribution $\hat{P}_{W_n}$ determined by the current weights on the individual training points.* In particular, the training error e_n of T_n is evaluated in terms of the weighted training data. The weights on the misclassified points are again multiplied by $(1-e_n)/e_n$, and the entire weight vector normalized to produce W_{n+1}.

After N trees are produced they are aggregated either as described in equation 9.4 or as proposed in Schapire, Freund, Bartlett & Lee (1998) by a weighted vote between the N classifications Y_{T_n}, namely

$$\begin{aligned}
\hat{Y}_B(\omega) &= argmax_{1 \leq c \leq K} A_N^B(\omega, c), \quad \text{where} \\
A_N^B(\omega, c) &= \sum_{n=1}^{N} \beta_n 1_{Y_{[T_n(\omega)=c]}}, \quad \text{and} \\
\beta_n &= \log\left(\frac{1-e_n}{e_n}\right).
\end{aligned} \tag{9.5}$$

In Friedman et al. (1998) it is shown that the somewhat mysterious protocol of boosting, together with the particular weighting in 9.5, corresponds to a stepwise gradient descent on a cost function formulated in terms of an exponential moment of the aggregate classifier, which is evaluated on the training set. Under quite general conditions this gradient descent procedure is guaranteed to decrease the value of the cost function very rapidly towards zero, on the *training set*. However typically on *test* sets this same cost function diverges very quickly. So the success of boosting can not be attributed to the particular cost function being used. Indeed many variations, including the one used in the experiments below, produce very good results. The main reason for this success lies in the fact that boosting is producing weakly dependent trees, perhaps even more so than the randomization protocol, as discussed in section 9.5 at the end of this Chapter.

The original boosting protocol encounters problems when $e_n \leq .5$, namely when the error of classifier T_n is higher than .5. This is by no means uncommon in multi-class problems. We therefore substitute a reweighting of the form $1/e_n$. The protocol no longer has the appealing interpretation of a gradient descent on a simple cost function. However it performs just as well and produces stable results. All experiments reported here with boosting employ this reweighting scheme. Furthermore we use the aggregation scheme of equation 9.4 as opposed to that of equation 9.5, so that the trees are all equally weighted.

9.2 Object recognition with trees

We now return to the original problem of object recognition. The crucial question is which predictors to use in producing the tree, or any other classifier. The most naive approach would simply use the gray level intensities at each pixel on the reference grid, which is the original form of the data. However we have seen throughout previous chapters that this would not accommodate the geometric and photometric variablities inherent in the problem. The gray level intensities are therefore again transformed into collections of binary features, robust to *local* geometric variations and to photometric transformations. Such binary features can then be *spread* (the OR-ing operation) in various ways to obtain flexible representations invariant to *global* geometric variations.

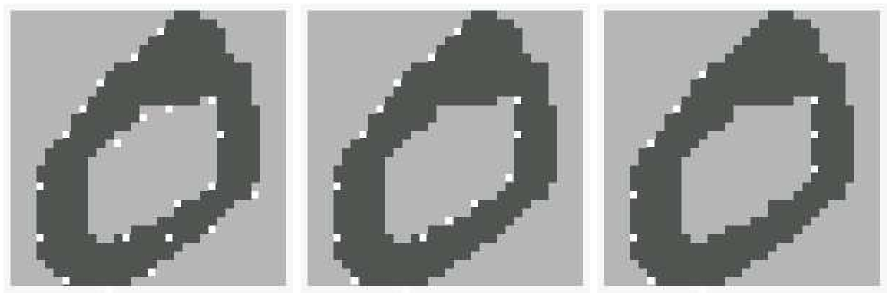

Figure 9.3: Locations of features corresponding to depth 1,2 and 3 along one path of the code tree.

Local features

Two types of binary local features have been used in the experiments. The first is adapted to the particular image population being analyzed and the second is generic.

Micro-image codes

In the particular setting of binary images we define local features which are adapted to the training data. These ideas can in principle be extended to gray level images, however in this more complex context it is not clear how to accommodate photometric invariance as well as invariance to local deformations. This is definitely a direction requiring more research.

Start with a sample S of small 4×4 subimages, henceforth called micro-images, of the training data, keeping only those which are not constant, namely those that are neither all black or all white. Produce an unsupervised tree of depth d with this data, using the methodology described in section 9.1. There are only 16 predictors corresponding to the pixel value (1/0) at any of the 16 locations in the 4×4 micro-image, thus there are at most 2^{16} 'classes' in this data. Such a tree provides a partition or quantization of the population of binary micro-images. Each terminal node of the tree contains approximately the same number of micro-images. The path from the root to a terminal node determines a subset of the form $\{s \in S : s_{i_0} = a_1, \ldots, s_{i_4} = a_4\}$ where $i_0, \ldots, i_4$ are the locations (on the 4×4 grid) used at the nodes along the path and $a_1, \ldots, a_4$ are 0 or 1 according to whether the path goes left ('no') or right ('yes') at a node. Not surprisingly the most common micro-image in the deeper nodes of the tree actually represents some form of oriented edge.

Any local 4×4 micro-image is now classified by dropping it down the tree and determining which terminal node it belongs to. The micro-image is labelled not only by the terminal node, but also by the intermediate nodes along the path to the terminal node. Thus if the tree is of depth 5 the micro-image is given 5 labels. The labels corresponding to depth 1 nodes are assigned to about half the data. They are very crude characterizations of the micro-image and are very common. The labels further down along the path to the terminal node represent finer and finer characterizations. In figure 9.3 we show the locations of features corresponding to depth 1,2 and 3 along one path of the code tree. We perform a clustering on the locations of the features so that only one instance of each local feature type is present in each 5×5 region. Due to this clustering the locations of depth k are not precisely a subset of the locations for depth $k - 1$.

In the experiments we find that all levels of detail are necessary not only the finest one. The coarser nodes are more invariant to local deformations, the finer nodes convey more information on the local binary configuration.

In figure 9.4 we show 3 levels of this tree and the most common configuration found at each of the eight depth 3 nodes. Observe that the tree partitions among micro-images corresponding to edges of different orientations. This is achieved in a purely data driven manner. The most common images in deeper nodes typically correspond to boundaries with a slight curve or bend, and may be viewed as a conjunction of two edges.

Generic local features produced from edges

From the quantization described above we see that oriented edges are reasonable representations of clusters of micro-images, as are finer features, represented by local edge arrangements, of which we have made extensive use in previous chapters. As an 'engineered' alternative to the features defined above we use the following 264 local features: 8 edges corresponding to the 8 basic orientations defined in section 5.4 and $256 = 8 \times 8 \times 4$ two-edge arrangements defined in Chapter 6. We use the larger wedges shown in figure 6.7 corresponding to the angles $0, \pi/4, \pi/2, 3\pi/4$. Due to symmetry other angles are not needed. The advantage of these features is that they are not specifically taylored to any particular data set and can be used in both binary and gray level images.

Absolute arrangements

Assume all images have been registered to a fixed reference grid G (say 32×32). The LaTeX database is created this way in the first place. In other cases one assumes that a bounding box has been identified for the symbol or a pose has been estimated allowing us to register the data to the reference grid. The dependence of the classifier on this step is crucial and should not be taken lightly. It implies that some form of detection is necessary prior to classification. We continue to explore this issue in Chapter 10.

The collection of binary local features $X_1, \ldots, X_N$ is now applied at all locations in G to produce a fixed dimensional binary predictor vector $X_\alpha(z), \alpha = 1, \ldots, N, z \in G$, of length $|G| \cdot N$. The gray level intensities have been replaced by an N-vector of binary variables. The set $\mathcal{F}$ is the collection of all pairs $f = (\alpha, z), \alpha = 1, \ldots, N, z \in G$. This predictor vector, after being calculated for each image in the training set, can be fed into *any* standard classifier: feed forward neural nets, classification trees, linear discriminant analysis etc.

First we should note that any of these classifiers would produce a very sensitive and unstable classifier. The reason being that under the smallest variations of an instance of a symbol of some class, the locations of quite a number of the local features will move and hence completely change the corresponding predictor vector. Invariance to local variations and a certain degree of scaling and translation must be explicitly incorporated into the predictors. With binary features this is easily and naturally done by 'spreading' (OR-ing) the output of the original local feature detectors to $s \times s$ neighborhoods. Define

$$X^s_\alpha(z) = \max_{y \in N_s(z)} X_\alpha(y), \tag{9.6}$$

where $N_s(z)$ is an $s \times s$ neighborhood of z, namely $X^s(z) = 1$ if $X_\alpha(y) = 1$ *anywhere* in the $s \times s$ neighborhood of x. This is an explicit disjunction or ORing that is directly incorporated into the data and any resulting classifier will inherit a certain degree of invariance. In our context spreading on rather large 15×15 regions yields optimal results. It should be kept in mind that while ORing increases the stability of the classifier it decreases the discriminating power of each of the coordinates of the predictor vector.

Multiple randomized classification trees are trained using these predictors and aggregated as described above. Results obtained on the LaTeX and NIST databases are reported in section 9.4 where they are also compared to results with the relational arrangements described in the following section.

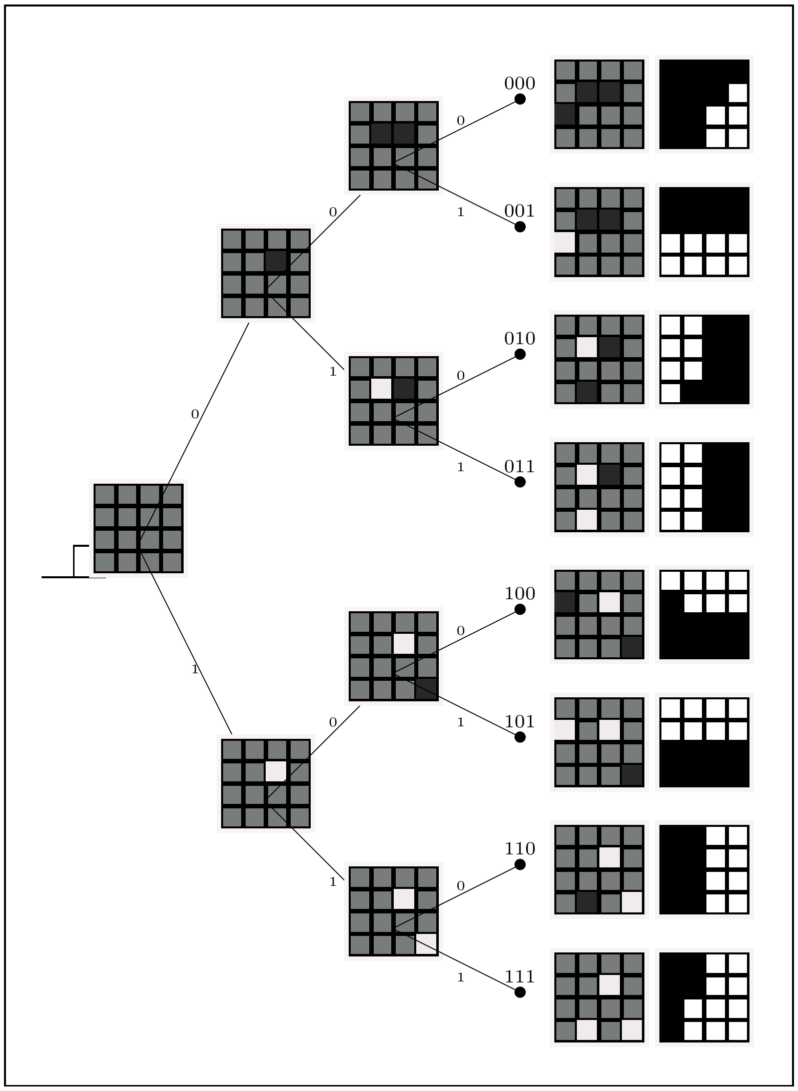

Figure 9.4: First three levels of the code tree. The location of the query at a node is shown in the children nodes as white if the value is ‘1’, and as black if the value is ‘0’. The most common configurations at the eight depth three nodes are shown.

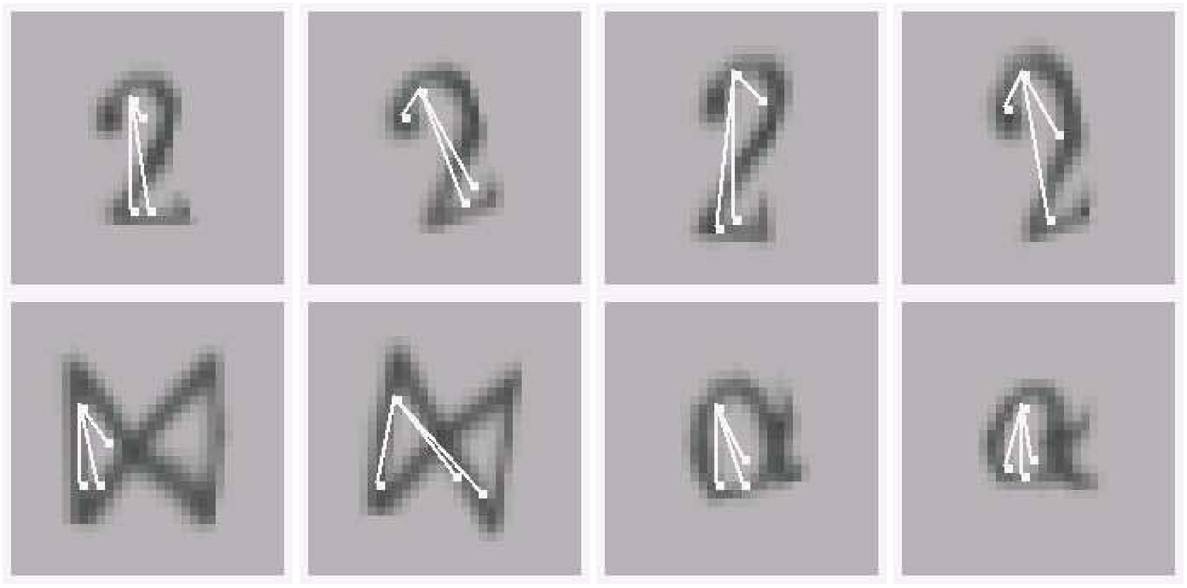

Figure 9.5: Eight images from a depth 5 node in a classification tree with the associated arrangement. Two of the queries on the path to this node were answered 'no'.

9.3 Relational arrangements

In this section we describe a more complex approach to the object classification problem. We no longer assume that the object has been registered to a reference location and scale. This means that absolute locations of features are meaningless and we can only use information regarding the relative spatial arrangement of local features. For example information on the angles of the vectors connecting pairs of local features. Once a relational arrangement is defined the associated query asks whether or not such an arrangement is present *anywhere in the image*. The arrangement is also defined in an entirely scale invariant manner it can be present at any scale. In the sequel we will define these arrangements more precisely, define a partial ordering on the set of arrangements and explain how the associated queries can be incorporated into classification trees. As will become apparent it is not clear how such 'queries' can be systematically employed in the context of other types of classifiers.

In figure 9.5 we show a collection of symbols in all of which a particular relational arrangement has been found, described in terms of a graph with edges between pairs of features between which a relation has been defined. Note the variations on the angles and distances between the nodes of the graph. There are no distance constraints in the relations defined between the features. All these images reached some internal node of a classification tree grown with relational arrangements as described below. The conditional distribution on the 293 LaTeX classes given the presence of this arrangement is far more concentrated than the original uniform distribution. The entropy is 5.74 down from the entropy on the uniform distribution on 293 class which is 8.19. Despite the large degree of flexibility introduced in the definition of the arrangement it carries a significant amount of information on the shape in the image.

Relations between local features

Define wedges $W_k, k = 1, \ldots, K_w$ centered at the origin spanning the angle $\frac{2(k-1)\pi}{K_w}$ to $\frac{2(k+1)\pi}{K_w}$. If feature X_α is present at location x and feature $X_{\alpha'}$ at location y we say that they satisfy relation R_k if $y \in x + W_k$.

Typically we use $K_w = 16$ so that there are 16 possible binary relations between any two features.

A relational arrangement is a labeled graph with vertices $V = \{1, \dots, d\}$ and directed edges defined as some subset of $V \times V$. Each vertex $i \in V$ is assigned a label $1 \leq \alpha(i) \leq n$, which corresponds to one of the local features. Each edge $(i, j) \in E$ is assigned a label $1 \leq \beta(i, j) \leq K_w$. The arrangement is present in the image if there exists an ordered set of locations $x_1, \dots, x_d$ such that a feature of type $\alpha(i)$ is present at x_i, for each $i \in V$, and $x_j \in x_i + W_{\beta(i,j)}$, for each $(i, j) \in E$. Each arrangement defines a binary variable on images: either it is present or it is not. Note that any ordered set of locations satisfying these conditions is an instance of this arrangement. Since the relations are defined very loosely and only constrain the relative angles, a relational arrangement can have a quite a number of possible instances.

We denote by $\mathcal{A}$ the set of all possible arrangements.

Note that the local edge arrangements defined in Chapter 6 are special cases of relational arrangements, where all relations are with respect to the first vertex, with an additional constraint on the distance between the local features in the relation.

Growing trees with relational arrangements

The arrangements are defined in terms of graphs, thus providing a natural partial ordering, under which a graph precedes any of its extensions. The partial ordering corresponds to a hierarchy of structure. Small arrangements with few local features produce coarse splits of shape space. As the arrangements increase in size they convey more information about the images which contain them. However fewer images contain these arrangements. One straightforward way to exploit this hierarchy is to build a classification tree, using the collection of arrangements as candidates for the queries, with the complexity of the queries increasing with the depth of the node.

Define a *minimal extension* of an arrangement $\mathbf{A} \in \mathcal{A}$ to mean the addition of exactly one local feature and one relation binding the new local feature to an existing one. Let $\mathcal{B} \subset \mathcal{A}$ denote the set of arrangements involving only two local features and one relation. Now build a tree using the protocol described in 9.1. At the root node r, F_r is a random sample from $\mathcal{B}$. Denote the chosen query $\mathbf{A}_r$. Those data points which do not have an instance of the arrangement proceed to the 'no' child node r_n where again we search through $\mathcal{B}$. Those data points which do have an instance of $\mathbf{A}_r$ proceed to the 'yes' child node r_y and have one or more instances of $\mathbf{A}_r$, which we call the *pending arrangement,* at r_y and denote it by $\mathbf{P}_{r_y}$.

At the node r_y the set F_{r_y} of candidate queries is a random sample of minimal extensions of the pending arrangement $\mathbf{P}_{r_y}$. If we select $\mathbf{A}_{r_y}$ as the query for node r_y, this will be an arrangement with three local features and two relations. The 'no' child node of r_y inherits the same pending arrangement $\mathbf{P}_{r_y}$. The 'yes' child node r_{yy} has the pending arrangement $\mathbf{P}_{r_{yy}} = \mathbf{A}_{r_y}$.

More generally, at any given node t the pending arrangement $\mathbf{P}_t$ is the same as at the parent node p if t is the 'no' child ($t = p_n$). At $t = p_y$ we have $\mathbf{P}_t = \mathbf{A}_p$, which again is a minimal extension of $\mathbf{P}_p$. F_t is always a random sample of minimal extensions of $\mathbf{P}_t$. There is no pending arrangement at a 'no' node t, for which all ancestors are also 'no' nodes, and the set F_t is again a random sample of $\mathcal{B}$.

The number of possible arrangements in $\mathcal{A}$ is essentially infinite. Using only minimal extensions greatly reduces the number of candidate queries at a node. It is also very easy to check if the minimal extension exists for each instance of the pending graph. Still there are very large numbers of possible minimal extensions at every stage. In this context randomization not only serves to create different trees. It also reduces the number of queries which need to be checked.

An illustration of the effects of these queries on splitting the data can be seen in figures 9.5 and 9.6. Figure 9.5 shows a collection of images which reached a depth 5 node, with one instance of the pending arrangement. The first row of figure 9.6 shows images which reached that same node and continued to the 'yes' child. The additional feature was found in these image in the proper angle relative to an existing feature in one of the instances of the pending arrangement. The 2's in figure 9.5 did not have such a feature and

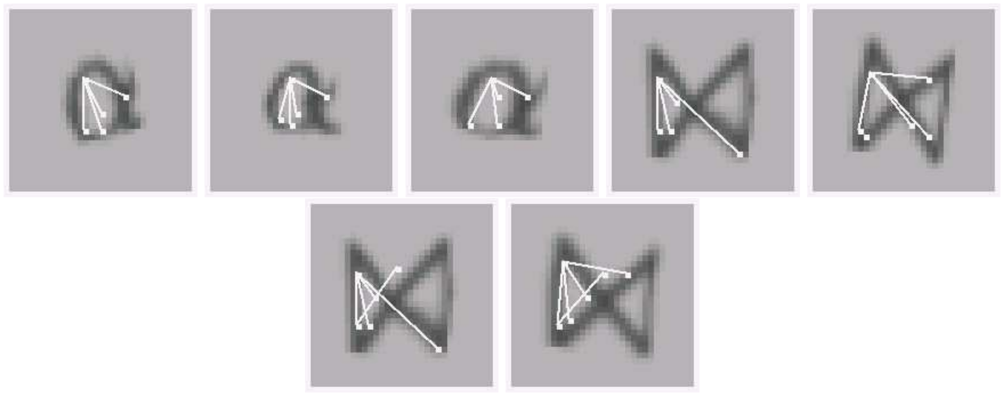

Figure 9.6: Examples of node splitting. All five images in the top row lie in the same node and have a pending arrangement with four vertices. All these images were in the node represented in figure 9.5. The two's shown in that figure went to the 'no' node. One more step of the splitting separates the α's from the $\bowtie$'s by asking for the presence of an additional feature.

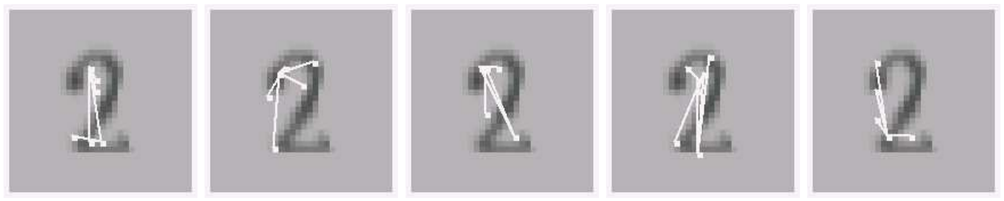

Figure 9.7: The pending arrangement at the terminal node in five different trees for a single instance of a 2.

went to the 'no' child node. One more split of the depth 6 node is also shown in the second row of figure 9.6. The α's have gone to the 'no' node and the $\bowtie$'s to the 'yes' node.

Training is more complex than for standard classification trees described earlier. At each node, a list must be assigned to each data point consisting of all instances of the pending arrangement, including the coordinates of each participating feature. If a data point passes to the 'yes' child, then only those instances which can be incremented with the new relation are maintained and updated; the rest are deleted. The more data points the more book-keeping. There is an inherent asymmetry in the tree, in that the pending arrangement at terminal nodes with many "no" answers on the path from the root will be small. However in many terminal nodes as shown in figure 9.7 the arrangement describes a complex structure, and dropping the data down the tree automatically identifies the instantiations of this structure in the image. In this sense the relational trees can be viewed as a classification *combined* with a sparse model instantiation.

The importance of using multiple randomized trees is illustrated in figure 9.7 where we show the arrangement found on a single instance of a 2 in the terminal node it reached in five different trees. The five trees are using different aspects and properties of the shape to classify this data point. A more robust classification is then possible when the information from all trees is pooled together.

Since the arrangements are defined in an entirely scale and translation invariant manner the resulting trees are very robust to scaling and translation. Indeed the incoming data does not need to be registered in any way as will be illustrated in the experiments below.

Tree Protocol	Absolute $\mathbf{Q} = 20$ $\mathbf{s} = 15$	Absolute $\mathbf{Q} = 100$ $\mathbf{s} = 15$	Absolute $\mathbf{Q} = 200$ $\mathbf{s} = 15$	Relational $\mathbf{Q} = 20$ $K_w = 8$	Relational $\mathbf{Q} = 100$ $K_w = 24$
Agg. Rate	98.7%	99.03%	98.9%	98.5%	98.8%
Ind. Rate.	90%	93%	94.2 %	77.2%	86.7%

Table 9.1: NIST: Different protocols. 100,000 training data and 50,000 test. $\mathbf{Q}$ - number of random queries sampled at a node. $\mathbf{s}$ - size of 'spread' parameter for trees with absolute arrangements. K_w - number of wedges in relative arrangements (width of wedge is $2\pi/K_w$.) $\mathbf{m} = 3$ - number of data points in second largest class at a node to stop splitting. First row: aggregate classification rate with 100 trees. Second row: average classification rate of the individual trees.

9.4 Experiments

We experiment with the LaTeX database and portions of the National Institute of Standards and Technology (NIST) database Garris & Wilkinson (1996), which consists of approximately 223,000 binary images of isolated digits written by more than 2000 writers. We use 100,000 for training and 50,000 for testing. For this database we use micro-image codes dedicated to binary images. On the LaTeX database the generic features are used.

In all trees reported below the nodes are split as long as there are at least $\mathbf{m}$ data points in the second largest class. The default value we use is $\mathbf{m} = 3$, but in some experiments $\mathbf{m}$ can reach up to several hundreds. The parameter $\mathbf{Q}$ represents the number of random queries sampled at each node while training the tree, this can range anywhere from several tens to hundreds of thousands. For absolute arrangements the parameter $\mathbf{s}$ denotes the degree of 'spread' (a detected feature is spread to the $\mathbf{s} \times \mathbf{s}$ neighborhood of the original location,) and ranges from 3 to 15. For relational arrangements the parameter K_w determines the number of wedges used and their width. At all nodes with over 200 training points we take a random sample of 200 to determine the best query from among the $\mathbf{Q}$ randomly sampled queries; Otherwise we use all training data at the node. Unless otherwise specified 100 trees are produced in each experiment.

NIST

The NIST data is initially pre-processed as follows. The original images are binary. Each image is blurred with a Gaussian filter of standard deviation 1 pixel. If the dimensions of the image are less than 32, the bounding rectangle of the digit is identified and is placed in the center of a 32×32 grid. If the largest dimension d of the image is greater than 32, the image is downscaled by a ratio of $32/d$ and is again centered in a 32×32 grid. Then the image is re-binarized using a threshold of .25 assuming the pixel values are in the range $[0, 1]$. It is necessary to binarize the image to detect the micro-image codes.

The results in table 9.1 show the effect of producing multiple classification trees. The aggregate classification rate as a function of the number of trees for one of the experiments is shown in figure 9.9. Note how very similar aggregate classification rates are obtained with very different classification rates on the individual trees. The best result obtained on one tree using no randomization at all was 95.8%, thus the aggregation of multiple randomized trees plays a very important role in reaching above 99% classification rates. With absolute arrangements similar results are obtained using a coarser grid for the features. Features are extracted on the original image, however their coordinates are rounded off to the nearest multiple of 2 or 4. The number of predictors is then reduced by a factor of 4 and 16 respectively. The $\mathbf{s}$ parameter is adjusted accordingly to $\mathbf{s} = 7$ and $\mathbf{s} = 2$ respectively. In both settings the classification rates obtained were 98.8%. In figure 9.8 we show the first 100 of the 500 misclassified NIST digits.

The trees generalize well to other data sets. For example the relational arrangement trees were applied

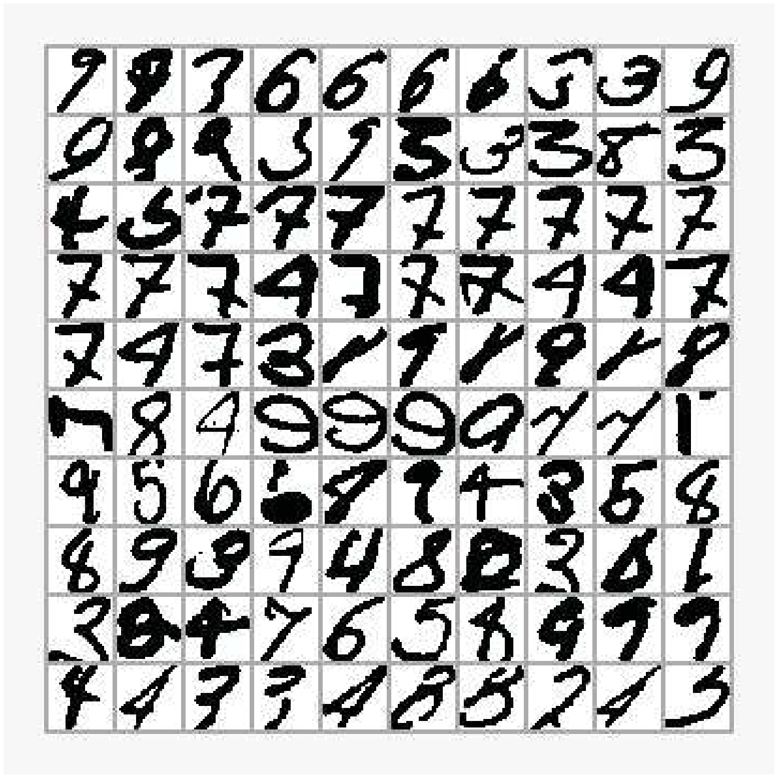

Figure 9.8: 100 misclassified NIST handwritten digits.

Sample size	5,000	10,000	50,000	100,000
Agg. Rate	97.3%	97.9%	98.6%	98.7%
Ind. Rate	80.23%	83.3%	88.6%	90%

Table 9.2: NIST: Classification rates of the aggregate classifier and the individual trees as a function of the training sample size - absolute arrangements. $\mathbf{m} = 3, \mathbf{s} = 15, \mathbf{Q} = 20$.

to a dataset of 5000 handwritten digits obtained from the USPS database produced and distributed by CEDAR, SUNY Buffalo. The classification rate was 97.3%. The conditions under which the postal digits are written are very different. In writing zip-codes on an envelope no bounding box is provided, so that their sizes and slants exhibit a larger variability than that of the NIST database.

Several other comparisons are interesting for this database. In table 9.2 we show classification rates as a function of the size of the training set, all other parameters fixed. In table 9.3 we compare results on the small 5000 sample training set as a function of the stopping rule, which affects the depth of the trees. As the value of $\mathbf{m}$ decreases the trees get deeper. In many instances of the use of classification trees, deeper trees may exhibit lower classification rates due to over fitting. This does not seem to occur in this case, probably due to the large degree of spreading. For $\mathbf{m} = 1$ we obtain pure trees, namely trees for which there is only one class of training data present at each terminal node. These appear to perform best.

Stopping criterion	**m**=20	**m**=10	**m**=3	**m**=1
Agg. Rate	96.0%	96.5%	97.2%	97.6%
Ind. Rate	72.8%	76.8%	80.2%	80.4%
Avg. depth	6.1	6.93	8.49	9.86

Table 9.3: NIST: Absolute arrangements. Classification rates of the aggregate classifier and the individual trees with 5000 training data, as a function of the stopping criterion. $\mathbf{m} = 3, \mathbf{s} = 15, \mathbf{Q} = 20$.

Stopping criterion	**m**=20	**m**=10	**m**=3	**m**=1
Rand. Agg. Rate	96.0%	96.5%	97.2%	97.6%
Boost. Agg. Rate	96.8%	96.5%	95.2%	$*$

Table 9.4: NIST: Absolute arrangements. Classification rates of the aggregate classifier with and without boosting as a function of the the stopping criterion. 5000 training data are used. Boosting is not applicable for pure trees since the training error rate is 0. $\mathbf{Q} = 20, \mathbf{s} = 15$.

Boosting

We implement the boosting protocol with the reweighting factor of $1/e_n$ and aggregate the trees according to equation 9.4. The terminal distributions μ_{T_n} are in this case estimated during training with the weighted training data. It appears that when deep trees are used so that the training classification rate is very high boosting overfits the training data and performs worse than simple randomization. However for shallower trees boosting enables the aggregate classifier to ‘overcome’ some limitations of the individual classifiers. If certain data points of different classes are hard to separate without querying a large portion of their respective strokes, shallow trees risk having them end up in the same terminal nodes, especially since the trees are aiming at improving the purity of the nodes over the entire training set. Boosting will cause the weights of such data points to increase significantly. Their relative weight in determining the purity of a node increases and subsequent trees in the boosting procedure may be able to separate these hard examples.

For the larger data set we observe the same properties of boosting. Using the parameters yielding the best performance with 100,000 training data for the randomized protocol produces a rather low classification rate with boosting. However table 9.5 shows randomized boosting outperforms simple randomization with shallower trees with $\mathbf{m} = 100$, (average depth of 9.76 vs 12.)

The best result of 99.29% *was achieved with boosting:* $\mathbf{m} = 100$ *and* $\mathbf{Q} = 1000$.

In figure 9.9 we show the aggregate classification rate as a function of the number of trees. The curves for other experiments look very similar.

Protocol	**m**=100 **Q** = 1000	**m**=100 **Q** = 20	**m**=50 **Q** = 20	**m**=20 **Q** = 20
Boost. Agg. Rate	99.29%	99.09%	99.07 %	98.8%
Ind. Rate	74.7%	62.4%	69.3%	76.4%
Avg. depth	9.8	9.8	10.8	12.0

Table 9.5: NIST: Absolute arrangements. Classification rates of the aggregate classifier with boosting for various protocols with boosting and 100000 training data..

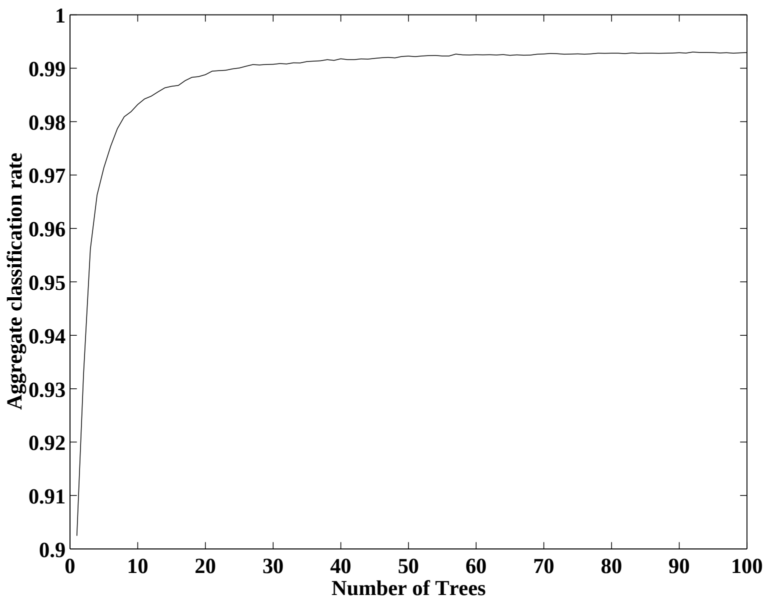

Figure 9.9: Classification rate as a function of the number of trees.

Spread parameter	$\mathbf{s} = 3$	$\mathbf{s} = 7$	$\mathbf{s} = 15$	$\mathbf{s}$=23	$\mathbf{s} = 31$
Agg. Rate	82.9%	94.7%	96.1%	94.9%	90%
Ind. Rate.	10%	28%	39%	40%	38%

Table 9.6: LATEX : Comparison of results for different values of the spread parameter $\mathbf{s}$. Absolute arrangements are used with the 264 generic features and $\mathbf{Q} = 20$.

The 'hard' NIST dataset

One additional portion of the NIST dataset, containing approximately 50,000 samples, was produced by high school students as opposed to employees of NIST. This data set is more challenging, the variability in shape and size appears to be greater. Using the best trees reported above on this more difficult set we achieve a very low classification rate of 95.2with a combination of 30,000 samples from the original set and 30,000 from the hard set, we reach over 99% on the original test set, and over 97% on a different part of the hard set. The best results on this combination of the datasets is reported in LeCun et al. (1998) and discussed below.

LATEX

Similar results emerge from experiments with the LATEX database. Here however there are only 32 training samples per image and the number of classes is 293. We use 50 samples per class for testing. Quite a large variability is introduced in the images with the synthesized deformations. Table 9.6 illustrates the importance of spreading showing that the optimal spreading box is around 15 which is *half* the image size. From this table we also see the drastic improvement in performance enabled by aggregation. Here we use absolute arrangements with the 264 generic features of edges and edge arrangements.

An interesting comparison in table 9.7 shows the effect of the number of randomly sampled questions

Num. of questions	**Q**=20	**Q**=500	**Q**=1000	**Q**=10000
Agg. rate	96.1 %	96.08%	95.9%	93.6%
Ind. rate	41.5%	58.9%	60.86%	64.99%
Boost Agg.	96.9%	97.2%	97.2%	96.6
Ind. Rate.	32%	51%	52%	56%

Table 9.7: LaTeX : Comparison of results for different values of **Q**. Top two rows randomized trees. *All* data points at a node were used to select the optimal query. Absolute arrangements, $\mathbf{m} = 3, \mathbf{s} = 15$. Bottom two rows with boosting.

Num. of questions	$\mathbf{Q} = 200$ $K_w = 8$	$\mathbf{Q} = 200$ $K_w = 16$	$\mathbf{Q} = 1000$ $K_w = 16$	$\mathbf{Q} = 20$ $K_w = 16$
Rand. Agg. Rate	94.5%	95%	95.5%	92%
Ind. Rate.	37%	35%	41%	19%

Table 9.8: LaTeX : Different protocols with relational arrangements.

on performance. No randomization is done in the data points used to select the split at each node, i.e. all data points are used to choose the optimal split. The optimal number appears to be at $\mathbf{Q} = 20$. Although the individual trees produced with higher values of **Q** have a higher classification rate the aggregate rate decreases due to the fact that the trees become more similar. The optimal performance is achieved with randomization both in the queries and in the sample used to choose the optimal split. When using $\mathbf{Q} = 200$ and a random sample of 200 of the entire training sample at a node (when the size is larger than 200), the classification rate achieved is 96.7% with mean classification rate per tree of 52%. A different pattern emerges for boosting. Here there is improvement with the number of random queries used and the best rate achieved is 97.2% with $\mathbf{Q} = 500$. This difference is discussed in section 9.5.

Clutter

The performance of the relational arrangements under various protocols is shown in table 9.8. On the whole it seems that the absolute arrangements perform slightly better than relative arrangements. However this is in an idealized setting where the object is well centered in the image.

It is therefore of interest to study the effect of clutter on the classifier. The clutter is produced by taking c additional random symbols, extracting a random 6×6 window and adding it to the image at random locations. Examples of cluttered images are shown in figure 9.10, with $c = 2$ in the top row and $c = 4$ in the bottom row. After clutter is added the bounding box of the data is recomputed and used for centering the image. This mimics the effect of clutter on the image registration procedure. The relational arrangements are by definition entirely translation invariant. With absolute arrangements despite the spreading there is some sensitivity to moderate shifts.

The performance with absolute arrangements degrades faster than that with relational arrangements, as shown in table 9.9. It appears that this is not only due to shifting, but also due to higher sensitivity to erroneous noisy features which are present in the background. However the drop in performance is too large in both cases implying something is missing in these classification scheme. Some consolation can be found if instead of taking the mode of A_N, we check if the correct class is among the top 5 classes. After all we start out with 293 classes and reducing to a choice among 5 is still useful. The results are shown in table 9.9 in parantheses.

Possibly the final discrimination between these 5 candidates should be done using some more detailed analysis in terms of a deformable model. We have already seen that the deformable curve models are very

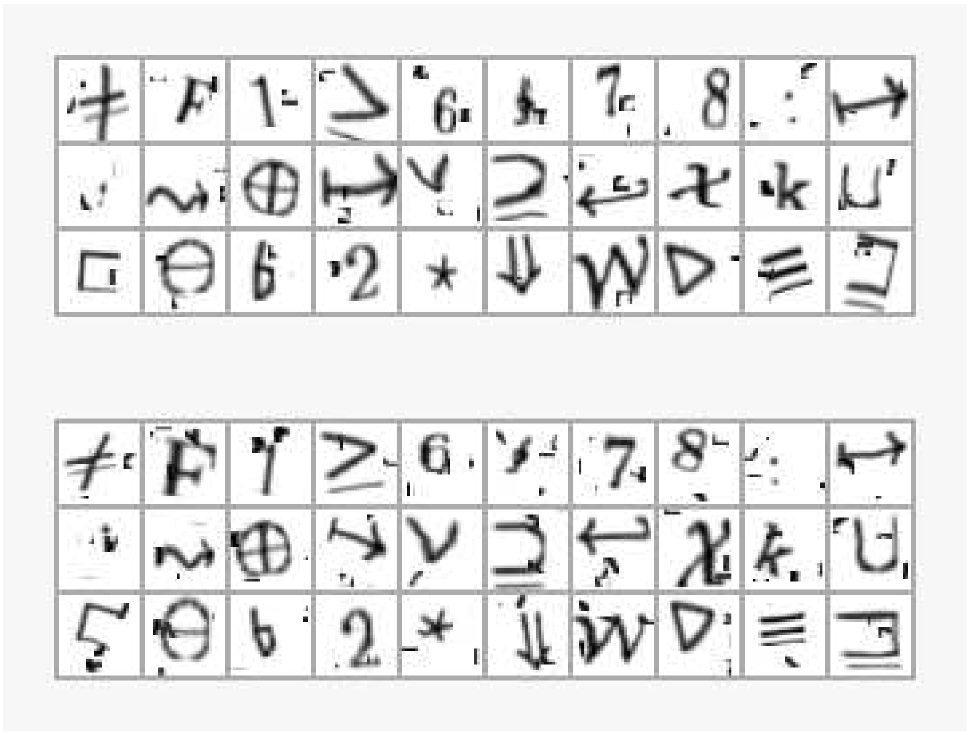

Figure 9.10: LaTeX symbols with clutter. Top: 2 clutter elements. Bottom: 4 clutter elements.

robust to clutter. We could try to fit each of these five models and choose the one with highest value of the cost function given in equation 4.12. Trying all 293 models would be computationally very intensive, but trying 5 is reasonable. Moreover it may not be necessary to try them at each classified data point, only for those where there is evidence for ambiguity.

9.5 Why multiple trees work?

In section 9.1 we motivated the aggregation of multiple trees with an ideal situation where the trees T_n, considered as random variables, are conditionally independent given the class label. Intuitively this is related to the observation made later that different trees classify the data using different aspects of the data, or different points of view. If we *know* the label of a data point ω, then observing $T_n(\omega)$, say in terms of the terminal distribution $\mu_{T_n(\omega)}$, will not help us predict the terminal distribution at the terminal node $T_m(\omega)$ encountered by ω in another tree. This is because the two trees are using different aspect of shape to classify ω and therefore the confusions, reflected in the terminal distribution, will probably be different. This is only a conditional property. If we *do not know* the class label of ω ahead of time, than the mode class in $T_n(\omega)$ would serve as a good prediction for the mode class in $T_m(\omega)$. Otherwise put, these trees are in no way *unconditionally independent*.

The trees are never really conditionally independent. Rather one observes that the conditional covariances between these trees, computed in terms of the terminal distributions μ_{T_n}, are much smaller than the conditional variances. Let ν denote a bound on the conditional variances, and let γ denote some bound on *all* the conditional covariances. We denote the conditional variance given $Y = c$ as $Var_c()$, and the conditional

	Relational $\mathbf{Q} = 200$ $K_w = 16$	Absolute $\mathbf{Q} = 200$ $\mathbf{s} = 15$
$c = 0$	95% (99.6%)	96.6% (99.8%)
$c = 2$	78.4% (91.4%)	62.5% (80.4%)
$c = 4$	57.8% (73.8%)	44.7% (63.8%)

Table 9.9: LaTeX : Performance results in the presence of clutter for relational arrangements and absolute arrangements. The values in parantheses denote the percentage of test points which had the correct class among the top 5.

covariance as $Covar_c()$. The conditional variance of the aggregate $A_N(d)$ is expand as follows.

$$\begin{aligned} Var_c(A_N(d)) &= \frac{1}{N^2}\sum_{n=1}^{N} Var_c \mu_{T_n}(d) + \frac{1}{N(N-1)}\sum_{n\neq m} Covar_c(\mu_{T_n}(d), \mu_{T_m}(d)) \\ &< \nu/N + \gamma, \text{ for } d = 1,\ldots,K. \end{aligned} \tag{9.7}$$

Thus when the number of trees is large the conditional variance of the aggregate is dominated by γ.

For each class c write the conditional mean of the aggregate as

$$\theta_c^{(N)}(d) = E_c(A_N(d)) = \frac{1}{N}\sum_{n=1}^{N} E_c(\mu_{T_n}(d)), \; d = 1, \ldots K.$$

If the individual trees have reasonable classification rates, then the mean weight on class c, given $Y = c$ will be larger than the mean weight on all other classes, i.e.

$$\theta_c^{(N)}(c) - \theta_c^{(N)}(d) > M^{(N)} > 0, \text{ for all } d \neq c,$$

for some constant $M^{(N)}$.

If $M^{(N)}$ is significantly larger than the standard deviation of each of the variables $A_N(d)$ for $d = 1, \ldots, K$, restricted to class c, i.e. if $M^{(N)} >> \sqrt{\gamma}$, then $A_N(c)$, with high probability, will be larger than $A_N(d)$ for all $d \neq c$ and hence the probability of error is small. We see that the key to low error rates with multiple trees is having γ small relative to $M^{(N)}$. By contrast the key to low error rates with one tree is having ν small relative to $M^{(1)}$, which is approximately the same as $M^{(N)}$, since it is an average of means. However in all LaTeX experiments ν is 10 to 20 times larger than γ.

The expectations and variances used in this discussion are never known, at best they can be estimated from training data. It is, however, interesting to observe that there is a high correlation between the quantities γ and M *estimated from training data* and the ultimate error rates on *test data*. We implemented 19 different protocols for the NIST database with absolute arrangements, with 5000 training data (500 per class), varying the stopping rule, the number of random questions sampled, with and without boosting. The log of the *test* error rate $\log(e)$ is then regressed on $\log(\gamma)$ and $\log(M)$. Here instead of upper bounds we redefine γ and M as follows.

$$\begin{aligned} \gamma &= \frac{1}{K}\sum_{c=1}^{K}\sum_{d=1}^{K} Var_c(A_N(d)), \\ M &= \frac{1}{K}\sum_{c=1}^{K}\big(\theta_c(c) - argmax_{d\neq c}\theta_c(d)\big). \end{aligned}$$

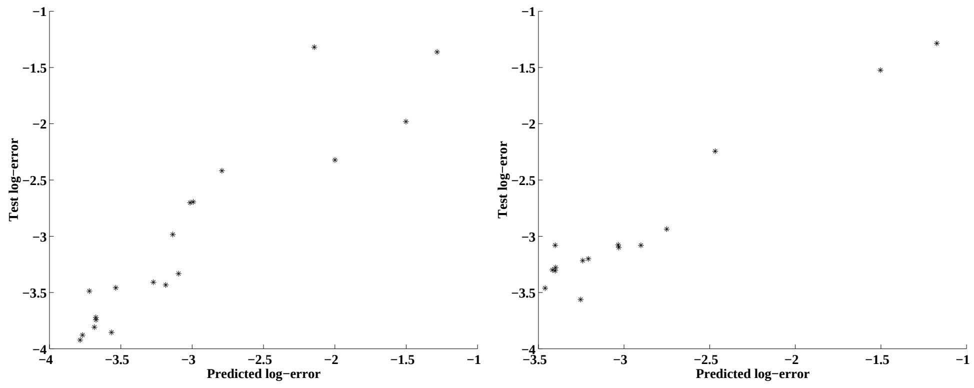

Figure 9.11: Scatter plot of log-error vs. predicted log-error for a variety of protocols. Left: NIST dataset. Right: LATEX dataset.

These quantities are all estimated from *training data*.

We obtain $R^2 = .92$ and the regression equation is $\log(e) = .32 - 1.8\log(M) + 1.3log(\gamma)$. The scatter plot of log error versus predicted log-error is given in figure 9.11. Although the information in γ and M is not sufficient for precise prediction of the test error it is quite suitable for deciding which protocol is preferable once the regression estimates are obtained. This decision can be made on the basis of *training data alone*.

Predicting the test error rate directly from the training error rate would be far less stable, in particular for protocols such as boosting which typically quickly achieve close to 0 error rate on the training data, or protocols growing pure trees, which by definition have 0 error rate on the data.

Note that the test classification rates for the different protocols ranged between 73% and 98%. The variable M ranged between .02 and .85 whereas γ ranged between .001 and .04. The best performance 98% was achieved at $M = .58$ and $\gamma = .034$. These values also gave the highest *predicted* classification rate of 97.7%. A similar regression using LATEX experiments yields $R^2 = 97.2$. It is clear from these regressions that the error rate depends on a ratio γ^a/M^b. Indeed, in some cases M can be higher but the error rate is also higher because γ is high as well.

Both randomization and boosting are somehow achieving low values of γ relative to M. For randomization we already discussed the intuition behind this. Different trees, using an enormous pool of features and queries, are accessing the data from different points of view. There is no danger of overfitting because the trees are trained in total disregard of each other.

By contrast, in boosting, the reduction of γ is achieved through the reweighting of the training points which are misclassified. In concentrating on these points the algorithm is actually trying to produce a *negative* conditional covariance between the new classifier and the current aggregate. This general effect can be achieved with a variety of protocols and has very little to do with the specific protocol involved in the original derivation of the boosting algorithm. However boosting still depends on estimates provided by the training set and risks over fitting, as seen in the deeper trees (smaller **m**) reported in table 9.4. On the other hand on the NIST data base with the large training set of 10,000 data points per class, when randomization is used in conjunction with boosting and a conservative stopping rule the best results are achieved. On the LATEX database the best results are also achieved with boosting using somewhat larger numbers of randomized queries.

9.6 Bibliographical notes and discussion

The work in this Chapter is based on Amit & Geman (1997), and Amit et al. (1997), where only relational arrangements were studied. Some of the ideas on shape quantization explored in Amit & Geman (1997) led to the development of the sparse models described in Chapter 8. On the other hand the very large disjunctions used in step I of the counting algorithm motivated the current experiments with absolute arrangements where very large values of the spread parameter **s** is used.

In studying shape recognition we have touched on two research areas. The first is general classification theory, otherwise known as pattern recognition or machine learning. This covers general issues such as types of classifiers, aggregation methods, estimation of test errors etc. The literature on this subject is described in detail in books such as Ripley (1994), Bishop (1995) and the classic book on pattern recognition Duda & Hart (1973). The main reference for classification trees in the statistics literature is Breiman et al. (1984) and in the computer science literature Quinlan (1986). The idea of growing multiple classifiers and aggregating them is more recent and can be found in Kwok & Carter (1990,), Breiman (1994), Breiman (1998), and Schapire, Fruend, Bartlett & Lee (1998). A more detailed analysis of multiple classifier from the point of view of the discussion in section 9.5 can be found in Amit & Blanchard (2001).

The second area of research is more specific to computer vision and involves identifying what are the appropriate predictors to be used in the shape and object classifiers. What functions of the data will have appropriate invariance properties on one hand, and discriminatory power on the other. Descriptions of different types of local features can be found in books on computer vision such as Haralick & Shapiro (1992) and many proposals for local features can be found in recent work such as Wiskott et al. (1997), Malik & Perona (1990). Any form of binary feature can be incorporated in the methodology described in this chapter. The binary aspect is crucial for the 'spreading' operation in the context of absolute arrangements, or for the definition of the relational arrangements. In principle features with continuous outputs could be quantized, each quantile serving as one form of binary feature. Alternatively continuous outputs can be maximized over a neighborhood, as a generalization of ORing. This has been proposed in Riesenhuber & Poggio (1999).

Much work has also gone into identifying functions of the data which are invariant to geometric transformations. In particular various functions on distances and angles between specific points which are entirely invariant to certain groups of linear transformations. An extensive treatment of such methods can be found in Reiss (1993). These methods are based on prior localization of very particular landmarks on the object and do not address the stochastic aspect of images which precludes such precise localization based only on local information. One could view the use of multiple flexible arrangements of robust local features as an attempt to adapt such ideas to a more realistic view of the data. This adaptation however means losing full invariance. The shape recognition methods described above, as the detection algorithms described in earlier chapters, are not fully rotation invariant, nor are they fully invariant to other linear deformations.

Another interesting approach to the classification of characters which attempts to deal explicitly with invariance to linear and non-linear deformations is described in Hastie & Simard (1998), and Simard et al. (2000). The idea is to implement nearest neighbor classification, or approximations to nearest neighbors using a deformation invariant distance. The distance is essentially based on the linearized two dimensional template matching, described in Chapter 5. The distance betwee a sample image and a training example from a given class is measured after the optimal deformation between the two is computed. A very low dimensional space of deformations is used. It is very computationally demanding to do this with respect large numbers of training examples, so the authors desribe a method for creating smaller numbers of prototypes for each class, essentially centroids of the training data of each class, computed using the deformation distance. The method still relies on preprocessing of the data into fixed size images, and uniform gray scale ranges.

The most succesful hand written character recognition reported to date can be found in LeCun et al. (1998). The classification tool used are multi-layer feed forward neural networks based on the raw pixel intensity input. Although the authors emphasize the fact that very little is done in terms of design of features, and that everything is left to the training procedure, the actual architecture and constraints imposed on the network are of primary importance and interest. Due to the enforcing of translation invariance of the network weights at certain layers, and blurring and subsampling in other layers, the network ends up producing a sequence of feature detectors which are then in some sense 'OR'ed to allow for local invariances. The layers involving translation invariant weights are called *convolution nets.* If instead of clasification trees we had used a feed-forward network based on conjunctions of edges, the resulting network would be qualitatively very similar to the one reported in LeCun et al. (1998). The difference being that we would have hand designed the input level weights defining the input convolution network, as well as the weights defining the second layer convolution network. We revisit this issue in Chapter 11 where we design a biologically plausible network for detection and recognition. The classification rate reported in LeCun et al. (1998) 99.2%. This is a more powerful result than the one reported here because it involves testing on the additional part of the NIST dataset produced by high-school students and exhibiting a larger degree of variability. An additional aspect of the work described in LeCun et al. (1998) is the parsing of character strings, i.e. zip-codes. We will discuss that aspect in Chapter 10.

Two aspects are clearly missing from the experiments described above, which could easily be incorporated in the same framework, both of which may also help with the sensitivity to clutter encountered earlier. The first is the use of features from more than one resolution. It is possible to extract edges from the data both at the original resolution and at lower resolutions and use them all in growing the trees. The second aspect which has been explored in the literature is the use of 'parts', see Shapiro (1980), Brooks (1981), and Haralick & Shapiro (1992) for an extensive review. Usually parts are given a very clear semantic meaning such as 'loop' or 'crossing' or 'ending', or a generalized cylinder in 3d. One expects to find a loop in an 8, but not in a 4. However the description of a loop would need to be very flexible if it were to 'hit' all possible 8's, and then probably loops would be found in many other places where they are not expected, including in some 4's. However if we view loops just as another generic object being detected as described in Chapter 8, then it becomes a feature detected at certain locations. It would have certain statistics on the different classes which would affect its use in the classification trees. It is a more global feature than the ones used up to now and perhaps bears more discriminating power. We will see some pointers to this in the next Chapter. Fully incorporating features which are more global and involve a prior detection process into classification appears to be a promising research direction.

Chapter 10

Scene analysis: merging detection and recognition

The object detection algorithms described in earlier chapters are dedicated to finding instances of a particular predefined object in a scene. On the other hand the input to the classification trees in Chapter 9 was an image in which the object is more or less centered and not much else is present. What should be done when faced with a scene containing several tens from a collection of hundreds or thousands of objects? If for each object we had a very accurate detector with extremely low false positive and false negative rates, we could in principle run each such model and obtain a labeling of all the objects in the image. However this would undoubtedly be a very inefficient approach.

As discussed in the Introduction the prevailing paradigm for analyzing complex scenes assumes an initial 'bottom-up' stage of segmentation which does not employ any prior information on the object class or classes. This stage provides candidate regions of objects in the scene which should in principle include the object and not much more. The regions would subsequently be fed into a classifier.

We will explore a different approach in which the initial processing stage always involves detection of a sparse model. However the detector is now designed to detect more than one particular class, it is a detector for a *cluster* of objects which may contain subsets of some classes and the entire population of other classes. The detection provides an estimate of pose with which data in a region of interest in the image can be registered to the reference grid. This registered data is then classified into one of the several classes in the object cluster. Even though the detector we use is less specific, it still selects only a small number of candidate poses for further processing, determined by the shape information encoded in the model. The advantage over bottom-up processing is that even if parts of the object boundaries are ambiguous, say the object is partially covered by another, and segmentation merges the two into one region, or a lighting effect creates an artificial boundary inside the object, the detector can still identify an approximate pose for the target object and thus provide reasonable input into the classifier. Another advantage is the following. The detection algorithm uses complex features such as edge arrangements which are rare in the background. If these are the local features being registered to the reference grid and used to classify there is much less clutter in the neighborhood of the object.

In the next section we illustrate these ideas in the context of detection and recognition among three different chess pieces in a real image. Following that we explore detection and recognition in scenes produced with symbols, both artificial LaTeX scenes and zip-codes. Finally in section 10.3 we discuss possible strategies for creating object clusters in a more systematic way.

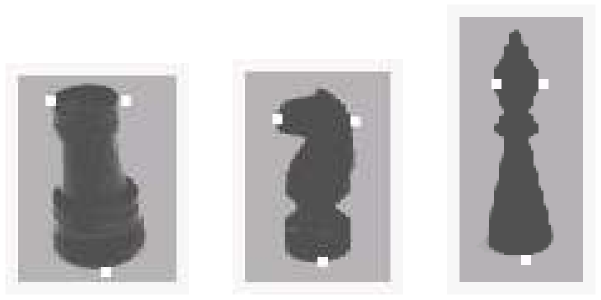

Figure 10.1: The prototype image of three chess pieces. Overlayed on the prototype are the three pose reference points used to register the object to the reference grid.

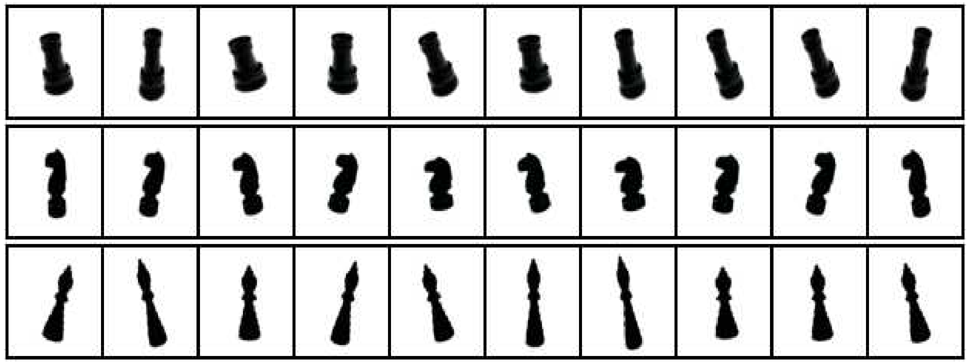

Figure 10.2: Samples of randomly perturbed prototypes of the three objects

10.1 Classification of chess pieces in gray level images

We study a very limited problem of detecting and recognizing among three different chess pieces: a knight, a rook and a queen, all black. The difference between the queen, the bishop and the king at the resolution of images we acquire was too small to obtain reasonable discrimination. One prototype image is acquired on a flat background for each of the three objects as shown in figure 10.1.

Each of the prototypes is perturbed with a sequence of 100 random affine maps to produce a database for training a sparse model and a set of classification trees. In terms of the production of the training set, this is the very same procedure used for the LaTeX data in the previous chapter as well as for the clip detector: one prototype is used to produce a small sample for training. Samples of each are show in figure 10.2. The coordinates of the three pose reference points chosen on the prototypes are mapped according to the random affine map to produce three anchor points for each of the perturbed images. Edges are extracted from each of the 300 perturbed images, 100 per class, and re-registered to the reference grid. This data is then used to train a sparse model for the *union* of the three classes. We use features with three edges, a center edge and two additional edges, ($n_r = 2$,) requiring each feature to be present in at least 50% of the data. These features will be used for classification as well so we take as many as are found, i.e. one for each 3×3 region in the reference grid, yielding 30 features. The threshold τ, for the number of hits to declare a detection, was set to 17, by finding the largest value with no false negatives on the training data. The local features obtained in the model are shown in figure 10.3. The model we observe is some combination of common elements we would obtain from producing separate models for the three different pieces. In figure 10.4 we

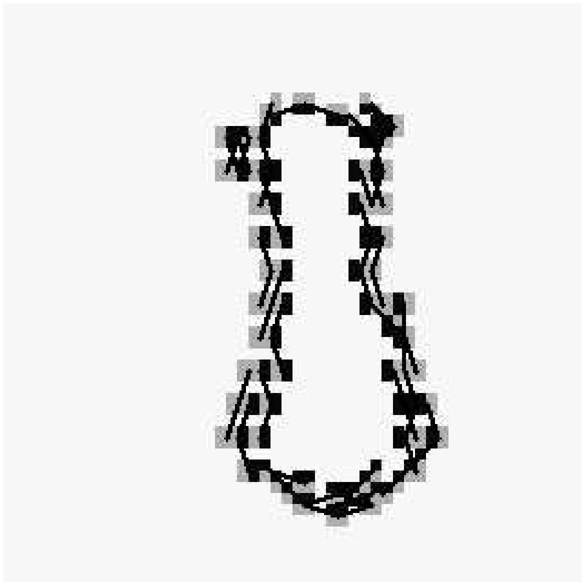

Figure 10.3: The 42 model features obtained for the combined class of rooks, knights and queens

show several detections of this model on training data from the 3 classes. Note in figure 10.4 that due to the low threshold a given instance of an object may be hit with several detections yielding several different poses. We need to keep this in mind when training the classification trees.

The detector is run on the training data of the three classes. For every training image we register the feature data to the reference grid, for *each* detection obtained in terms of the detected pose, using equation 8.2. In this case the data consists of all instances of the 30 features used for the detection. Thus if a detection hits the upper part of the shape the registered data concentrates on the lower part of the reference grid, and vice-versa if the lower part of the shape is hit. This is illustrated in figure 10.5 where the registered locations of 2 of the 30 features are shown for the detections on the rook in figure 10.4. Note that we have constrained the reference grid to a 60 by 30 window around the three reference points. This allows us to limit the clutter to the close neighborhood of the detection. The labeled and registered local feature maps obtained from training data are used to produce fifty randomized decision trees using absolute arrangements and a spreading parameter $\mathbf{s} = 5$.

In figure 10.6 we show a detection and classification on an image with a rook. The rook is placed on a wood texture which responds to the edge detectors in many locations. Despite the fact that to our eyes the rook appears perfectly well separated from the background, in terms of the initial edge input to the algorithm, there is significant clutter in the neighborhood of the object. The left hand column of 10.6 shows the locations of horizontal and vertical edges of the two polarities. The lower three panels of the right hand column show the locations of three of the edge arrangements in the model. The density is significantly lower. It is possible to employ other complex local features for classification, however from the computational point of view it is very convenient to use features which have already been detected.

In figure 10.7 we show registered local features (0,4,12) from the region of interest around the detection. Despite the fact that these features are less frequent they still produce noise in the background which can potentially confuse the classifier. Constraining the reference grid to a narrow window around the three pose reference points helps but can not entirely solve the problem, see for example the registered data for feature 0. Moreover constraining the window too far will lead to a loss of important information for classification. In

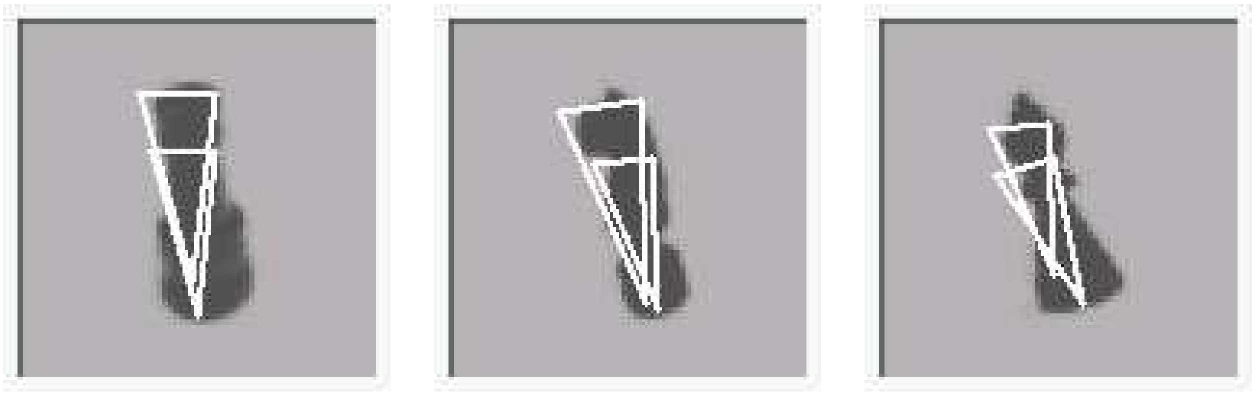

Figure 10.4: Hits of the detector on a sample rook, knight and queen

figures 10.8, 10.9 we show some additional examples of classified detections, including some misclassifications.

First note that classification is invariant to a range of variations in scale, rotation and other transformations occurring due to the change of the position of the object relative to the camera. The reason for the misclassifications are usually quite apparent. For example in many cases if some long dark vertical object is standing behind the rook it will be misclassified as a queen which has no horizontal structure at the top. If a dark structure is present near the upper left hand part of the rook it will be misclassified as a knight. In addition there are false positives which do not correspond to any of the three pieces. The human eye is rarely fooled by such problems. This is probably due to two factors. At the low level the visual system probably employs a richer and more powerful collection of local features not solely based on edges. At the high level we are probably able to *simultaneously* classify other objects in the immediate neighborhood of the piece and thereby provide more convincing explanations of the data.

Further analysis of the output of the classifier is always possible. One of the deformable models for the identified class can be fit to the data in the region of interest in order to obtain more detailed instantiation information. Furthermore an upper bound on the fit of each model to examples of the correct class can be estimated from training data. Any fit above this bound is rejected. We have not implemented such a step in the current experiments.

This form of processing can also be used to solve ambiguities. Note that the output of the multiple classification trees consists of a weight on each of the classes. If other classes have weight close to that of the mode it may be of use to check each of these candidates in terms of a more detailed model.

Some degree of rotation invariance in detection and classification is apparent, but these are limited and the algorithm will fail with significant rotations in the plane of view, or rotations around the vertical axis of a piece, such as the knight, which is not rotationally symmetric. In the present context, the opposite orientation of the knight would have to be treated as a separate object in the collection.

10.2 Detecting and classifying characters

LaTeX scenes

In this section we report similar experiments with 62 LaTeX symbols (10 digits, 26 upper case and 26 lower case letters,) and handwritten zip-codes. The only difference with the previous section is that we use a sparse model trained on a particular class, in this case the symbol ‘0’, as opposed to the entire collection

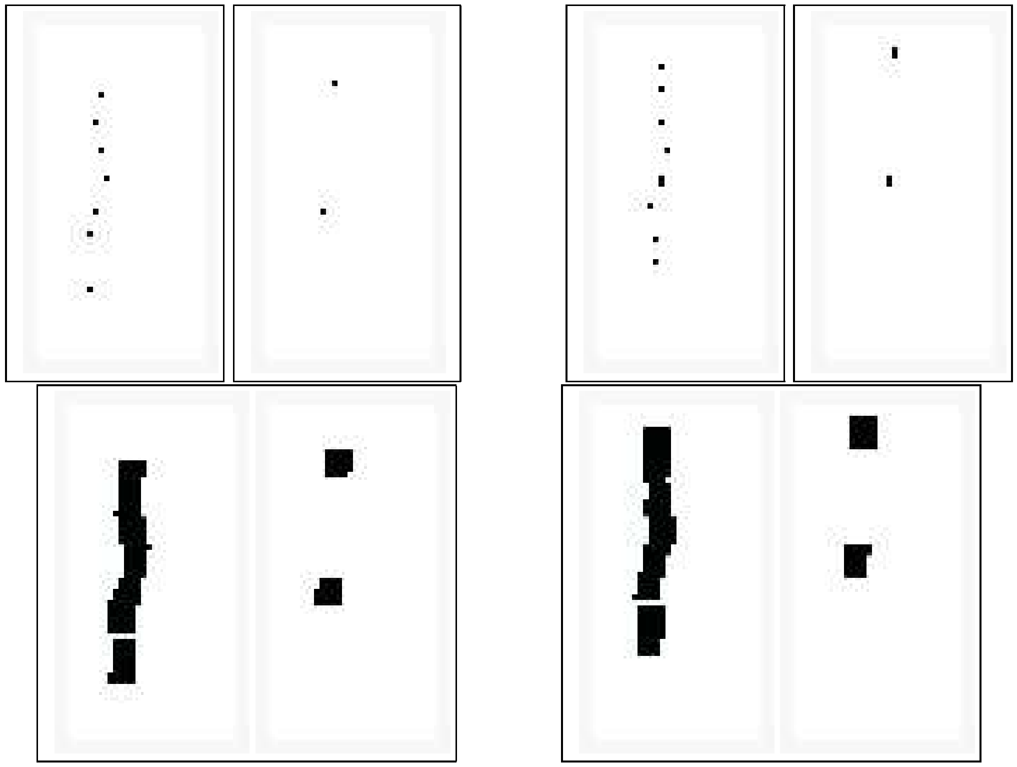

Figure 10.5: Top: Registered local features 0 and 4 for the two detections on the rook in figure 10.4. Bottom: The data after spreading the features in 5×5 neighborhoods prior to classification.

of classes. The motivation here is that training with the union of all the classes would result in a very non-specific model with numerous false positives. There is not much in common in terms of the shapes of all 62 symbols. One alternative is to train for one class and then resolve the ambiguities due to false positives using a classifier. The sparse model for the '0' is trained with 32 random samples of the '0' class. From this sample 32 local edge arrangements were found of complexity $n_r = 3$. The threshold $\tau = 24$ is needed to keep all training samples of the '0'. This detector is then run on examples from all 62 classes (32 of each) with a threshold $.8 * \tau = 19$. This is quite a loose threshold permitting quite a number of detections on other classes. Detections are registered to the reference grid, and multiple classification trees are trained, using the local features from the '0' model. In the first histogram in figure 10.10 we show the distribution over the 62 classes, of the number of training examples (out of 32) hit by the '0' detector.

In figure 10.11 we show results on some artificial LaTeX scenes. The entire procedure of detection and classification on images with approximately 30 symbols takes on the order of 250 milliseconds on the PENTIUM III 700Mhz. There are more detections present then shown in the image. Detections are clustered according to the procedure outlined in Chapter 8. However now, to represent the cluster, we no longer choose the detection with largest number of detected local features, rather the choice depends on the output of the classifier. Recall that classification is performed with 50 randomized trees which are aggregated as in equation 9.4. Each detection in the cluster is classified according to the mode of the aggregate. We choose the detection with highest weight at the mode. This in some sense is the detection in the cluster which is classified with most 'confidence'. Since every detection is classified prior to clustering, there is more information available which is not used in the present context. For example the actual weights on the different

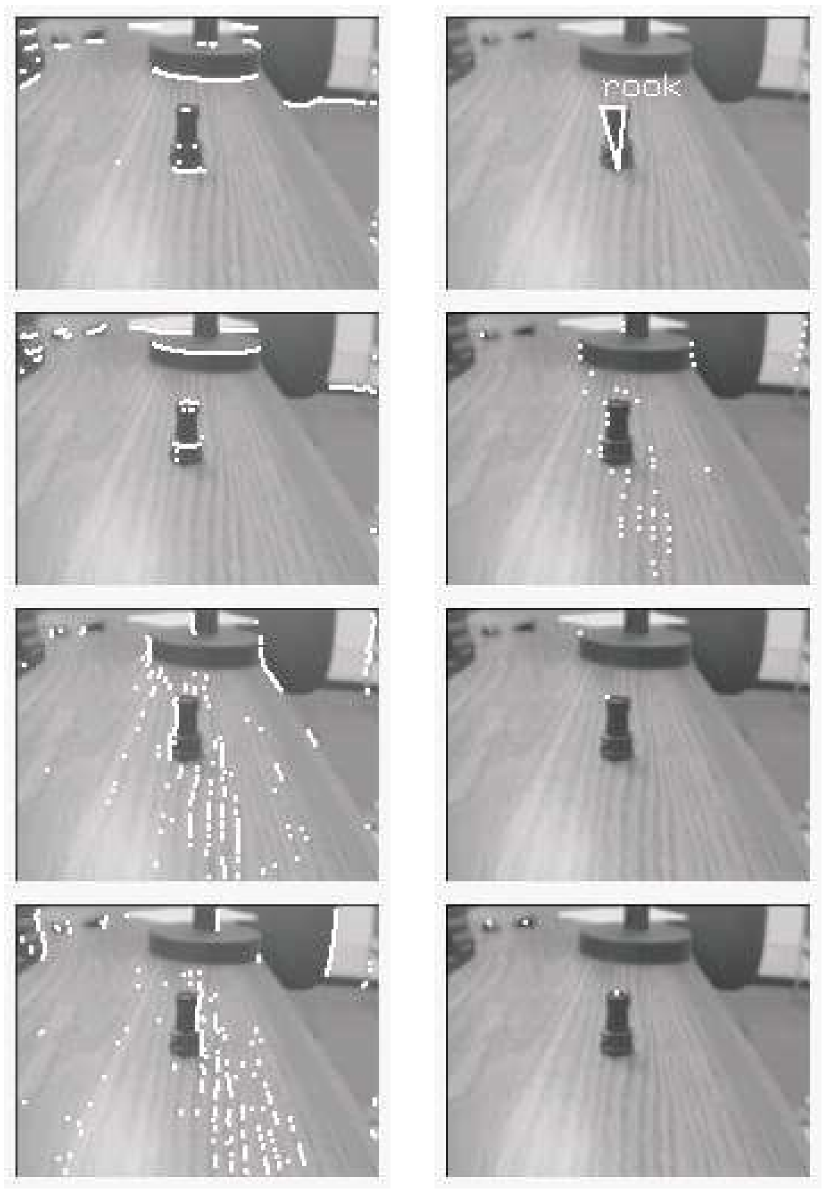

Figure 10.6: Right column: Locations of horizontal and vertical edges of two polarities in the image. Left column: Top: classified detection. Locations of local features 0,4,12 of the sparse model in the image.

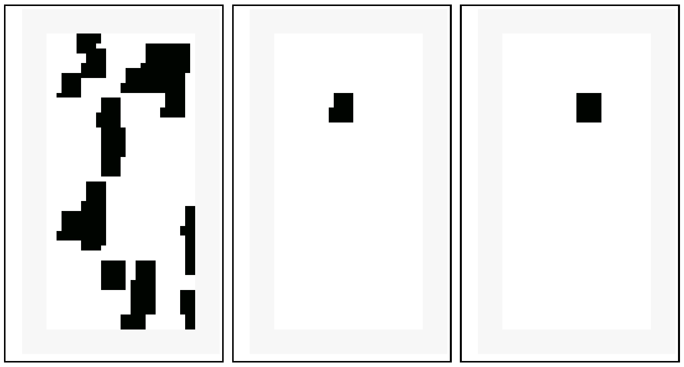

Figure 10.7: Registered local features 0,4,and 12 on the reference grid from around the detection shown in figure .

classes provided by the aggregates, or the class labels assigned to other detections in the cluster. All these could help disambiguate confusions and reduce the number of errors.

Zipcodes

For the zip-code experiment, a ‘0’ detector is trained on 1000 zeros from the NIST dataset. Due to an erroneous setting of the scaling of the training set, the detector was produced for a ‘0’ half the size of the standard 32×32. This detector is therefore hitting loops and parts of loops in the larger images. The detector is run on a subset of 20000 images from the NIST dataset, from all classes, to produce the data for training the classification trees. In this case the edges and not the edge arrangements were used for classification. Twenty randomized classification trees were trained. Some results of detection and classification on zip-code images are shown in figure 10.12. This does not represent the final analysis of a zip-code, merely the classification of the detections of the ‘0’ detector.

In some of these images one could have easily implemented a prior segmentation using some simple principle. This is not always the case, and any bottom-up segmentation of zip-codes must be able to identify the misleading cases. One example is shown in figure 10.13. Here some digits are joined together. We purposefully show all the detections in this image with no clustering to illustrate the fact that there is information to be obtained from all detections not just those which end up representing a cluster.

There are still plenty of errors in these initial detection and recognition experiments. Less so if tested on isolated digits. For example the same procedure was applied to 10000 USPS digits. On each USPS image all detections are classified and if any one of them is correct the digit is counted as correctly classified. Of the 10000 images there was a detection in 7416 and of these 93% were correctly classified in this manner. The large number of detections of the ‘0’ detector is due to the fact that the model was trained for ‘0’s half the standard size. With the lower threshold any part of a loop is detected, for example the upper part of a ‘4’. It appears that we have used something between a global model for a particular object, and a local feature. The ‘0’ detector trained on the smaller scale can be viewed as a generic part that appears in many classes, however when detected this part helps determine the location and scale of the object, as opposed to a local feature which in itself can only loosely indicate location.

Bear in mind that these results are produced through direct implementation of the detection and classification procedures. Both the detection and classification models are trained on relatively small samples of the NIST and are being tested on the more variable USPS database. Thus their appears to be some promise in such a top-down approach in which scene analysis is guided by some form of sparse object model which facilitates useful groupings of local features. It is important to recall that no prior segmentation is performed to identify objects, and the computation time is on the order of a second or less.

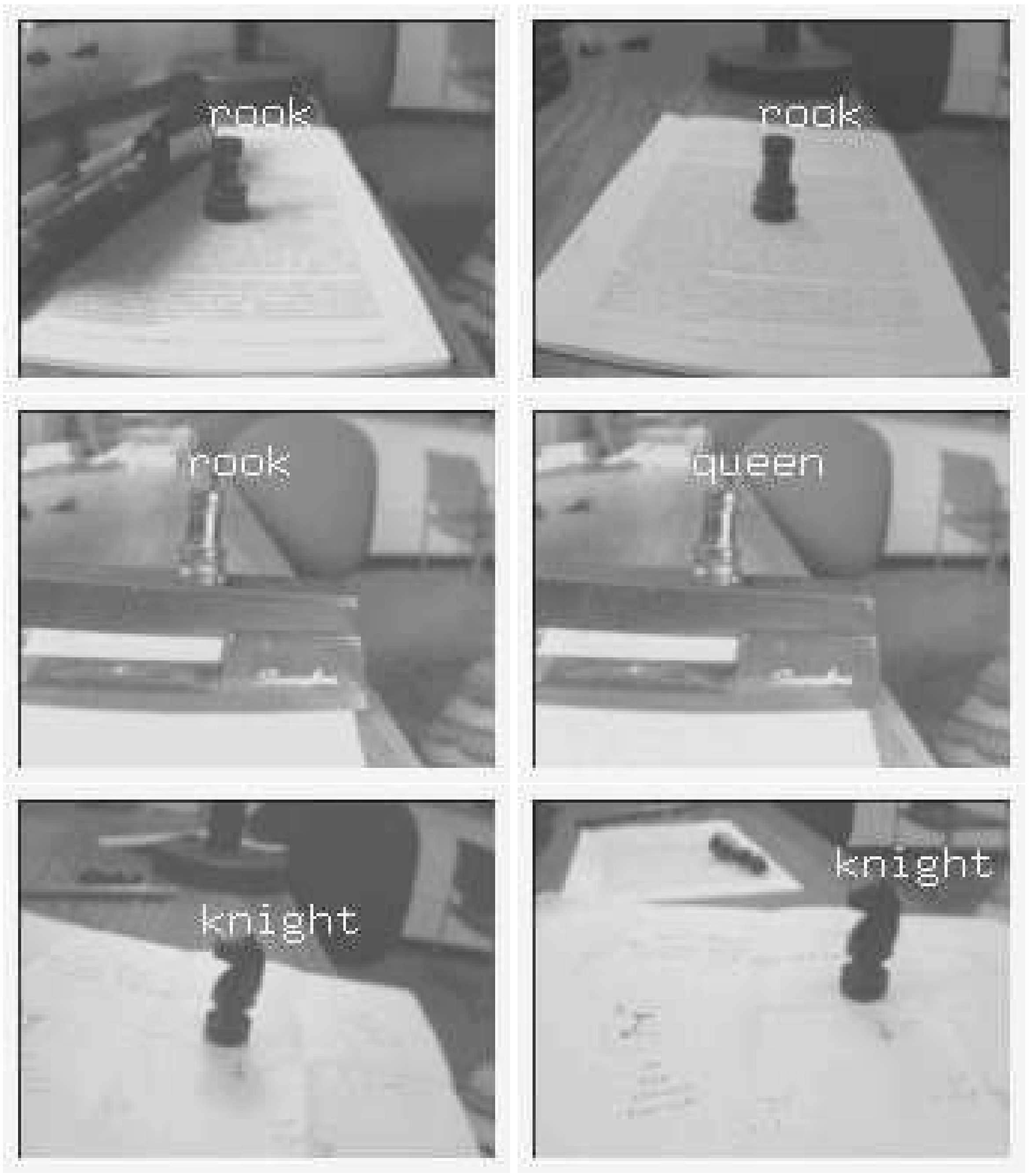

Figure 10.8: Example of detections and their classification.

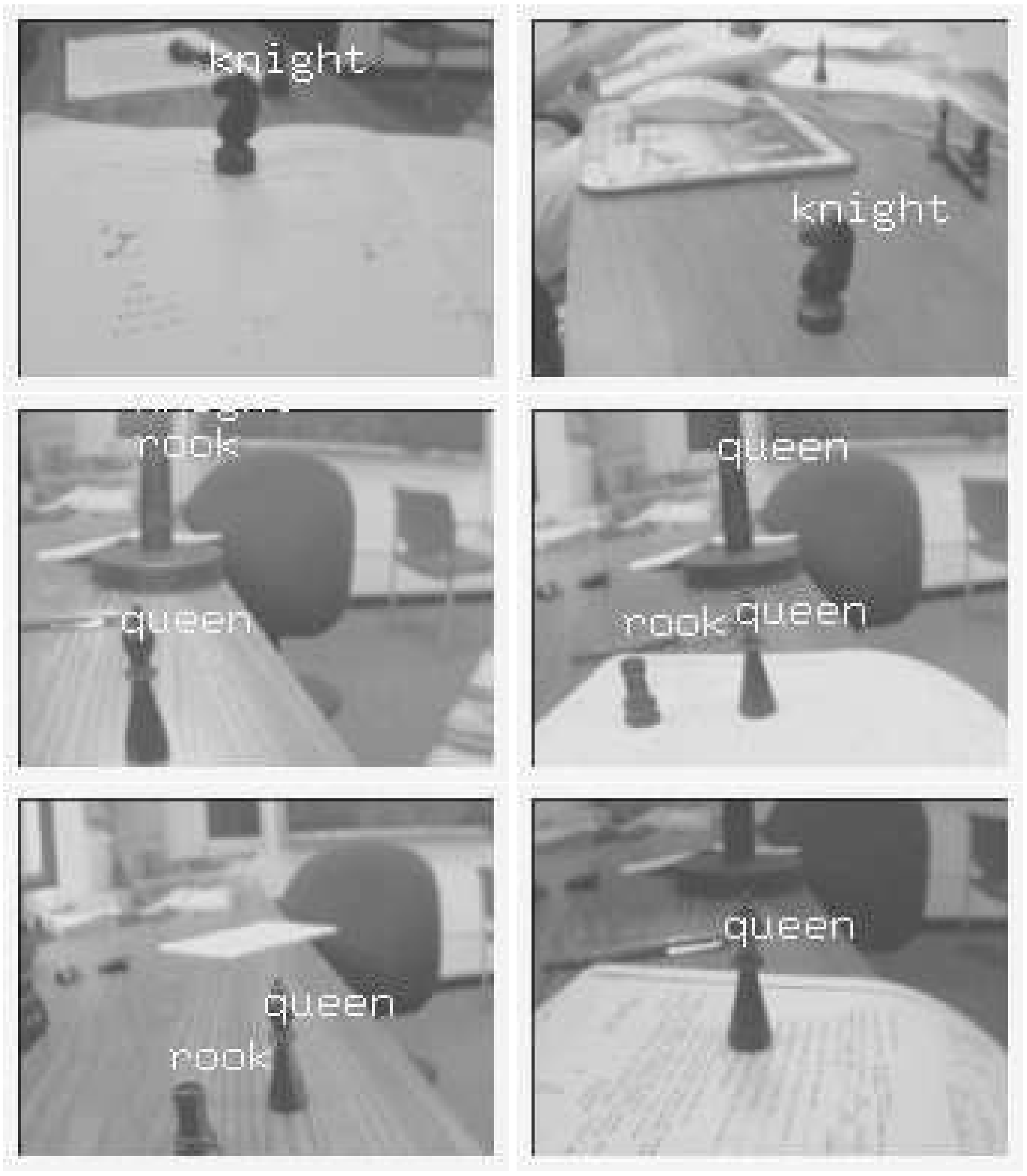

Figure 10.9: Example of detections and their classification.

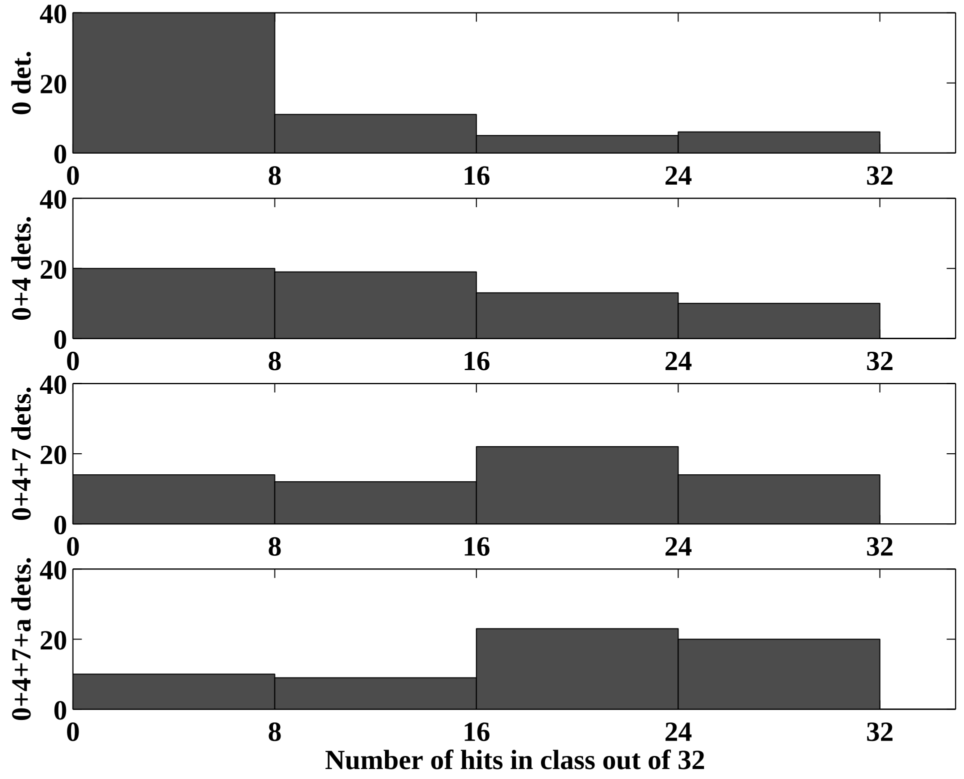

Figure 10.10: The histograms show the distribution over the 62 classes of the number of samples (out of 32) hit by the corresponding union of detectors. For example with the '0' detector alone only six classes are hit at above 28 samples out of the 32. With all 4 detectors 20 classes are hit at above 28 samples out of the 32.

10.3 Object clustering

A fundamental question arises in this context. How should the sparse detection models be constructed so that on one hand all elements of all classes are detected by at least one detector, and on the other hand the number of detectors and the number of false positives on *generic background* is minimized. This can be formulated as a clustering problem. We want to form object clusters which cover the entire population, minimizing the number of clusters on one hand, and the 'diameter' of the clusters on the other. Here the term 'diameter' is used figuratively. Clusters with a large 'diameter' will yield detectors with many hits and false positives. Note that there is no particular need for the clusters to be disjoint. However it is obviously wasteful to detect the same object class multiple times.

Another question is how to produce the final scene interpretation. As we have seen the detectors hit different parts of the object and the same object can be classified in several ways. Some classified detections will be inconsistent as can be observed for example in the top left zip-code image. There needs to be a clearly defined mechanism to generate candidate *scene interpretations* from the set of classified detections, and to rank them according to some cost function. An interpretation involves making decisions regarding inconsistent classifications, deciding which objects are occluded etc. This also requires deciding what are exactly the 'on-object' pixels for a particular classified detection and is closely related to final processing perhaps using some of the deformation algorithms described in earlier chapters. Solutions to the second

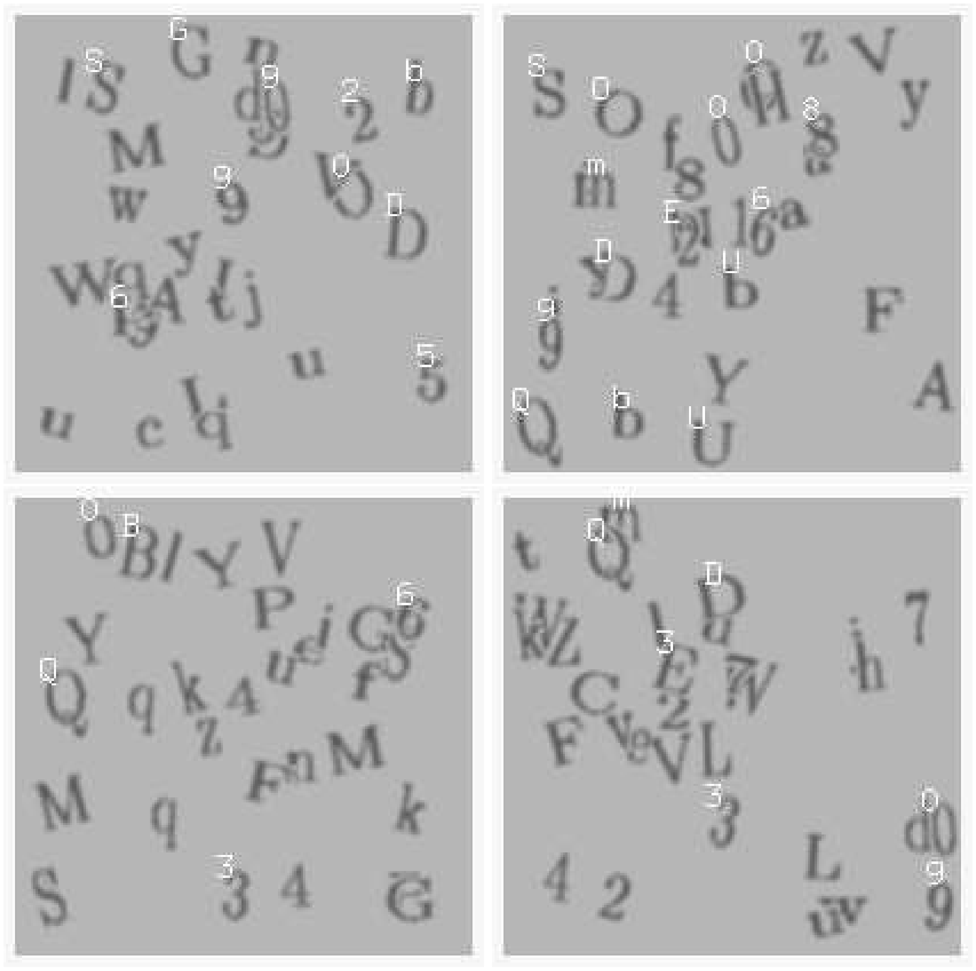

Figure 10.11: Synthetic LATEX scenes with classified detections of the '0' detector.

question of scene interpretation are beyond the scope of this book, although hopefully we have presented an array of tools which can contribute towards the solution. In the discussion section we briefly mention some scene interpretation methods dedicated to 'linear' scenes such as zipcodes and other images of text.

In the next section we offer some suggestions for dealing with the problem of object clustering. We describe two possible approaches to object clustering. In the first we do not assume all classes are presented from the start. Classes are learned sequentially. In the second approach we assume all example ares present at the initial stage and clusters are created using classification trees.

Sequential object clustering

Classes are presented sequentially in some predetermined order. A sparse model for the first class is trained and the detection algorithm corresponding to this model is applied to samples from a number of new object classes as demonstrated in the previous section. Those data points from the training set which are hit form a cluster. The cluster is entirely defined in terms of the detector and the threshold used. A lower threshold

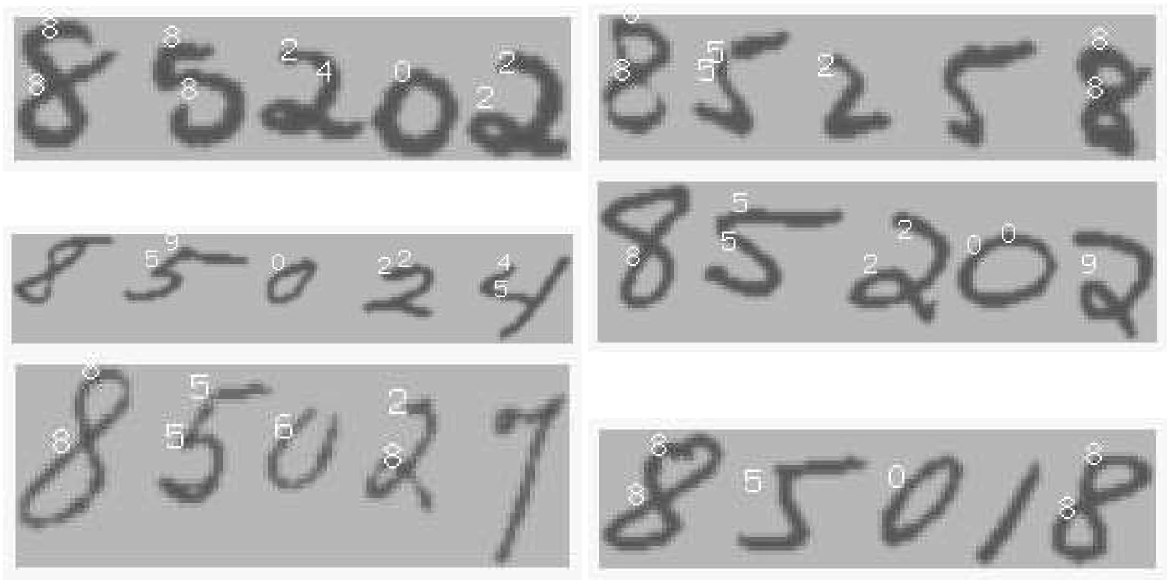

Figure 10.12: Detection and classification on zip-codes

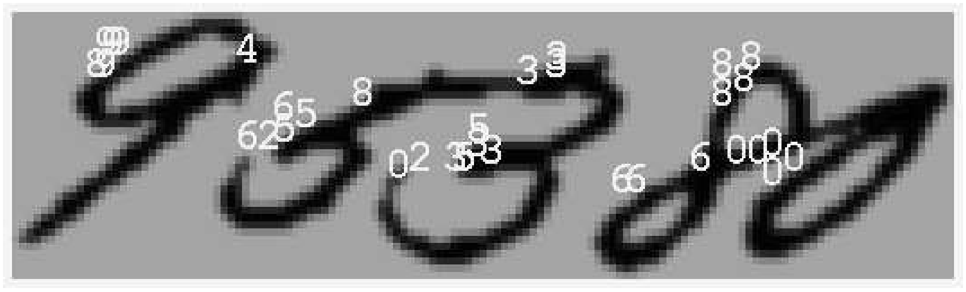

Figure 10.13: All classified detections on a zip-code with no clustering

produces a larger cluster. The training data can be be used to estimate the conditional probabilities on class within this cluster.

Some classes will be fully detected and need not be addressed any more. The original '0' detector serves as their detector as well. Other classes are partially detected and will need an additional detector to fully cover all samples. One approach is to produce a detector for a class which is poorly detected by the '0', for example the '4' (we skip the '1' because it has very little structure). Run the identical procedure with this detector. Identify the training data in the new cluster and produce appropriate classifiers. There will be overlaps between the two clusters, meaning that some data will be hit by the two detectors and may even be classified in two different ways. In the second histogram in figure 10.10 we show the distribution over the 62 classes, of the number of training examples (out of 32) hit by the '0' *or* the '4' detector, namely the union of the two clusters. Some progress has been made in terms of covering the entire population. The third and fourth histograms shows the same information for the union of the '0', '4', '7' detectors and of the '0', '4', '7' and 'a' detectors.

One hopes that the number of models needed to cover the entire population grows much slower than the number of classes. In the context of view-based 3d object detection and recognition each object is represented by a number of 2d models, namely each 3d object generates a number of 2d classes, corresponding to different viewpoints. If the number of models grows linearly with the number of 2d classes we are headed towards an explosion in the number of required models. Hopefully different views of different objects will merge into common clusters and this explosion can be avoided.

This clustering approach does not require having all the training data for all the classes available ahead of time, and can be viewed as a crude way to mimic sequential learning of new classes. However it is very redundant in that many data points will be hit by more than one detector. In other words the clusters are far from disjoint.

Tree based class clustering

An alternative approach to creating the clusters is using the classification tree machinery introduced in Chapter 9. We start with a simplifying assumption that all detectors are produced with a fixed set of feature types, such as for example all two-edge arrangements such as those defined in Chapter 9. Grow a classification tree, or an unsupervised tree, ignoring class labels (see section 9.1, of moderate depth on the training data, using absolute arrangements, as described in Chapter 9. Each terminal node of the tree defines an object cluster. These are images of various classes which have answered the same way to a sequence of queries and hence necessarily have certain similarities in terms of shape. They all carry the same absolute arrangement. In fact the queries along the path to the terminal node can serve as *part* of the model which is subsequently derived for the cluster. More than one tree can be used to obtain a more robust covering of the population.

Once a model is determined for the data at a terminal node, a classifier is trained to discriminate between the classes in the cluster. The detector is applied to each of the training images, the labeled data is registered to the reference grid in terms of the detected pose, and then multiple randomized classification trees are used to produce a classifier. Note that in this context the depth of the clustering tree determines the number of clusters. By definition the set of terminal nodes of the tree covers the entire training set. This is in contrast to the previous clustering method where there was no way to predetermine the number of clusters. The deeper the tree the larger the number of clusters on one hand, but then the associated models are more specific and one would expect a smaller number of false positives from this collection of models. The hard question is what is the appropriate balance between these two variables.

It is interesting to note that this approach closes the loop between sparse models and classification trees based on arrangements of local features. A detection model is based on a cluster defined by a classification tree, and the registered data around a detection is subsequently classified with additional classification trees.

10.4 Bibliographical notes and discussion

In this chapter no model is presented for the layout of the objects in the scene or their number. Typically there is more contextual information which can be exploited. Zip-codes are a very clear example where there is a fixed number of objects in the scene, and there are very clear constraints on how the objects are arranged. Given a set of multiple trees trained to classify among the digits, any candidate slice of the zipcode image can be classified, and also assigned a 'confidence level' according to the magnitude of the mode of the aggregate distribution $A_N(c)$ (see equation 9.4.). A range of slices covering some locations and slants is determined for each of the five digits. Dynamic programming based on the confidence levels of the slices finds the optimal set of five slices. The labels assigned to these slices have already been recorded and provide the outcome. This is the outline in the approach taken in Wang (1998). In LeCun et al. (1998) a much more sophisticated version of this idea is implemented. A bottom up over-segmentation of the image is performed, with certain segments possibly containing subsets of the digits. Dynamic programming is now performed on possible individual segments or consecutive pairs. The system is imporved by training classifiers with 'parts' of other digits in the image, as well as training the weighting of the different components of the cost functions of the dynamic programming using labeled zipcodes, where the answer is known. Another interesting idea suggested in this work is using the classifier to determine candidate segments. The classifier is run on the entire scene and locations with high confidence become candidate segment for the dynamic programming. Other ideas on the use of high level knowledge on the layout of the scene has been extensively exploited in the document analysis community, see Nagy (2000).

Typical images which our visual system encounters do not have such clear structure. Humans can quickly and succesfully parse a scene with digits scattered about randomly with various sizeds, shapes and occlusions. It is therefore of interest to pursue the issue of scene analysis using less restrictive models from the 'linear' ones described above.

It is important to reiterate that each detection carries with it much more information than has actually been used. For example the precise locations of those features detected by the detector. So far we have classified by registering *all* features found in the region of interest to the reference grid. It may be that there is useful information in the particular features found by the detector. Other information comes in the form of a match to a deformable model. In Chapter 8 we showed how more detailed models can be matched to the object following detection. Quantities measured in terms of these matches may be useful in pruning out false positives and making final determinations in classification.

The sparse models used here are quite successful in detecting objects with reasonable numbers of false positives. These models make a 'leap' directly from the local features to the global models. With hundreds of possible local features and hundreds of possible locations on the reference grid the number of possible models is enormous. The question is whether a relatively small number of models can cover a large collection of object clases without being too 'trivial' in the sense that these models end up having numerous false positives. In Chapter 8 we mentioned the ideas proposed in Biederman (1995) whereby objects are represented as coarse arrangements of generic parts. It may be that an intermediate level is necessary, involving a rather small number of generic shape models, much like the '0' model used for the zip-codes, which effectively detects small loops. These generic shape models are then used to produce the entire library of object models. The object models could be even coarser than the current sparse models, since the components are more complex. For example they could be defined on a much coarser reference grid. The chance of a particular configuration of these components occurring at random is very small. It is possible that then detection is limited to very simple configurations of these generic shapes. The difficult question is what price is payed in terms of false positives, if these generic components are to be stable on the objects.

Chapter 11

Neural network implementations

There are fundamental differences between the form of computation performed in a serial computer and that performed in a massively parallel system of very primitive computational units. On a computer we can learn a model for faces or any other object using our favorite technique, the model is stored using whatever complex data structures we need, and then by brute force, using the power of random access memory, the model can be applied at every shifted location and scale to decide if an object is present there or not. In a parallel system which mirrors the visual scene, with a unit or multiple units for every pixel location, processing the data in its neighborhood, the question is how to *shift* a model learned and stored in some central location to all other locations in the system.

The dominant paradigm for learning in neural systems is through modification of the value of the connections between pairs of units. If say the connections at the central location of the system have been modified to classify face vs. non-face, we can hardly imagine the values of these connections being shifted in some mysterious way to all other locations of the system. Moreover what happens when we want to detect a clip and not a face. How does the entire parallel system know to change so as to detect clips and not faces.

Turning to classification, assume that the connections at the central location have been modified so that we can classify among the ten handwritten digits. What if a digit appears at a shifted location? The visual system can still recognize the digit without actually directly looking towards it, as long as it is not too far out in the periphery. Again we ask, how does the classifier encoded in the central location get shifted. One possibility is that the classifier is learned separately at each location. The appropriate connections are modified at each location in the system using training data presented at that location. This is not a viable solution since we can very well learn an object model or a classifier at one location and scale and then recognize it anywhere else in the scene.

Both the sparse detection models and the recognition algorithms described in the previous chapters are based on simple computations in terms of simple binary local features. In this chapter these ingredients will be used to construct a parallel system able to overcome the difficulties described above. The system will integrate learning of individual object models for detection, learning of classifiers and the implementation of detection and recognition over the entire visual scene. Neurons and their synaptic connections are modeled at a very simplistic level, avoiding the detailed complexities of their operation, however the entire system does offer a 'global' hypothesis as to how the visual system learns objects and detects them and how different object classes are recognized. After describing this network architecture some analogies to the architecture of the biological visual system will be discussed.

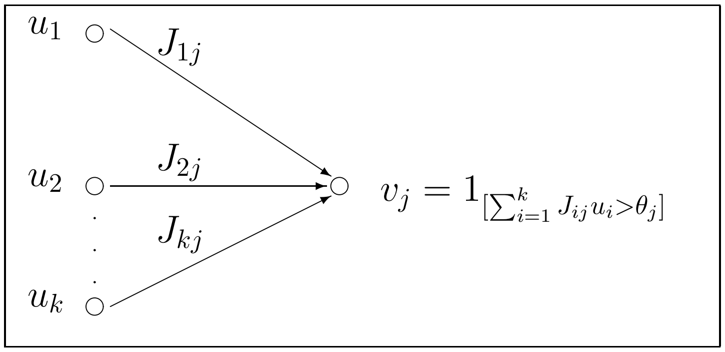

Figure 11.1: k pre-synaptic units feeding into v. The output of all units is binary 0/1. The output of v_j is obtained by taking the thresholded sum of the output of the units u_i.

11.1 Basic network architecture

Neurons and synapses

The building block of the model is a binary neuron v receiving input from binary neurons u_i through directed synapses with efficacies J_{ij}, as illustrated in figure 11.1. The input connections from the neurons u_i are called *afferent* connections of v_j. Each neuron can either be on or off, denoted 1 and 0 respectively, and has an associated threshold θ. The neuron is on if the local field of the neuron, namely the weighted sum of its inputs, is above the threshold θ. Specifically

$$v_j = \begin{cases} 1 & \text{if } \sum_{i=1}^{k} u_i J_{ij} > \theta \\ 0 & \text{Otherwise.} \end{cases} \tag{11.1}$$

The local field h_j of the neuron j is defined as

$$h_j = \sum_{i=1}^{k} u_i J_{ij} \tag{11.2}$$

Layer connections

The network architecture has the form of layers of neurons feeding into other layers. Different neurons can have different thresholds, however in order to maintain the simplicity of the system, neurons of the same layer all have the same threshold. In the material presented below there are no connections within a layer, known as recurrent connections. Such connections are a crucial component of the real neural system and have a very important role to play in stabilizing the system, and enabling consistent memory recovery, but are beyond the scope of this chapter. Material on recurrent mechanisms can be found in Amit (1989), Brunel et al. (1998).

In figure 11.2 we see a schematic diagram of a number of layers with connections between them. The arrows represent synaptic connections and indicate the direction information is flowing. Given two layers A and B we denote by $A{\rightarrow}B$ synapses, those directed from layer A to B. We will also assume that the maximal synaptic efficacy for $A{\rightarrow}B$ synapses is constant at J^{AB}. Input presented at some point in the system, for example layer A in figure 11.2, is assumed to propagate in an orderly fashion from layer to layer one step at a time. This is a gross over simplification since the actual dynamics of a neural system is much more complex and chaotic, with neurons typically firing asynchronously at random times.

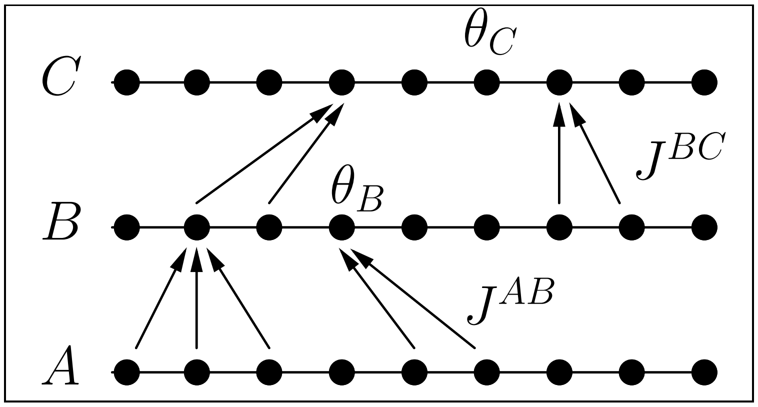

Figure 11.2: A schematic diagram of layers of neurons feeding into other layers. The efficacies of A–B synapses are all fixed at J^{AB}, and efficacies of B–C synapses are all J^{BC}. Neuron thresholds in B are all θ_B and in C are all θ_C.

Visual input: edges and edge arrangements

The visual input layers are organized *retinotopically*, meaning that they form a two dimensional grid corresponding to the entire image lattice L. These layers will consist of coarse oriented edge detectors of the same type described in Chapters 5 and 6. For any retinotopic layer B we write $B(x) = 1/0$ according to whether the unit in B corresponding to location $x \in L$ is on or off. We thus have eight retinotopic layers $E_e, e = 1, \ldots, 8$, where a unit $E_e(x) = 1$ if an edge of type e is found at location x in the image.

The next system of layers will be wired to compute the locations of all $N = 256$ two-edge arrangements consisting of a center edge and one additional edge as described in section 6.4. This particular collection of simple two-edge arrangements was also used as input to the classification trees in Chapter 9. Let $\mathcal{R} = \{R_1, \ldots, R_m\}$ be the collection of regions used to define the edge arrangements, see section 6.4. For each edge type e define a system of retinotopic layers $C_{e,k},\ k = 1, \ldots, m$. A unit $x \in C_{e,k}$ receives input from the region $x + R_k$ in E_e. It is on, i.e. $C_{e,k}(x) = 1$, if any unit in $x + R_k$ is on in E_e, namely if

$$\max_{y \in x+R_k} E_e(y) = 1.$$

Let the synaptic efficacies between layers E and C all be of value J^{EC} and let the threshold of all neurons in C be $\theta_C = J^{EC} - \epsilon$. This implies that input from only *one* pre-synaptic neuron is sufficient to activate the neuron in C.

The two-edge local features are computed in retinotopic layers $F_\alpha, \alpha = 1 \ldots, N$. Each feature α is defined by a triple (e, e', R_k), where e denotes the type of the center edge, e' the second edge which is required to be present somewhere in the region R_k relative to the location of e. Each $x \in F_\alpha$ receives input from $x \in E_e$ and from $x \in C_{e',k}$, and is on only if *both* are on,

$$F_\alpha(x) = \min(E_e(x), C_{e',k}(x)).$$

Let $J^{EF} = J^{CF} = J$, and for the neurons in the F layers set the threshold to $\theta_F = 2J - \epsilon$. In this case two pre-synaptic neurons need to be on for the unit in the F layer to be on. The full system of E, C and F layers is shown in figure 11.3.

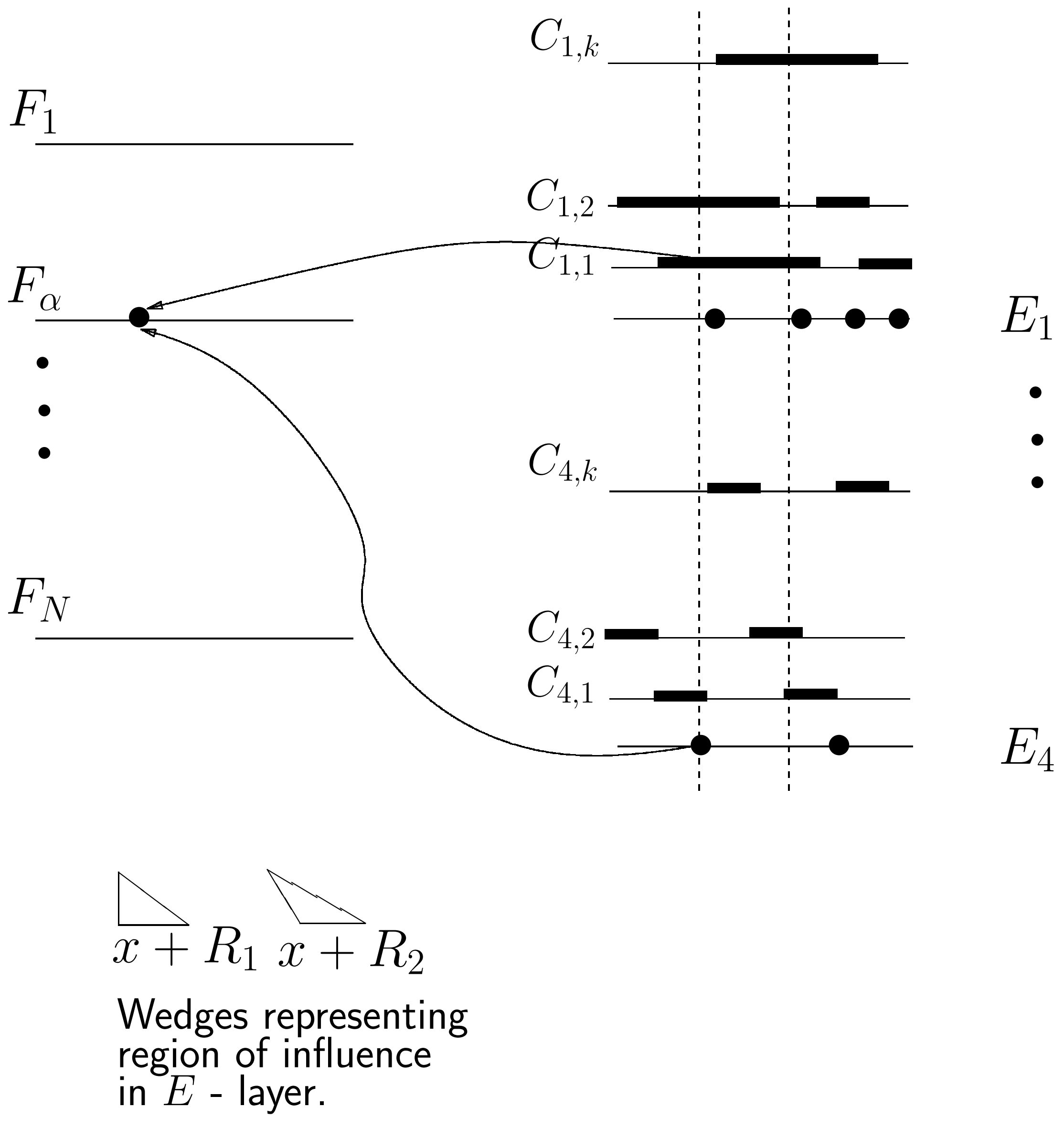

Figure 11.3: Retinotopic layers for detecting two-edge arrangements. Four E layers detect edge locations. For each E_e layer there is a $C_{e,k}$ corresponding to each $R_k, k = 1, \ldots m$. All in all $4 \times m$ - C layers. There is an F layer for each local feature. For feature $\alpha = (4, 1, R_1)$ each location x in the F_α layer receives input from *the same location x* in $C_{1,1}$ and in E_4.

11.2 Hebbian learning

The prevailing assumption regarding synaptic modification, formulated in Hebb (1949), is that it depends only on the state of the pre-synaptic neuron a and that of the post-synaptic neuron b. There are many possible ways to implement this general principle, see for example Hopfield (1982), Kohonen (1984), Amit (1989), Fusi et al. (2000). In the current context we employ a simple version of the methods used in Amit & Brunel (1995). Information learned by the system is coded exclusively in internal states of the synapses denoted S_{ab}, for a synapse connecting neurons a and b. The activity of the two neurons produces modifications in the internal synaptic states which then translates through a transfer function into the synaptic efficacy.

If both neurons are on *synaptic potentiation* occurs, namely the internal state increases by c_+. If the pre-synaptic neuron is on but the post-synaptic neuron is off, *synaptic depression* occurs, namely the internal state decreases by c_-. The values of S_{ab} are always restricted between 0 and S_{max}. This is written in compact form as

$$\Delta S_{ab} = c_+ u_a u_b - c_- u_a(1-u_b), \text{ if } 0 \le S + \Delta S_{ab} \le S_{max}, \tag{11.3}$$

otherwise $\Delta S_{ab} = 0$.

The synaptic efficacies are simple functions of this internal state, i.e. $J_{ab} = J(S_{ab})$ where

$$J(S) = \begin{cases} 0 \quad \text{for} \quad S \le L \\ \frac{S-L}{H-L} J^{AB} \quad \text{for} \quad L < S < H \\ J^{AB} \quad \text{for} \quad S \ge H \end{cases} \tag{11.4}$$

for some positive $L \le H$ and J^{AB}, which is the maximal value of the efficacy of a synapse connecting two layers A and B. If $H = L$ the synaptic efficacies are binary with values 0 and J^{AB} only. If $L < H$ there is a ramp between the low value 0 and the high value J^{AB}.

11.3 Learning an object model

Images from a training set for a particular object class are now presented to the system, registered to the reference grid G. Think of G as a subgrid placed at the center of the image grid L. Those parts of the F layers corresponding to the region G feed directly into a module M. This component of the network is no longer assumed to be retinotopic and is therefore called a module. Each unit in M receives input from one unit in the G subregion of the F layers. Thus M has a unit for each pair consisting of local feature α and a location $z \in G$, namely $N \times |G|$ units. In our particular example we will be using a 32×32 reference grid and 256 two-edge features so that the number of units in M is $N_M = 262,144$. A unit $m = (\alpha, z) \in M$ is on if $F_\alpha(z) = 1$.

Also feeding into M is an 'abstract' module A that we use to code object classes. For each class c there is a class subset A_c of A, which is randomly selected, of fixed proportion P_A. The set A_c is used to code for class c. While the system is learning the model for class c there is some hidden 'teacher' which is able to activate the subset A_c through channels outside the visual system. Each unit in M receives input from some *random* collection of units in A, of proportion P_{AM}. We place random connections because these modules are *not* retinotopically organized, and there is no preference for any particular connection. During training the local features from a registered image of class c are computed in F and the corresponding units are turned on in M. At the same time the units in A_c are turned on in A, see figure 11.4.

Let $m = (\alpha, z)$ be an activated unit in M. About $P_A \cdot P_{AM} \cdot |A|$ units $a \in A_c$ are feeding into m and all are on. The synaptic state S_{am} from each such unit is increased by c_+ as prescribed in equation 11.3. The synaptic state of any synapse connecting a unit $a \in A_c$ to a unit $m \in M$ which is *not* on is decreased by c_-, see figure 11.4. As examples of objects of class c are presented the internal state of each synapse, connecting

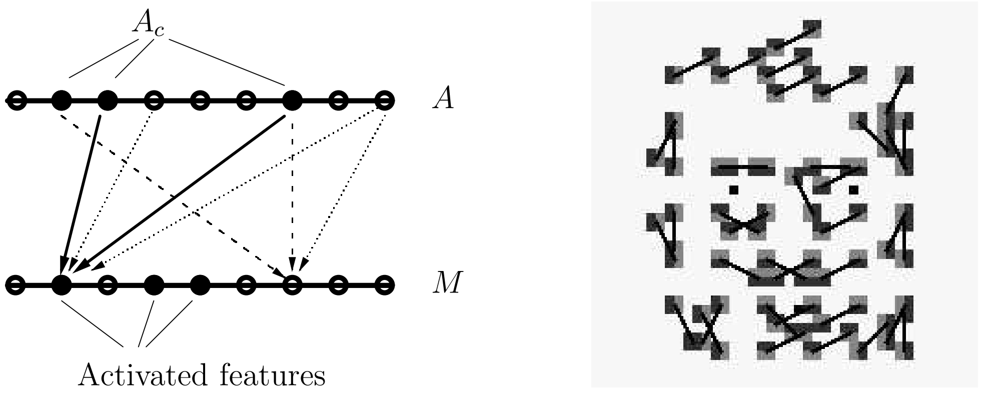

Figure 11.4: Left: Learning the object model. A_c units are activated at the same time data from class c is presented to F and replicated in M. Dark solid arrows: potentiating synapses (pre- and post-synaptic neurons on.) Dark dashed arrows: depressing synapses (pre-synaptic neuron on, post-synaptic neuron off.) Dotted arrows: no change in synaptic state. Right: An object model learned for faces using two-edge features.

a unit $a \in A_c$ to some unit $m \in M$, is performing a random walk. The mean increment of this walk is given by

$$\delta_{am} = c_+ p_{m,c} - c_-(1 - p_{m,c}),$$

where

$$p_{m,c} = P(F_\alpha(z) = 1|Y = c)$$

is the probability of feature α occurring at location z for object class c. Let $\sigma = c_-/c_+$, then after a large number of presentations, for those units m with

$$\frac{p_{m,c}}{1 - p_{m,c}} > \sigma, \quad \text{or} \quad p_{m,c} > \frac{\sigma}{1+\sigma} \doteq \rho. \tag{11.5}$$

the synaptic state S_{am} will tend to have a value closer to S_{max}, otherwise the synaptic state will have a lower value closer to 0. If the efficacies are binary, it is possible to set the value of $L = H$ in equation 11.4 in such a way that when $p_{m,c} > \rho$ the synapse J_{am} will be enabled with high probability. This system is learning an object model as prescribed in the learning stage of the sparse model described in section 6.4. For each training image all two-edge arrangements are computed and detected at all locations on the reference grid. Those above some probability are recorded as part of the model. This learning mechanism is schematically described in figure 11.4. The efficacy J^{AM} is set so that the input into a unit m from afferent units in A_c is above threshold only if most of these units are on. Thus we take $J^{AM} \sim (\theta_M - \epsilon)/(P_A \cdot P_{AM} \cdot |A|)$, for some small $\epsilon > 0$.

We have already defined the visual low level input obtained in the layers of oriented edges E_e. High level input telling the system which object to look for is given through external activation of a particular subset A_c, which in turn, after training, will cause the activation of the elements $m_i = (\alpha_i, z_i), i = 1, \ldots, n_c$ belonging to the model for class c. Ultimately this evoked model will drive a detection system which is capable of applying the model at every location in the image. This is described in section 11.5. On the right panel of figure 11.4 we show a model based on two-edge features trained on the face data. A schematic diagram showing the connections between A, M, F and E is shown in figure 11.5.

In the sequel it will be important that the same number of features/location pairs m_i are activated for each object model. However different object classes may have different numbers of activated model units,

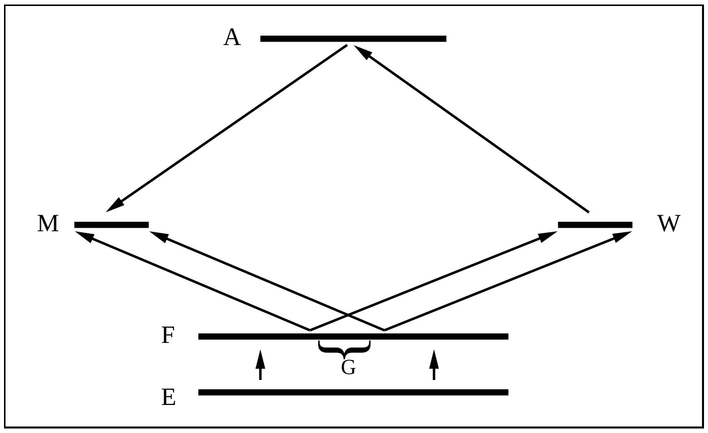

Figure 11.5: Network architecture. E layers - oriented edge detection on visual input. The two-edge arrangements are computed in the F layers. The center part corresponding to the reference grid G is replicated in the M and W modules. The M module receives input from the 'abstract' A module. The $A \to M$ synapses are used to learn object models. A receives input from W and the $W \to A$ synapses are used to learn classifiers.

corresponding to feature-location pairs m with probability $p_{m,c} > \rho$. If all models are epxected to have a fixed number n of features, using inhibitory units which receive input from the M layer and feed back into it, it is possible to ensure that only a random subset of size n of the total collection of n_c units remains active. We will not describe the details of such a mechanism here, since it requires introducing some real dynamics into the system.

11.4 Learning classifiers

The previous section presented a mechanism for learning object models by activating the units A_c at the same time that examples from class c are presented at the reference grid. The two-edge arrangements in the central area of the F layers are computed and activate the corresponding units in M. Now we turn to the problem of recognition or classification which involves discriminating between different object classes.

While the object models are being learned, the same visual information is copied from the F layers into another module W which again has a unit (α, z) for every feature/location pair, i.e. $N \times |G|$ units. A unit $w = (\alpha, z)$ is on if any unit in some neighborhood of $N_s(z)$ of z in F_α is on. The radius s of the neighborhood determines the degree of 'spreading' which was introduced in Chapter 9 and is important to ensure geometric invariance of recognition over local deformations and some range of linear transformations. The W module is also connected to A but now the synapses are directed from W into A. See figure 11.5. Each neuron in A receives connections from a random subset of proportion P_{WA} in W.

Hebbian learning

When an image from class c is presented in the reference grid and the neurons in A_c are activated, those synapses connecting activated neurons in W to neurons in A_c are potentiated. If w corresponds to a high probability unit for class c, those synapses connecting w to units in A_c will all tend to have a high internal state after several presentations of samples from class c. When an object from another class d is presented to the system a different subset A_d is activated in module A. If w also corresponds to a high probability unit in class d it will tend to be on during presentations of class d but the corresponding units in A_c will be off. This is precisely the setting in which the synapses connecting W to elements in A_c get depressed, i.e. the internal state decreases.

For each unit $w = (\alpha, z)$, let $p_{w,c}$ denote the probability of feature α in the neighborhood of z on class c

$$p_{w,c} = P(\max_{y \in N_s(z)} F_\alpha(y) = 1 | Y = c).$$

We introduce an additional quantity $q_{w,c}$ which is the probability of feature α at location z on the set of images *not* of class c

$$q_{w,c} = P(\max_{y \in N_s(z)} F_\alpha(y) = 1 | Y \neq c).$$

Let $P(c), c = 1 \ldots, K$ denote the prior distribution on object classes. Finally let

$$\tilde{p}_{w,c} \doteq p_{w,c} P(c), \quad \tilde{q}_{w,c} \doteq q_{w,c}(1 - P(c)) \quad \rho_{w,c} \doteq \frac{\tilde{p}_{w,c}}{\tilde{q}_{w,c}}. \tag{11.6}$$

The mean synaptic change for a synapse connecting a unit w to a unit $a \in A_c$ is given by

$$\delta S_{wa} = c_+ \tilde{p}_{w,c} - c_- \tilde{q}_{w,c}$$

With a large number of training examples for each class, S_{wa} will move toward one of the two reflecting barriers 0 or S_{max} according to whether the probability ratio

$$\rho_{w,c} > \sigma = \frac{c_-}{c_+}, \tag{11.7}$$

or not. The resulting synaptic efficacies will then be either 0 or J^{WA} respectively. This is in contrast to the situation for $A \to M$ synapses, where the only relevant information is the probability $p_{m,c}$ (except for the case where $a \in A_c \cap A_d$ which we ignore.)

After learning is completed, presenting an image from class c at the reference grid will bring about the activation of units in W, which in turn will cause the activation of units in A. The goal is to have the number of units activated in the set A_c to be larger than the number activated in other sets, thus signaling the presence of class c, or the fact that the system has recognized the input as being from class c. The number of units activated in each class subset A_d will depend on the value of J^{WA}, on the threshold θ_A of units in A, on the statistics $\tilde{p}_{w,d}, \tilde{q}_{w,d}$ of the units w and on the particular image presented. In real data sets the distribution of the quantities $\tilde{p}_{w,c}, \tilde{q}_{w,c}, w \in W$, and hence of $\rho_{w,c}$, is highly variable among different classes. See for example figure 11.6 where we show the 2d distribution of these pq probabilities for classes '0' and '1' of the NIST dataset with which we worked in Chapter 9.

The ratio θ_A / J^{WA} determines how many enabled units $w \in W$ need to be on in order to activate a unit in $a \in A_c$. These are necessarily units for which $\rho_{w,c} > \sigma$, (defined in equation 11.7), otherwise the efficacy $J_{wc} = 0$. As an example take $\theta_A / J_M^{WA} = 10$, i.e. input from ten afferent units is sufficient to activate a unit in A. Assume that in class c there are 64 out of 262144 units satisfying $\rho_{w,c} > \sigma$. For each unit in $a \in A$ let W_a be the set of afferent units of a and suppose that $P_{WA} = .5$ so that W_a is half the size of W, i.e. $|W_a| = 131072$. Since W_a is a random subset, the expected number of units from W_a satisfying $\rho_{w,c} > \sigma$, is 32 (one half of 64), and the corresponding synapses will have efficacy J^{WA}. Assume that for each such unit $p_{w,c} > .5$, then upon presentation of an image of class c the expected number of these units which are activated is 16. Taking the various random fluctuations into account, with high probability the actual number of activated units will be greater than 10 so that with high probability most units in A_c will be activated.

If on the other hand in class d there are only 32 features satisfying the inequality there is high chance that upon presentation of an image of class d only a small number of units of A_d will be activated. There is

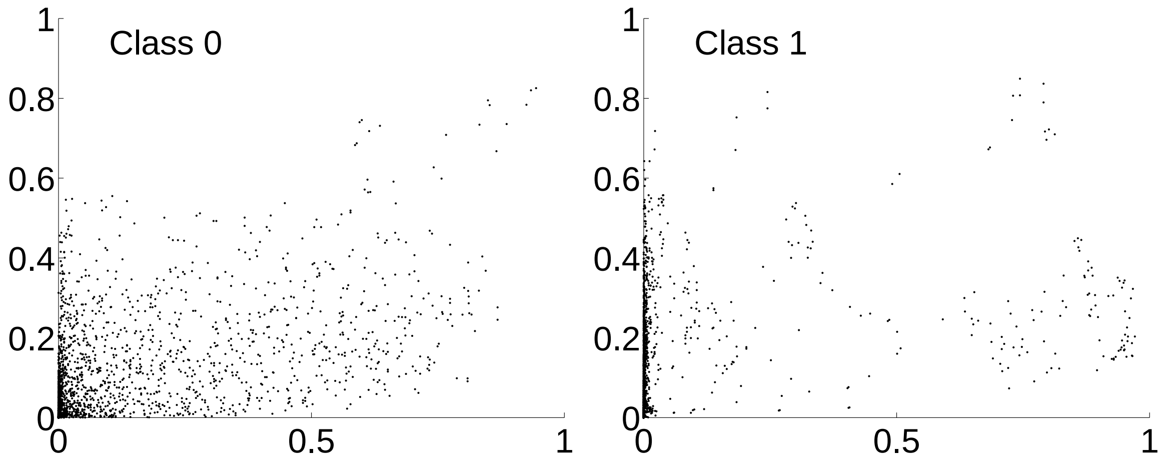

Figure 11.6: pq scatter plots for classes '0' and '1' in the NIST dataset. Each point corresponds to one of the feature/location pairs= (α, z). The horizontal axis represents the on class p-probability of the feature, and the vertical axis is the off class q-probability.

a significant probability that the number of units activated in A_d will be less than the number activated in some other subset $A_{d'}$, leading to a high error rate for class d. Lowering σ in equation 11.7 will enable more synapses from class d to have non-zero efficacy, but then units in A_c will tend to be activated too easily. One solution is to adapt the threshold of the units in A_c to the particular statistics of class c. This however would require quite a wide range of thresholds and violates our assumption that neurons in the same module have the same threshold.

Field dependent Hebbian learning

An alternative is to modify the learning rule to depend on the local field of the post-synaptic neuron which is recomputed for each training image. When the subset A_c is activated together with the presentation of an image from class c, potentiation of an afferent synapse occurs only if the local field h_a, as defined in equation 11.2, is below $\theta(1 + k_p)$, where θ is the threshold of a and $k_p > 0$. Formally the internal synaptic state is modified as follows.

$$\Delta S_{wa} = \begin{cases} c_+, & \text{if } h_a < \theta(1+k_p) \text{ and } u_w = 1, u_a = 1 \\ -c_-, & \text{if } u_w = 1, u_a = 0 \\ 0. & \text{otherwise} \end{cases} \tag{11.8}$$

The transfer from internal synaptic state to synaptic efficacy, $J_{wa} = J(S_{wa})$, needs to be computed at every update step of training in order to calculate the local field $h_a = \sum_{W_a} J_{wa} u_w$.

Assume that for some $a \in A_c$, at some stage of training, there are K afferent units $w_k \in W_a, k = 1, \ldots, K$ for which the internal state $S_{w_k,a} > H$. The efficacies of these synapses are at the maximum value J^{WA}. If $K = \theta(1 + k_p)/J^{WA}$, and if the particular units $w_k, k = 1, \ldots, K$ are activated by the current presented image of class c, the field h_a will be at least $\theta(1 + k_p)$. This prevents the internal states of *any* synapse feeding into a from increasing at this presentation, not only those at high internal state. Other images of the same class c may not activate all the units $w_k, k = 1, \ldots, K$. The resulting field on a could be smaller. For those units w that are activated by such images, potentiation *can* occur on S_{wa}, see figure 11.7. In contrast to regular Hebbian learning this form of learning is not entirely local at the synaptic level due to the dependence on the field of the neuron.

The main purpose of field constraints on synaptic potentiation is to ensure that after training the average field at a unit $a \in A_c$ is at approximately the same level upon presentation of an example of class c, irrespective of the specific class distribution of feature probabilities.

For example after field dependent training is completed for the NIST database, a certain distribution on the local field of units in A_0 and A_7 will be observed for examples from the correct and incorrect class. This is illustrated in the two panels on the left of figure 11.8. The top panels show the distribution of the field for 0 and 7 respectively, when the correct class is presented. The bottom two panels show the distributions for the incorrect class. It is clear that the average fields are very similar for both cases. This allows us to use a *fixed* threshold for all neurons in A. Note that the separation between the correct and incorrect class distributions does not reflect on the classification rate of the network as a whole as will emerge from the discussion below. For comparison, on the right of figure 11.8 we show the histograms of the fields when standard Hebbian learning is employed, all else remaining the same. The large variation in the location of these histograms between the two classes precludes the use of a fixed threshold for both classes.

Field dependent Hebbian learning and feature statistics

An interesting question is which synapses does the field dependent Hebbian learning tend to enable. There is some form of competition over 'synaptic resources' and if there are many units w connected to a unit $a \in A_c$ with a high value of $\rho_{w,c}$, not all will be enabled. On the other hand if there is only a small number of such units, the learning rule will tend to employ more features with lower ratios. In all cases those synapses coming from units with a higher $\rho_{w,c}$ have a higher chance of being enabled after presentations of examples from class c. Those with higher on-class probability $p_{w,c}$ may get potentiated sooner, but synapses with low $p_{w,c}$ but large $\rho_{w,c}$ gradually get potentiated as well.

If the $W \rightarrow A$ synapses are binary the outcome of the potentiation rule can be approximated quite well as follows. Sort all units according to their $\rho_{w,c}$ ratio. Let $p_{(w),c}$ denote the sorted probabilities on class c in decreasing order of ρ. Pick as many features from the top of the sorted list as are necessary to obtain an expected field of $\theta(1 + k_p)$.

$$\sum_{(w)=1}^{g_c} J_m^{WA} p_{(w),c} \sim \theta(1 + k_p). \tag{11.9}$$

The number g_c depend on the class c.

If the transfer function $J(S)$ is not a simple step function but has a 'ramp', the situation is somewhat more complex. However this allows the network greater flexibility and the classification rates are better.

Classification with multiple randomized perceptrons

After training is completed each unit $a \in A_c$ has a number of afferent connections, (approximately g_c as defined in equation 11.9,) from the set $W_a \subset W$ that have been enabled. Due to the randomized connections from W to A these subsets W_a are different. Now each unit $a \in A_c$ can be viewed as a simple classifier P_a which discriminates between class c and all the rest.

$$P_a = \begin{cases} 1 & \text{if } \sum_{w \in W_a} J_{wa} u_w > \theta_A \\ 0 & \text{otherwise.} \end{cases} \tag{11.10}$$

In other words unit a is a simple two-class linear classifier, also known as a *perceptron*, see Minsky & Papert (1969), Duda & Hart (1973). It is a very simple and constrained perceptron in that the weights on the synapses are all positive and bounded by J^{WA}. The classical perceptron can have negative weights on the

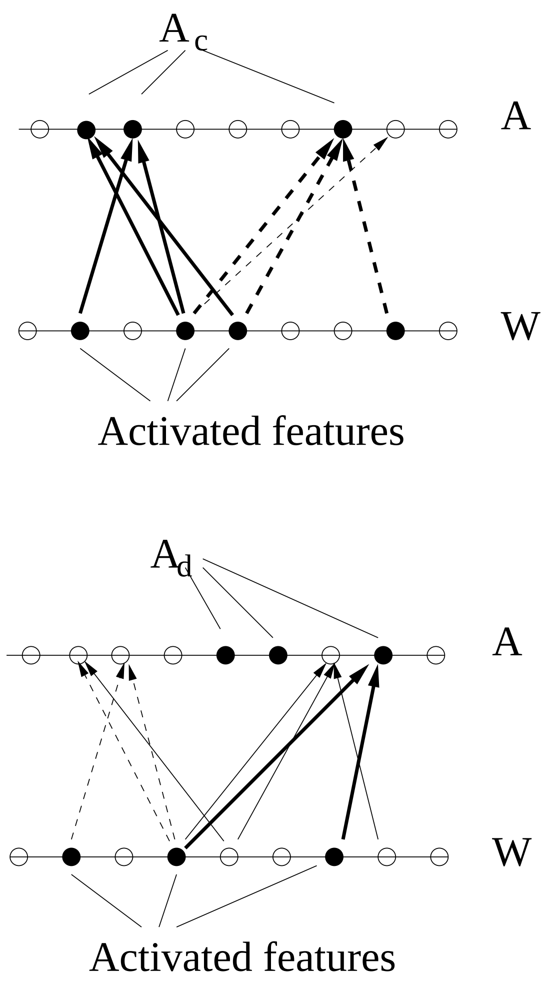

Figure 11.7: Training of classification network. Top: class c is presented. Bottom: class d is presented. Activity on the same synapses is compared. Thin solid line arrows: nothing happens in these synapses. Thick solid line arrows: potentiating synapses (the local field is not too large.) Thick dashed arrows: local field at neuron is high (three incoming enabled synapses) hence no potentiation, although both pre- and post-synaptic neurons are on. Thin dashed arrows: depressing synapses, pre-synaptic neuron is on, post-synaptic neuron is off.

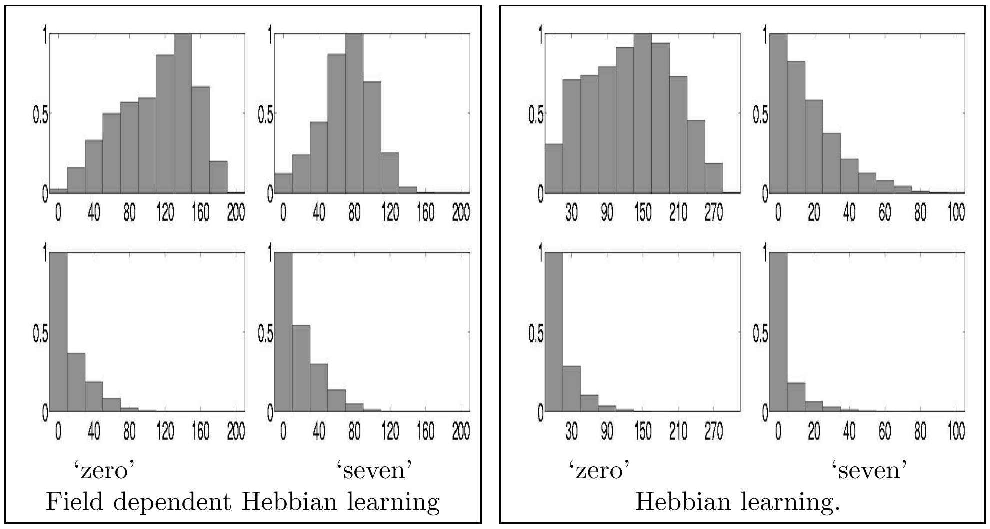

Figure 11.8: Left panel: Field dependent Hebbian learning. Right: Hebbian learning. In each panel the top two histograms are of fields at neurons belonging to class '0' on the left, class '7' on the right, when data points from the *correct* class are presented. The bottom row in each panel shows histograms of fields at neurons belonging to class '0', '7' when data points from the *other* classes are presented.

connections. Since there are many units in A_c, and since W_a is selected randomly, we have produced a large number of such simple classifiers with different sets of enabled synapses.

Visual input is presented at the reference grid and activates units in the W module through the E and F layers. The W layer then causes certain units in A to be activated. Classification is represented by the set A_c with the largest number of activated neurons. Namely the class that received most votes from its set of perceptrons.

This classifier based on multiple randomized perceptrons that are aggregated by voting is based on the same principle that led to the production of multiple randomized trees in Chapter 9. The difference is that each of the classifiers employed here distinguishes only between one class and all other classes joined together. This aggregation of simple classifiers yields surprisingly good results given the limitations imposed on each of the individual classifiers. For example on the NIST data set, with 10000 training samples, a network with 3000 neurons in the A layer, taking A_c to be of size 300, and $P_{WA} = .2$, $J^{WA} = 10$, $\theta_A = 100$ and in equation 11.8, $k_p = .2$, $c+ = 4$ and $c_- = 1$, the classification rate is higher than 94%. If several such networks are used with a simple boosting procedure we observe classification rates of 97.4%. Even for the LaTeX database with 293 classes such networks can reach classification rates near 80%, which although much less than that achieved with trees is still very encouraging. Note that currently there is no neural analogue of the boosting procedure. For more details on the performance of such networks and the incorporation of attractor dynamics in the A layer see Amit & Mascaro (2001).

11.5 Detection

We now construct a network for implementing the first step of the counting detector described in Chapter 8. Recall that all features used in any object representation come from a predetermined pool $\mathcal{F}$ of 256 features, and the image locations of each feature in $\mathcal{F}$ are detected in the arrays $F_\alpha, \alpha \in \mathcal{F}$. A location x in F_α is on if local feature α is present at location x in the image.

When a particular class A_c is activated in A, the corresponding units $m_i = (\alpha_i, z_i), i = 1, \ldots, n$ of the model are activated in M. This is a result of the learning procedure for the object representation. Step I of the counting detector described in Chapter 8 involved counting at each location $x \in L$ how many of the regions $x + B_{z_i}$ contained an instance of the feature α_i for $i = 1, \ldots, n$, see section 8.1. To implement this in a network we define, for each local feature array $F_\alpha, \alpha \in \mathcal{F}$, a system of retinotopic layers $Q_{\alpha,z}$ indexed by the locations z in the reference grid G. A unit at location $x \in Q_{\alpha,z}$ receives input from the region $x + B_z$ in F_α and is responsible for checking whether feature α is present in $x + B_z$.

For each unit $m = (\alpha, z)$ in M there is a corresponding $Q_{\alpha,z}$ array. In order for $x \in Q_{\alpha,z}$ to be activated both $m = (\alpha, z) \in M$ must be on *and* at least one unit in the region $x + B_z$ in F_α. Thus the model evoked in M *primes* the appropriate $Q_{\alpha,z}$ layers to a point where they *could* be activated if the appropriate input comes from below, i.e. the F_α layer. In terms of synaptic efficacies this can be achieved by having a very strong connection $J^{MQ} = \theta - \delta$ from $m = (\alpha_, z)$ to each unit in $Q_{\alpha,z}$ but very weak connections J^{FQ} of order δ such that $|B_z|\delta < \theta$. Input from the region $x + B_z$ in F_α alone cannot activate $x \in Q_{\alpha,z}$. It is necessary for $Q_{\alpha,z}$ to also receive input from $m = (\alpha, z)$. This form of 'priming' is rather unrealistic since it assumes a very strong input from one unit $m \in M$ to an entire Q layer of units. One solution is to have a population of units corresponding to each $(\alpha, z) \in M$ that collectively increase the input to all units in $Q_{\alpha,z}$. The system of Q layers sum into a retinotopic detection layer S. A unit at location $x \in S$ receives input from *all* $Q_{\alpha,z}$ arrays at location x, and is on if

$$\sum_{\alpha \in \mathcal{F}} \sum_{z \in G} Q_{\alpha,z}(x) \geq \theta_S,$$

where $\theta_S = \tau$ is the threshold for step I of the counting detector. Since the only units active in the Q layers are those corresponding to a count for the specific sparse model activated in M, the active units in the S layer are precisely those locations detected by step I of the counting detector.

Note that if we traverse the $Q\alpha, z$ layers at a fixed location x those units which are primed by input from M constitute a *shifted* copy of the object model to the region $x + G$. Thus the Q layers are the means by which the model learned in the central M module gets shifted to all locations. Since the detector is a simple thresholded sum with uniform weights all that needs to be 'shifted' is the information on the features and their relative locations, and as we see this can be achieved by a fixed wiring. No connections need to be changed. If the detector was more complex, even involving a weighted sum, it is not at all clear how the weights in the sum could be shifted in a simple and transparent manner.

This detection network is described in figure 11.9. This is a clear example of location selection driven by top-down flow of information. If a unique location needs to be selected, say for the next saccade (eye movement), then picking $x \in S$ with the highest sum seems a natural choice and can be achieved through competitive mechanisms within S using recurrent connections.

11.6 Gating and off center recognition

The visual system is able to recognize objects that are not present in the center of the visual field. So far our recognition scheme assumed that the data was present in region G, located at the center of the F layers and fed directly into W. We now introduce an intermediate system of layers between F and W whose role is to translate data from subregions of the size of the reference grid to the center. The main assumption is that all the information needed for recognition is stored in the synaptic connections between W and A and cannot be physically transfered to other regions of the image lattice.

For every possible location $y \in L$ define a layer U_y of units corresponding to feature/location pairs $(\alpha, z), \alpha \in \mathcal{F}, z \in G$ as in W. A unit $(\alpha, z) \in U_y$ receives input from the neighborhood $N_s(z + y)$ in F_α. Each of the U_y layers also receives input from the corresponding unit y in the detection layer S. The unit

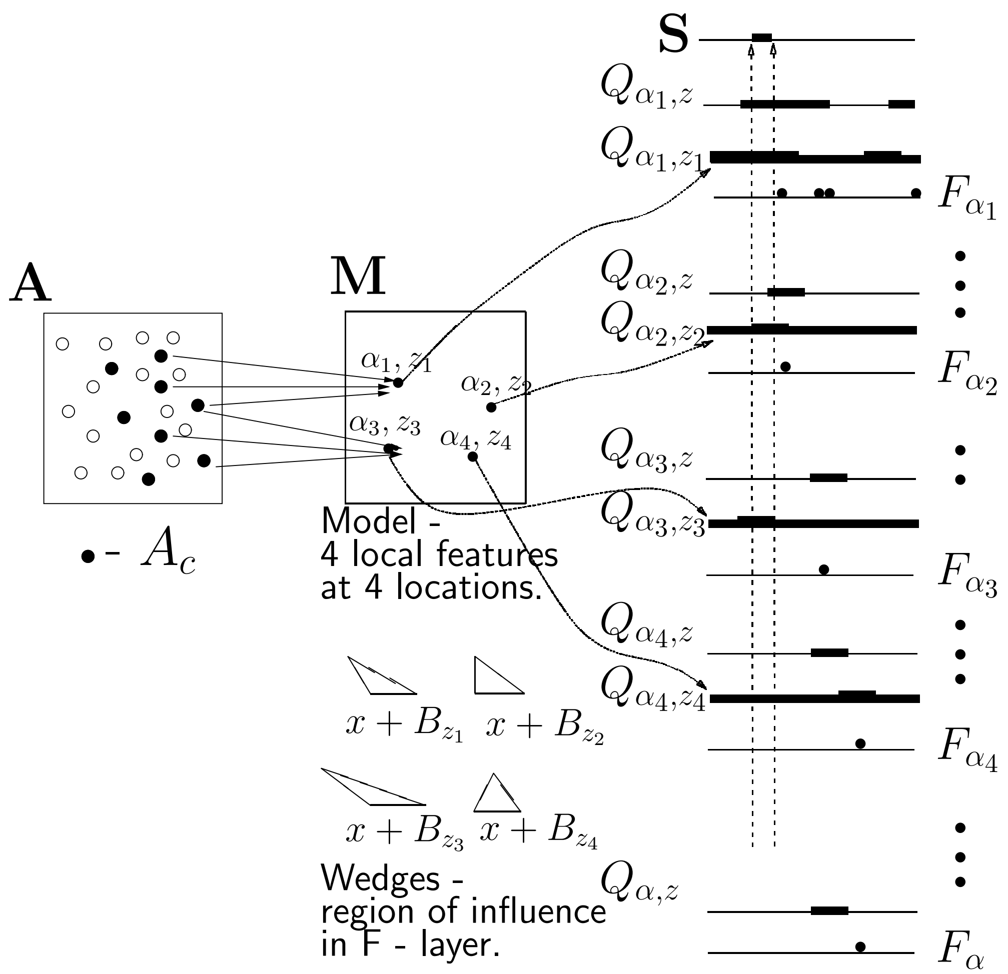

Figure 11.9: Detection network: The sparse model is composed of four features at four locations. Neurons of class c are evoked in the A module and turn on the model of class c in M. Each feature/location $(\alpha, z) \in M$ pair provides input to all units in the corresponding $Q_{\alpha,z}$ layer (thick lines). The locations of the feature detections in the F layers, are shown as dots. They provide input to regions in the Q layers shown as double thick lines. At least three local features need to be found for a detection. Each $x \in S$ sums all inputs $Q_{\alpha,z}(x)$ and is on if this sum is above a threshold τ. In this example the fourth feature does not contribute to the detection - it is not present in the correct location.

$(\alpha, z) \in U_y$ is on if any of the units in a neighborhood $N_s(z + y)$ of $z + y$ is on in F_α, so that U_y replicates the information in the F layers, in a window the size of the reference grid centered at y.

The input to W no longer arrives directly from F. Rather we set $w = (\alpha, z) \in W$ to be on if the unit (α, z) is on in *any* of the U_y layers. Thus the activity in W at (α, z) is the *union* of activities of (α, z) in all U_y layers. In the absence of any gating of activity in the U system, a complex image will generate a large amount of incoherent activity in W, since each unit $(\alpha, z) \in W$ is on if the corresponding unit is on in *any* of the U_y layers, it is on if feature α is present *anywhere* in the visual field.

Only when a particular location is selected for attention, namely a particular unit in S is activated, does order emerge. If a particular location y is activated in the detection layer S, there is an increase in input to U_y through priming mechanisms similar to those suggested for the Q layers. At the same time, any activity in S will activate a pool of inhibitory neurons which feeds into the entire system of U layers. This inhibitory input weakens the input to all neurons in U, and the only ones surviving with inputs above threshold are the units in U_y, corresponding to the selected location, which also received input from the F layer. We will not describe the details of such mechanisms but refer the reader to Amit & Brunel (1997), and Mascaro & Amit (1999), where inhibitory units are employed to achieve similar results.

Now the only input into W is therefore a translation of the data in the neighborhood of the activated location $y \in S$. This is a *gating* procedure which singles out a particular spot in the scene for classification. Whereas the Q layers are primed by the detection model in M in terms of which *local features* should be attended to, the U layers are primed by S in terms of which *location* to attend to. This network is shown in figure 11.10. Note that if y is close to y' the windows $G + y, G + y'$ can have significant overlaps, hence the need for replication. Otherwise the priming could occur directly in the F layers.

The location y in S can be selected through the detection process described earlier, which is initiated with a top-down flow of information through the model evoked in M. As discussed in Chapter 10 this may be a coarse model representing an *object cluster*, and further classification is needed. Once W is activated by the translated data from U_y, the connections from W to A produce a classification of the data. This is essentially equivalent to the detection and classification scheme proposed in Chapter 10.

Alternatively the selected location in S can be the outcome of very primitive bottom-up processing. For example, if only one object is present in the scene, its location pops out easily as a 'blob' at very low resolution and perhaps there is a separate mechanism whereby S gets activated by such simple 'blob' detections as suggested in Olshausen et al. (1993). During learning only one example of the object is presented in the image at each presentation. This simple form of bottom-up selection provides a mechanism whereby the information gets properly translated into W. Since only one object is present in the scene there is no ambiguity. Another mechanism to overcome ambiguity during the learning phase is motion. If the learned object is moving against a static background, again a simple mechanism is able to detect the location of interest.

11.7 Biological analogies

The system described above provides a possible answer to the main questions posed at the outset. Learning is achieved in a central module. The object models are simple lists of binary features at particular locations on the reference grid. This allows for simple mechanisms for transmitting model information to the entire field of view. Those units coding for the features appropriately shifted are primed and detection is achieved by simple counting at the S layer. Implicitly we have achieved the implementation of a centrally stored classifier (object/non-object) at every place in the scene, since all it requires is counting and thresholding. It would be very hard to do so for more complex types of classifiers.

Recognition is achieved by selecting a particular location for processing, and translating the data to the central recognition module. One could argue that detection can also be achieved by simply shifting every

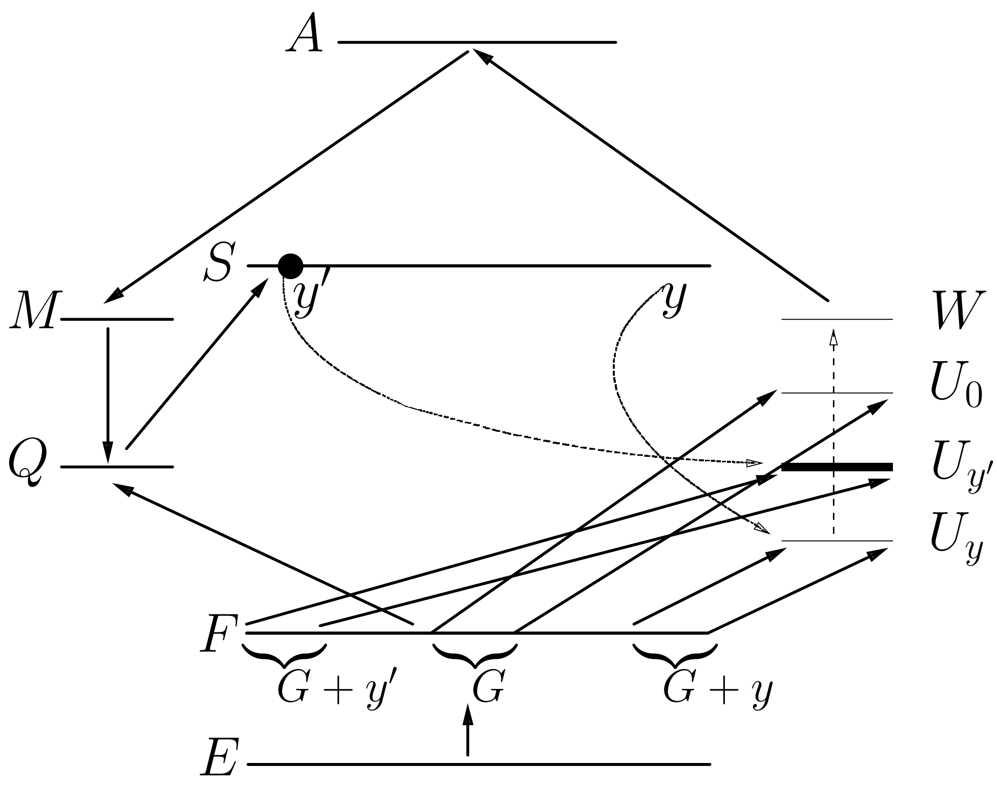

Figure 11.10: The translation and gating mechanism. Each region $G+y$ is replicated in the layer U_y, which also receives input from $y \in S$. The W module takes in all the activity in the U layers. y' - selected location in S, primes $U_{y'}$ and suppresses the other U layers. The activity in W is then a copy of $U_{y'}$. The system of Q layers is shown as one line with the arrows representing the direction of information flow. The F layers feed into the Q layers. The object model in M primes Q, and the detected location y' in S is obtained by summing and thresholding the Q layers at each location.

reference grid window to the center and classifying object/non-object. This would be prohibitive in time, since the windows would have to be processed sequentially, whereas detections occurs very fast.

The system also has interesting analogies to the biological visual system which we describe below.

Labeling the layers.

Analogies can be drawn between the layers defined in the network and existing layers of the visual system. Edge detector layers $E_e, e = 1 \ldots, 8$ correspond to *simple* orientation selective cells in layer V1. As noted in the introduction these layers represent a schematic abstraction of the information provided by the biological cells. However the empirical success of the algorithm on real images indicates that not much more information is needed for the task of detection and recognition.

The $C_{e,z}$ layers correspond to *complex* orientation selective cells in layer V1. One could imagine that all cells corresponding to a fixed edge type e and fixed image location x are arranged in a cortical column. The only difference as one proceeds down the column is the region over which spreading (ORing) is performed, namely the receptive field. In other words the units in a column are indexed by the displacement region R_k. Indeed any report on vertical electrode probings in V1, in which orientation selectivity is constant, will show a variation in the displacement of the receptive fields, see Zeki (1993), Hubel (1988).

The local feature layers F_α correspond to cells in layer V2 which respond to more complex structures as reported in von der Heydt (1995) and Hedgé & Van Essen (2000). The Q and the U layers may correspond to layer V4 cells. These have much larger receptive fields, and the variation of the location of these fields as one proceeds down a column is much more pronounced than in V1 or V2, it is on the order of the size of the reference grid.

Alternatively it may be that the Q and U layers are within V2. Without an hypothesis on the particular shifting function of the neurons in these layers, one would not be able to distinguish the behavior of Q or U layer neurons from F layer neurons. They all respond to more complex features perhaps with some differences in the size of the receptive field. The difference would then be in terms of their projections to higher levels of the system. Under this hypothesis V4 is simply a collection of even more complex features, perhaps generic parts such as the 'loop' encountered in Chapter 10. These are hard wired retinotopically, since they are useful in constructing object models and classifiers. Furthermore associated Q and U layers exist within $V4$ as well.

Learning

Hebbian learning is the dominant paradigm in the neural network literature, see Hebb (1949), Hopfield (1982), Amit (1989), Oja (1989) and Durtewitz et al. (2000). The idea that synaptic potentiation is modulated as a function of the local field or firing rate of the post-synaptic neuron is still speculative, although some form of regulation of synaptic activity as a function of the post-synaptic neural activity is reported in Abbott & Nelson (2000). The attractive aspect of the simple classification network described above, together with the field dependent Hebbian learning rule, is that real data can actually be successfully classified with such a system. High classification rates on character recognition have been achieved with two-layer feed forward networks (LeCun et al. (1998)), but the training procedure for such networks is not local and depends on the optimization of a global function of all the synaptic weights.

The 'abstract' module A which codes for the classes could well be in parts of the cortex which are not directly involved in processing visual input, or it could be present in higher levels of infero-temporal cortex (IT). It could fit in a location which integrates inputs from diverse sensory modules such as prefrontal cortex or receive input from such a location. The external activation of the units corresponding to a class, during training, could be enabled by a non-visual stimulus which is well recognized. This can be viewed in the context of reward type stimuli, arriving from some other pathway (auditory, olfactory, tactile) as described in Rolls (2000), potentiating cues described in Levenex & Schenk (1997) or 'mnemonic' hooks introduced in Atkinson (1975).

Interaction between bottom-up processing and top-down information flow

Top-down and bottom-up processing are explicitly modeled. Bottom-up processing is constantly occurring in the simple cell arrays E_e, which feed into the complex cell arrays $C_{e,z}$ which in turn feed into the F - V2 type arrays. The priming from the M module or from the S layer determines which components of the data flowing up from the bottom will be processed at higher stages. The object class which is evoked in the main memory module A activates the object representation in M, and this determines which of the Q arrays will have enhanced activity towards their summation into S. Thus the final determination of the candidate locations is given by an interaction of the bottom-up processing and top-down information flow. The active location in the S layer determines which data flows from the U layers to the W module to be ultimately classified in the A module.

Gating and invariant detection.

The summation array S serves as a gating mechanism for visual selection through its input into the U layers. This could provide a model for the somewhat puzzling behavior of IT neurons in delay match to sample (DMS) experiments, see Chelazzi et al. (1993), Desimone et al. (1995). Two objects are selected and neurons in IT are identified which respond to the first and not to the second and vice-versa. The subject is then presented with one of the two objects as a target to be detected - the sample. After a delay period an image with both objects is displayed, both displaced from the center. The subject is supposed to saccade to the sample object. After presentation of the test image, neurons responsive to *both* objects become active. About 100 milliseconds later, and a few tens of milliseconds prior to the saccade, the activity of the neurons selective for the non-sample object decays. Only those neurons selective for the sample object remain active.

At first glance the fact that both types of neurons are active at the start could be interpreted as the brain having very quickly recognized the two objects, as well as having identified their location. In other words two detection and recognition tasks had already been performed. This explanation is quite unlikely if, as discussed above, we preclude the possibility of many classifiers operating simultaneously in parallel across the scene.

Alternatively this experiment can be explained in terms of the network described above. Assuming the recordings were made in A, at the outset there is no gating and all information from all U layers flows into W. Sets of features characteristic of the two objects, and probably many others, are activated and subsequently activate quite a number of class populations in A. This activation is essentially meaningless and conveys very little information to the higher processing centers of the brain. However once the object model, which was evoked in M at the sample presentation stage, is detected through the S layer, the detected position gates the activity from the U layers. Now the resulting activity in W comes from a particular window in the field of view, which contains the target. The only activity persisting in A corresponds to the subset coding for that object. This corresponds to the weakening of the activity of the non-sample neurons in the DMS experiment. Indeed this weakening signals that the information conveyed by the A layer, IT in this context, has become meaningful, and represents recognition of the sample object.

11.8 Bibliographical notes and discussion

Detailed descriptions of the human and primate visual systems can be found in Hubel (1988), Zeki (1993) and Tovee (1996). Recent physiological experiments on object recognition and detection in primates are summarized in a number of reviews, see Tanaka et al. (1991), Desimone et al. (1995), Desimone & Duncan (1995).

The detection and recognition networks described in this Chapter were proposed in Amit (2000) and Amit & Mascaro (2001). In the use of simple/complex layers there are important similarities with Fukushima (1986) and Fukushima & Wake (1991). Indeed in both papers the role of ORing in the complex cell layers as a means of obtaining invariance is emphasized. However there are major differences. In the detection model presented here training is only done for local features. The global integration of local level information is done by a fixed architecture, driven by top-down flow of information. Therefore features do not need to get more and more complex as in Fukushima & Wake (1991). Namely there is no need for a long sequence of layers. Robust detection occurs directly in terms of the local feature level which has only the oriented edge level below it. With large numbers of learned complex features it is not clear how invariant detection can take place.

The ideas behind the translation mechanism in the U layers are very similar to that of Olshausen et al. (1993). Here translation is simply done through replication. By contrast in Olshausen et al. (1993) a more complex mechanism is proposed involving control neurons which directly effect synaptic connections and allow for an orderly translation of data directly from the F layer to W. The proposal in Olshausen et al. (1993) for *selecting* a location for translation is limited to very low resolution 'blob' detection. This could be useful for direct bottom-up selection, but can not accommodate top-down selection due to a specific target object.

The network learning mechanism proposed above consists of an input module coding for feature/location pairs feeding into an 'abstract' classification module. The main activity during learning involves updating the synaptic weights of the connections between the input module and the classification module. Such simple networks have recently been proposed both in Riesenhuber & Poggio (1999) and in Bartlett & Sejnowski (1998). In the former the classification module is a classical output layer with individual neurons coding for different classes. This is insufficient for classification in relatively simple problems such as character recognition. Populations of large numbers of perceptrons are essential.

In Riesenhuber & Poggio (1999) the input layer computes *large range OR-ing*, or MAX operations, of complex features which are in turn conjunctions of pairs of 'complex' edge type features. In an attempt to achieve large range translation and scale invariance, the range of OR-ing is the *entire central visual field,* corresponding more or less to the reference grid. This means that objects are recognized based solely on the presence or absence of the features, *entirely ignoring their relative locations.* With sufficiently complex features this may well be possible, but then the combinatorics of the number of necessary features appears overwhelming. It should be noted that in the context of character recognition studied here we find a significant drop in classification rates when the range of OR-ing or maximization is on the order of the reference grid size, see for example the experiments reported in 9.6 in the context of classification trees. This tradeoff between feature complexity, combinatorics, and invariance is a crucial issue which has yet to be systematically investigated.

In the architecture proposed in Bartlett & Sejnowski (1998) the input features are again edge filters. Training is semi-supervised, not directly through the association of certain neurons to a certain class, but through the sequential presentation of slowly varying stimuli of the same class. Hebbian learning of temporal correlations is used to increase the weight of synapses connecting units in a recurrent layer responding to these consecutive stimuli. At the start of each class sequence the temporal interaction is suspended. This is a very appealing alternative form of supervision which employs the fact that objects are often observed by slowly rotating them in space. This network employs continuous valued neurons and synapses, which can in principle achieve negative values, and learning is not really local and incremental. It would be interesting to study whether field dependent Hebbian learning can lead to similar results without use of such global operations on the synaptic matrix.

The field dependent learning rule is motivated on one hand by work on binary synapses in Amit & Fusi

(1994), Mattia & Del Giudice (1999), where the synapses are modified stochastically *only* as a function of the activity of the pre- and post-synaptic neurons. Here however we have replaced the stochasticity of the learning process with a continuous internal variable for the synapse. This is more effective for neurons with a small number of afferent synapses. On the other hand there is a close connection to the work in Diederich & Opper (1987), where Hopfield type nets with positive and negative valued multi-state synapses are modified using the classical Hebbian update rule, but modification stops when the local field is above or below certain thresholds. There exists ample evidence for local synaptic modification as a function of the activities of pre and post-synaptic neurons, Markram et al. (1997), Bliss & Collingridge (1993). However at this point we can only speculate whether this process can be somehow be controlled by the activity of the post-synaptic neuron.

The network described above is based on a fixed set of hard wired two-edge features. In Amit (2000) it is shown how more complex features can be used for detection. These are no longer hardwired and are adapted to the detection task at hand. Again top-down priming plays a central role. In this context it is operating at the lower level of the C layers. The complex edge arrangements are detected in certain F layers. These will be responding to different features depending on the detection task at hand and the models evoked in the M module.

The suggested architecture provides and explanation to a variety of physiological experiments on one hand and on the other performs well on detection and recognition tasks involving real data, i.e. detection of objects in gray scale images, and classification of shapes from a large number of classes. There are many aspects which may turn out biologically impossible, such as the particular connections required between M and Q or between S and U, or even whether U and Q type neurons exist. Yet it appears to be a productive framework for formulating hypotheses on the mode of operation of higher level functions of the visual system.

Chapter 12

Software

This chapter provides some information regarding the software and data sets which accompany this book. The important data structures, as well as the most commonly used parameters are described. More information can be found in the documentation written in the source files and the script files. There is no guarantee attached to this software (it is not too hard to make it crash,) nor is any support to be expected. A certain level of proficiency in C++ is essential to understand the program, and some experience with Unix and an X11 based window manager are necessary to get things running smoothly.

12.1 Setting things up

The source code provided here will compile on Linux.
Download
detect.tgz
from
http://galton.uchicago.edu/˜amit/book.
to a directory whose full path will be called *base* for further reference.
Type
tar xvfz detect.tgz.

The following must be set for things to work.

- Set an environment variable *$DETDIR* to *base.*

  In *csh* add the line

  setenv DETDIR base

  to your *.cshrc* file.

  In *bash* add the line

  DETDIR=base

  to your *.bashrc* file.

- Add *base/bin* and *base/bin/script* to your path.

  In *csh* add the line *setenv PATH "$PATH":base/bin:/base/bin/script*

In *bash* add the line
PATH="$PATH:base/bin:/base/bin/script"

- Add the *bash* shell program to your */bin* directory. All the scripts are written in *bash* and assume it is in directory */bin*

In directory *base* you will now see several directories.

- *source.* Contains the code with graphic options. *cd* into *source* and type

  make

  The program will compile and *face* will be written to *base/bin*

- *sourcenox.* Contains the code with no graphic options. *cd* into *sourcenox* and type

  make

  The program will compile and *facenox* will be written to *base/bin*

- *bin.* Contains the compiled executables, *face* and *facenox*, and a subdirectory *scripts* where all the script files written in *bash* are stored.

- *book.* This directory has subdirectories corresponding to the chapters of the book (*chap1, chap2, ...*), as well as some subdirectories with data written in upper case. Within each subdirectory corresponding to a particular chapter are parameter files that more or less reproduce the figures in that Chapter. Running these scripts is a good way to begin getting acquainted with the program and the relevant parameters.

  The subdirectories with data are the following.

  – *FACES.* Contains a subdirectory *train* with 300, 110×96, images of faces from the Olivetti dataset, ten images per person. These are used to train the detectors. There is an additional directory *test* with 100 faces from the same dataset. The directory *pgm* contains a number of pgm images on which detectors can be tested. The directories *filt1, filt_d2, filt_from_edges* contain different sparse models trained using different parameters.

  – *HEART.* Contains ultrasound images of heart ventricles in directories *pat1*, *pat2*, *pat3*, and a couple of angiograms in directory *ang.*

  – *BRAIN.* Contains two directories of axial MRI brain scans, train and test, as well as a directory *filt3* which contains a sparse model for these images, and a directory *grmtch* containing parameters for a sparse model of these scans for detection with dynamic programming.

  – *LATEX.* Contains a directory *protos* with the prototypes of all the 293 LaTeX symbols, and a subdirectory *latex_0* with a sparse model for the symbol *0* as well as a classifier for the hits of this detector (see Chapter 10). Subdirectories *latex_1, latex_4, latex_7* contain models for the 1, the 4 and the 7.

  – *ESCR.* Contains a sparse model and various templates for the $\mathcal{E}$ used in Chapter 2. Also the classifier for hits of this detector on other script style symbols.

  – *CLIP.* Contains a sparse model for the clip shown in Chapter 8

  – *CHESS.* Contains the sparse model and classifiers for the chess-pieces.

  – *NIST.* Contains one set of classification trees trained on the NIST data set and a small sample of 10,000 NIST digits for testing. The full data set is very large but can be obtained upon request.

Running the program

The program *face* can receive input from a parameter file or from the command line or from the parameter file and the command line. The general form for running *face* from the command line is

face file par1=n1 par2=n2 ...

or

face par1=n1 par2=n2 ...

The parameter file, if used, must come first. Parameters set on the command line override values set in the parameter file. Among the parameters *opt* must be set to a particular option which tells the program what routine to use.

If no graphics is needed then *facenox* can be run the same way. The list of parameters needed for each of the algorithms is detailed in section 12.4

In each subdirectory of *book* with a name corresponding to a chapter are prepared parameter files for the figures in that chapter. Type

face f.par

on the command line to obtain the corresponding figure.

Graphics

The program will show results of the algorithms as they are computing, as well as the final result, depending on parameter settings. Often the program will show one or several windows and will not continue until prompted by the user. This is done by typing *c* inside the active window which is should be highlighted by the window manager. Typing *q* kills the window and the program. Typing *n* magnifies the window to *n* times the original size of the image.

12.2 Important data structures

In this section we provide a brief summary of the important C++ templates and classes used in the program. More details are to be found in the corresponding *.h* files. All classes come with some IO functions for reading, writing and printing information.

grid

The *grid*, defined in *grid.h*, is a template for arrays of all types. It contains functions for allocating memory, accessing coordinates, copying, copying sub-arrays, etc. *dgrid*, *igrid*, *ucgrid* are particular instantiations of this template for the types *double*, *int*, and *unsigned char.*

image

image, defined in *image.h*, is a friend class to *ucgrid* (unsigned character grid). This class has various image processing type member functions.

The program reads in images from *Images_n* files $n = 0, 1, ..N$. These files contain up to 100 images. If $N > 0$ then all files for $n < N$ contain exactly 100 images and the last one may contain less. Multiple images created in the program can also be written in to such files. There is a function which takes a sequence of *pgm* images called *pic0.pgm, pic1.pgm .. picN.pgm* and translates them into *Images_n* files. The corresponding script is call *pgmtoIm.* Images are stored with 1 byte per pixel allowing for 256 gray scale values. The only exception is the NIST data which is stored with 1 bit per pixel. To view an image from such a dataset which is located in directory *IM* type *face opt=40 dataset=IM first=10 numdigits=30*

Each image will appear and you will need to type 'c' in the window to proceed to the next, see script *viewim*. To see a collection of images at once type

face opt=41 dataset=IM first=10 numdigits=30 last=10 scale=32

This will put up the 30 images starting from the 10'th, 10 images per row each taking up a 32x32 box. If viewing NIST data add *datadepth=1* to the command line.

viewport

viewport defined in *viewport.h* manages the interface between the images and the X11 graphics and has various graphics functions such as line, point and text drawing on images. It is used in conjunction with *xobject* defined in *xobject.h*. An *image* or *ucgrid* is attached to a *viewport* and then the *viewport* is displayed with all the relevant information on *points*, *lines* etc. The display is interactive, the window can be magnified by typing the magnification digit on the window, *q* kills the entire process, *c* continues the program, *r* reverses the video.

landmarks

landmarks defined in *landmark.h* is a class for storing locations of detected local features of several types. The type is coded by an increasing index from 0,1 etc. Functions are included to obtain the number of instances of each type, coordinates of each instance of each type. The *landmark* (without an *s*) class is a structure used within the programs to record two coordinates and a type. These are accumulated during the actual detection process. Then a member function of *landmarks* called *setup_from_lmarklist* takes the array of *landmark* - s and arranges them in one *landmarks* class. For the use of various training procedures, landmarks for large numbers of images are stored in files *Lands_n* organized like the *Images_n* files in batches of 100. The member functions for reading and writing *landmarks* from these files are provided. To dump the landmarks of an image see script *dumplandmarks*.

tree

The *tree* template is defined in *tree.h* and includes definitions needed to create binary trees with quite general nodes. The main component is the *treeiterator* which has built in functions for moving up and down the tree, creating children nodes etc. The information in each node denoted *data* is defined for particular instantiations of this template, such as an *ltatree* defined in *ltatree.h*.

- *ltatree*. LTA stands for *local tag arrangement*, where local tag refers to any local feature such as an edge, a micro-image code, ridge-detector etc. This class is used to construct and store trees with relational and absolute arrangements of local features, as described in Chapter 9, as well as the local edge arrangements. The local edge arrangements can be viewed as relational trees with only a small number of *yes* answers, all the relations being with respect to the first edge, and all edges constrained to be near the center edge.

  Class *ltadata* is used for the *data* entry. The *ltadata* class codes for the additional relational question asked at the current node using the *question* class, it codes for the list of data points at the node while growing the tree, and the histogram of frequencies, also used for growing the tree. To view an *ltatree* use script *dumpltatree*.

- *carttree*. Another *tree* example is *carttree* defined in *carttree.h*. The query at each node is simply a pair (f, s) where f is a predictor index, and s is a threshold for the value of the predictor. The query is whether $X_f > s$. When using binary local features, $s = 0$, and the size of the predictor vector is

$dx \cdot dy \cdot N$, where dx, dy are the dimensions of the image and N is the number of local features. The class *cartdata* is where the information for simple cart type questions (coordinate and threshold) are stored as well as the list of data points and the histogram also used in training. There is some duplication in that *carttrees* do exactly what *ltatrees* with absolute arrangements do. For various reasons the former are used for post-detection classification: classifying one class against rest of the world, and then among several object classes. The latter is used in the experiments on isolated object recognition. To view a *carttree* use script *dumpcarttree.*

detection

The *detection* class is defined in *detection.h.* The important data entries are the *scale* which records at which resolution the detection was found, *tripts* the coordinates of the detection triangle. *cartprobs*, *cartprobs2*, *cartprobs3* are used to record the output from the various classification stages, *counts* used to store the number of hits from the counting detector and other related information, *locs* is an *igrid* for storing the coordinates of the instantiation. The number of points depends on the particular application.

12.3 Local features

All routines for detecting binary local features in an image can be found in the file *edge.C.* Below we list the different types together with the relevant parameters and how they can be computed and displayed.

Edges and Ridges

These are computed through the *detectedges* subroutine which loops through disjoint blocks of size *EdgeSizeBlock* in the image, calls a routine to find all binary features in that block, sorts these detections according to some criterion and keeps the top *EdgeNumberOfPixels.* The final detected features are stored in a *landmarks* structure. To view the edges on an image (say no. 3) in *Image_* files in directory IM run

face parfile opt=53 dataset=IM showedges=1 first=3 last=4

with *parfile* set according to one of the parameter sets defined below. To create a database of local features. Do

mkdir LF

face parfile opt=44 onlyedges=1 dataset=IM numdigits=num landdir=LF

with parfile set according to one of the parameter sets defined below. See also script *getedges.*

EdgeType=0. 0 - detect edges defined in section 5.4 using *find_edge_in_block* subroutine. 1 - Detect ridges with values on the ridge higher than outside. 2 - values on the ridge lower, (see section 4.1). Last two use *find_ridg_in_block* subroutine.

EdgeNumberOfPixels=2. Keep the top 2 edges found in each disjoit block, in terms of absolute value of the differences.

EdgeSizeBlock=2. Use disjoint 2x2 blocks.

EdgeRadius. The value of distance along ridge for comparisons, μ_a in section 4.1.

EdgeEpsilon=.03. Minimal edge threshold is $255 \times .03$

edgeinc=1. Use all 8 oriented edges. *edgeinc=2*, use only vertical and horizontal.

primitiveedges=0. Use polarity sensitive edges. *primitiveedges=50* use polarity insensitive edges (so only four types: horizontal, vertical, and two diagonals).

EdgeGoodTests=5. Number of inequalities defining an edge (in equation 5.11.) which are satisfied is greater than 5. Total number is 6. Lower values make more robust and frequent edges.

Local edge arrangements - LTA's

These arrangements are chosen through training for a particular object (see below) or a fixed pool can be defined. The individual trees containing the information for each of the local edge arrangements are all stored in a file *ltatrees* in a directory (say *treesgood*). These arrangements are detected in the routine *edges_to_ltas* which is called from *extract_landmarks.* The number of local edge arrangements is also stored in this file. To view the locations of local edge arrangements on an image (say no. 3) do

```
face parfile opt=53 onlyedges=0 showltas=2 ltatreedir=treesgood first=3 last=4.
```

where parfile contains the parameter settings for the edge extraction, as described in the previous section, which must precede the computation of local edge arrangements. The locations of each local feature type are shown in sequence. Press *c* to proceed. To create a database of local edge arrangements. Do

```
mkdir LF
face parfile opt=44 onlyedges=0 merge=0 dataset=IM numdigits=num landdir=LF ltatreedir=treesgood
```

If *merge=1* the edge information is appended after the local feature information in the *landmarks* structure. See script *getedges.*

Hardwired edge pairs

A special case are local edge arrangements with only two edges and 8 possible boxes defining the relative location of the second edge (as opposed to wedges). The advantage of these is the speed of computation. All 256 are computed in one loop based on an initial edge extraction, in *detectpairs* which is called from *extract_landmarks.*

```
face parfile opt=53 onlyedges=0 ltausedepth=0 showltas=1 spread=1 EdgeRadius=2 first=3 last=4
```

spread - radius of the box, *EdgeRadius* - the distance of the box center from the center edge of the arrangement. For storing the information in files do

```
mkdir LF
face parfile opt=44 onlyedges=0 merge=0 dataset=IM numdigits=num landdir=LF
```

See also script *getedges.*

Comparison arrays

The comparison arrays defined in section 6.3 are computed in *detect_masks* using a file *masks.asc* that has the masks used by the current model and the minimal thresholds for the *-1* and *+1* regions. The format for this file is defined through the *mask* class in *mask.h.* The file directory is given in *maskdir.* Also in this directory is a parameter file defining the graph to be used and the hard constraints on the angles and lengths in the triangles composing the graph. The format for this file is defined in the *graph* class in *graph.h.*

micro image codes

These are features used exclusively for recognition of binary images. See Chapter 9. The training procedure is described below. The quantization tree is stored in a *carttree* structure in a file called *codes.tree.* The micro image codes in the image are computed in *detecttags.* To store the codes in *Lands_ codes.tree* must be in the current directory. Then do

mkdir LF
detecttags dataset=IM numdigits=num newlanddir=LF

12.4 Deformable models

In each subsection is a list of the parameters used for a particular algorithm, some possible values, a short description and references to the notation used in the corresponding Chapter. Further information can be obtained in the documentation within each program. Each of these algorithms can either be run directly on an image as described in this section, or following a detection of the sparse model as described in the next section.

Deformable contours

These parameters refer to the deformable contour algorithms described in Chapter 3. The program file is *defcont.C.*

opt=113. Option number.

tempfile=snake.circ. File with a list of coordinates for model. If *tempfile=""* then the program prompts the user to point out the curve by hand and writes it to *defcont.temp.*

detfile=detout. File for writing out *detection* class with coordinates of final curve in the *locs* entry.

dumpdetections=1. Dump the detection info to command line.

curve_length=128. Length of curve to work with, can be larger or smaller than model, in which case the program interpolates to the desired size.

dataset=IM. The directory containing the *Images_* files.

inside=0. The average value of the inside $\mu_{in} \geq 0$. If *inside=-1* the online estimation of the parameters is enabled.

outside=.1. The average value outside μ_{out}.

dt_scale_factor=1. A scaling factor for the computed time step.

base=5. Wavelet basis to use in expanding curve - *base=1* - Haar, up to *base=6* smoothest.

numiters=10000. Number of iterations.

post=.3. Scaling of model curve.

minenergychange=.001. If change in coefficients is below this value add more coefficients for optimization or exit if at maximal number of coefficients.

FIRSTDIM=1. Initial number of coefficients updated $\mathcal{N}_1$, for each curve component.

LASTDIM=16. Final number of coefficients to update $\mathcal{N}_A$.

prior_fac=0. Factor multiplying the prior term.

first=3. Index number of first image to run.

imrange=.2. For graphic purposes, reduces gray value range of image display.

basecol=10. Color of the curve, *basecol=0* - red, *basecol = 10* white.

displayinter=100. Display result every 100 iterations.

point1x=79. Initial points, if *point1x=-1* [default] the user provides the initial point by pointing with the mouse.

point1y=67.

Deformable curves: dynamic programming

This algorithm is described in section 4.2. Only the simpler model of equation 4.9 which involves only two parameters p_o and p_b has been implemented. The program file is *curve_dyna.C.*

opt=112. Option number.

tempfile=dyna.escr. File name with coordinates of model curve. If *tempfile=""*, user is prompted to pick points on displayed image. When finished, type *c* inside image. Points will be written to default file *dyna.temp.*

detfile=detout. File for writing out *detection* class with coordinates of final curve in the *locs* entry.

dyna_dis=2. Distance of comparison pixels in the orthogonal direction to the curve for determining *ridg* type features. μ_a in equation 4.1. ν_a is taken to be half μ_a.

prior_angle=10. Factor on prior term penalizing changes in angle of segment relative to model curve. A in equation 4.10.

prior_length=30. Factor on prior term penalizing changes in length. B in equation 4.10.

dyna_regionsize=10. Radius of a square neighborhood around each pixel of model curve. This is the radius of each of the S_i neighborhoods. See section 4.2.

inside=.95. Probability of getting $X_a = 1$ for curve of angle a, i.e. p_o in equation 4.9.

outside=.05. Probability of getting $X_a = 1$ otherwise, i.e. p_b in equation 4.9.

dataset=IM. Data set with images to process.

first=3. First image to process.

revvideo=0. Reverse video graphic display.

dumpdetections=3. How much information to dump on screen and show. If value is 3, show each step of the dynamic programming. If value is 1 only show final result.

imrange=.3. Gray level value range.

basecol=10. Color for showing curve.

pixsize=2. Size of pixel in display.

point1x=43. Two initial points determining, location, rotation and scale of curve.

point1y=23.

point2x=43.

point2y=33.

Deformable curves: tree based algorithm

This algorithm is described in section 4.3. The data structure needed for this algorithm is very similar to the *tree* template except it is ternary, three children for each node and is defined in *curve_tree.h.* The algorithm is coded in *curve_tree.C.*

opt=117. Option number

dataset=IM. Directory with images.

inside=.7. Probability p_o of getting $X_a = 1$ on segment with angle a.

outside=.4. Probability p_b of getting $X_a = 1$ otherwise.

LEN=10. Length of each arc.

ORTHLEN. Length orthogonal to arc for testing μ_a (same as dyna_dis above).

deg=10. Degree of change allowed from one arc to the other (β).

ZSTAR=.999. Above which posterior to declare an arc on the track and restart algorithm.

mindepth=3. Minimum depth of arc with posterior above ZSTAR to allow reinitialization. (The top arcs quickly reach a posterior of over ZSTAR and are not so interesting).

numtrack=10. Number of times to reinitialize with high posterior arc.

numiters=10000. Maximal number of arc testings in each reinitialized run.

first=0. First image to process

last=1. Last image to process.

pixsize=2. Image magnification.

imrange=.3. Gray value range (imrange=0) maximal range.

basecol=10. Color of curve.

point1x=43. Two initial points determining first arc. If any of these is negative user is prompted to provide initial two points.

point1y=98.

point2x=41.

point2y=112.

Deformable images: Gaussian data model

This corresponds to the algorithm described in 5.3 based on the least squares data term, as well as the linearized algorithm described in section 5.5. These are implemented in *defimage_gaus.C.* Only a wavelet expansion of the deformation is coded. Because of this the dimensions of the image need to be a power of 2. If they are not, the program embedds the image in the smallest power of 2 larger than the dimensions.

*opt=111***.** Option number.

tempdataset=TEMPIM. Dataset with template

tempnum=45 Template image number

*dataset=IM***.** Dataset with images.

*first=123***.** Data image number

*post=1***.** Factor multiplying data term.

*prior_fac=.3***.** Factor multiplying prior term. Relative weighting of prior on coefficients λ_k is computed as in equation 5.6, with $\rho = 2$.

*minenergychange=.01***.** Minimum total change in coefficients to increase dimension $\mathcal{N}_a$ to $\mathcal{N}_{a+1}$ by factor of two or stop, if $a = A$.

*linear_elas=0***.** Regular deformation algorithm. *linear_elas=1*, uses iterations of linearization step described in section 5.5. Starts solving at FIRSTDIM and increases dimension at each iteration.

*FIRSTDIM=1***.** First level of coefficients to be updated. This is half of $\mathcal{N}_1$.

*LASTDIM=8***.** Last level of coefficients to be updated. This is half of $\mathcal{N}_A$.

*dt_scale_factor=1***.** Factor multiplying computed time step.

*numiters=1200***.** Maximum number of iterations.

*pixsize=4***.** Magnification

*displayinter=100***.** Display result every 100 iterations.

*arrow_spacing=2***.** Spacing of arrows in display of displacement field.

*imrange=.2***.** Range of gray levels.

*basecol=10***.** Color for displaying arrows.

Image warping with Bernoulli model

This implements the Bernoulli model for deformable images described in section 5.4.

Training

The probabilities of the edges on a reference grid of size *DIMX* by *DIMY* are saved in a file *edgetemps*. The training is implemented in routine *get_edgetemps* in *trainstat.C*, see script *getedgetemps*.

***opt=28*.**

***DIMX=64*.** Dimensions of the reference grid to which the extracted local features are registered.

***DIMY=64*.**

***point1x=25*.** Coordinates of three reference points p_1, p_2, p_3 to which the anchor points in each training image are mapped

***point1y=32*.**

***point2x=40*.**

***point2y=32*.**

***point3x=32*.**

***point3y=46*.**

***dataset=train*.** Dataset with training images.

***num=300*.** Number of training images in dataset

***rt=.4*.** Downsampling ratio to apply to each image (this is the ratio used for the faces.)

***onlyedges=1*.** Only use edge features. onlyedges=0 together with an *ltatreedir* will extract edge arrangements and then register. Edge parameters are below.

***EdgeEpsilon=.03*.**

***edgeinc=1*.**

***EdgeGootTests=4*.**

***EdgeNumberOfPixels=2*.**

***EdgeSizeBlock=1*.**

***spread=1*.** Radius of box into which each detected edge is spread.

***tempfile=edgetemps*.** File for writing out the frequencies.

***showtriangles=1*.** Show images with the frequencies.

Detection

Implemented in *defimage_ber.C*.

opt=115. Option number.

dataset=IM. Dataset with face images of more or less the correct size (DIMX by DIMY) with faces more or less centered so that the eyes are about 14 pixels apart. Significant variation is tolerated.

tempfile=edgetemps Name of probability map file produced in training.

minval=.3. Background edge probability. All areas where the edge probabilities p_b are less than *minval* are set to minval.

first=0. First image to process.

last=10. Last image to process.

eyex=32. Use these dimensions of the data around the middle point. Allows use of smaller windows for matching.

eyey=32.

shift=5. Range of shifts (+/-) in brute force search for optimal shift and scale.

lowscale=.6. Lower and upper bounds of scaling in brute force search for optimal scaling.

highscale=1.4.

spread=1. Amount of spread of detected edges in data.

dt_scale_factor=1. Factor multiplying computed time step for gradient descent stage.

minenergychange=.01 As in Gaussian model.

FIRSTDIM=1. First level of optimized coefficients. $\mathcal{N}_1 = 2 \times FIRSTDIM \times FIRSTDIM.$

LASTDIM=4. Last level of optimized coefficient. $\mathcal{N}_A = 2 \times LASTDIM \times LASTDIM.$

numiters=20. Number of iterations of gradient descent.

prior_fac=.01. Factor of prior term.

post=1. Factor of data term.

pixsize=1. Magnification

displayinter=10. Display result every 10 iterations.

imrange=0.. Gray level range.

basecol=10. Color for dots and lines.

12.5 Sparse models

Dynamic programming

There is no routine for identifying the comparison arrays and thresholds for the chosen landmarks on the training sample. All we provide is an existing model is encoded in two files *masks.asc* containing the masks being used with the corresponding thresholds and *tempfile* containing information on the graph and model locations. The algorithm is implemented in *sparse_dyna.C*. The corresponding scripts are to be found in directory *chap7* and the files in BRAIN/grmtch

opt=116.

maskdir=grmtch. Directory where *masks.asc* is found as well as the *tempfile*.

usemasks=1. If set to 0 - use ltas defined in *ltatreedir*

dataset=test. Images dataset.

first=2. First image

last=30. Last image

ratio=.5. Downsampling ratio to apply to image

pixsize=2.

dumpdetections=1. Show final result. 2 - show each step in dynamic programming. 3 - show template and locations of local features.

tempfile=PRMla.asc. File containing graph information.

imrange=.3.

basecol=10.

Counting detector - running an existing detector

A trained detector stored in directory *filt* consists of some subdirectories and one parameter file *pars*:

pars. All edge parameters.

ltatree1x, ltatree1y, ltatree2x, ltatree2y, ltatree3x, ltatree3y - coordinates of the anchor points.

hits, acchits. The thresholds τ and τ_e for Steps I and II of the counting detector.

numcarts=20. Number of classification trees - object against false positives. 0 - no classification.

fnumcarts=20. Number of classification trees among detected classes. 0 - no classification.

numratios=6. Number of resolutions at which to run detector.

testrotations=1/0. Whether to estimate rotations.

boxcluster=5. How close are two detected triangles (3 reference points) for them to be clustered.

ltatreedir=treesgood. Contains a file *ltatrees* with information regarding the LTA's and their coordinates in the reference grid.

edgetreedir=treesedge. Contains a file *ltatrees* with information regarding the edge model for Step II. This model is used to adjust scale if parameter *edgesforscale=1.* If *edgesforscale=0* the LTA's are used to adjust for scale and *treesedge* is not needed.

carttreedir=treescart. Contains classification trees to classify detections of Step I and II as object or not object. If *numcarts=0* no classification is performed and this directory is not needed.

carttreedir=facecart. Contains classification trees to classify detections of Step I and II among different classes If *fnumcarts=0* no classification is performed and this directory is not needed.

showtriangles=1. Show detection triangles. 2- Show only vertices of triangle.

dumdetections=1. Print out some information as detection procedes. 2 - Print out all information on all detections, dump the detection structures.

If there is a list of *pgm* files in file *image_list* and the detector parameters are in directory *filt* run.
test_pgm_list image_list filt first=3 last=20
If the images are in a directory *image_data* in *Images_* format
test_imaged 3 20 filt
will run the detector on images 3 through 20.

Running a deformable model initialized with a sparse model detection

The parameter *templatestyle*, if greater than 0, determines which deformable model is run. Depending on the model the appropriate parameters, defined earlier must be included in the parameter file.

templatestyle=1. Apply deformable contour directly on image, the pose $\mathbf{a}_p$ of the detection is applied to the template contour to produce initial contour.

templatestyle=2. Apply deformable curve dynamic programming algorithm directly on image, the pose $\mathbf{a}_p$ of the detection is applied to the template curve to produce initial curve.

templatestyle=3,4,5,6. The region of interest around the detection is registered to a reference grid. This reference grid could have different dimensions and the reference points could be different from those of the sparse model. The dimensions of the reference grid are set in *defsizex, defsizey* and the three reference points in the reference grid are given by *defpoint1x, defpoint1y*, *defpoint2x, defpoint2y*, *defpoint3x, defpoint3y.* The detection triangle *det.tripts* is mapped to these 3 points determining the affine map for registration.

The registered image is then processed with a deformable model as follows

templatestyle=3. Deformable image - Gaussian model.

templatestyle=4. Deformable image - Bernoulli model.

templatestyle=5. Deformable contour.

templatestyle=6. Deformable curve -dynamic programming.

12.6 Sparse model - counting detector -Training

Here we describe how to train models for Step I and Step II of the counting detector described in Chapters 6 and 8.

Assume we are in directory *object* with *numtr* training images in directory *train*, and *numte* training images of the object in directory *test.* These images are used to adjust the two thresholds τ and τ_a. In

both directories there is supposed to be a *coords* file with the coordinates of the three anchor points for each image, which are marked before hand, 3 rows per image, 2 coordinates per row. The training procedure is implemented with the script *doitob*. The parameters defining the edge detection are set in doitob and can be changed. Also two parameters defining the wedges for the edge arrangements. *numang=16* number of wedges. Angle of each wedge is 360/*numang* (see section 6.4.) *near=4* is the radius of the neighborhood containing the wedges. The script *doitob* is run from the command line with 7 arguments, for example:
doitob 32 20 3 2 3 .5 1

1. Number of training data
2. Number of LTA's to use in final model.
3. Complexity of edge arrangements, number of edges (n_r).
4. The features used for the classification trees of object against the false positives. 0 - normalized and registered gray levels, 2 - edges, 3 - edge arrangements.
5. Size of disjoint blocks in which only one edge arrangement is chosen. Typically 3×3.
6. In each block the highest frequency edge arrangement is found. It is taken only if its probability is higher than *objprop*. ρ - in algorithm 6.4.
7. By what factor to dowscale the training images before edge extraction.
8. If greater or equal to 0, the classification trees are made of this object class against false positives obtained by the detector. Otherwise, classification trees are made between the different class labels of the detections of the detector. The false positives for this step are found on images in directory *../trainall*, which contains training images from all object classes.

The steps in *doitob* are the following.

1. Write edge extraction parameters to *pars*
2. Obtain three reference points either from user written in *newbasis* or by taking averages of anchor points in training images.
3. Extract edges from data in *train*
4. Register the edge locations to reference grid, anchor points get mapped to reference points. Script - *register_edges.*
5. Find high probability edge locations (>*objprop*) and store them in *treesedge/ltatrees.* Assume *nume* are found. Script - *choose_edgelocs.*
6. Find high probability edge arrangement locations (>objprop) and store them in *treesgood/ltatrees.* Assume *numl* are found. Script - *make_ltas.*
7. Find thresholds for two steps of counting detector using test set. First find τ by starting at τ=*numl/2* and incrementing by one until a test point is missed. The estimated τ is assigned to variable *hits.* Write *hits* to *pars.* Script - *findhits.*
8. Then with this τ find maximal τ_a for Step II of counting detector in the same way. Assigned to variable *acchits.* Write *acchits* to pars. Script - *findacchits.*

9. (optional) Classification trees. Script *findnewp*.

   findnewp 0 cartstyle classlabel

   The parameters inside *findnewp* are described in section 12.10.

   (a) Extract detections from dataset *../trainall* using parameter file *pars*. The detection triangle is recorded in file *detcoords*. Each detection in an image is reported in one line with 8 numbers, 2 coordinates of 3 points, the image number and class number of that image. If no detections occur in the image, there is a line with -1's and the image number. Several detections are possible in one image.

   (b) Loop over images of *../trainall* extract data (gray level, edges, or edge arrangements) and register using affine map determined by detection triangle in *detcoords*. Data is stored in file *ftrraw*.

   (c) Read in data in *ftrraw* and train multiple randomized classification trees using the *carttree* structure.

       If these trees classify object against the rest of the world set *numcarts* to number of carttrees and write to pars. Store *carttrees* in directory *treescart*.

       If these trees classify among detected object classes set *fnumcarts* to number of carttrees and store trees in *facecart*.

12.7 Example - LaTeX

In directory *LATEX* the subdirectory *protos* contains images of the 293 LaTeX symbols. Synthetic deformations of any number of these are obtained using the parameters bellow. Script - *make_train*. These synthetic deformations also create a *coords* file mapping the three reference points marked on the prototype according to the generated random map.

opt=47.

dataset=protos.

ratio=1.0. What scaling to apply to each prototype.

first=0. First prototype to be deformed.

last=292. Last prototype to be deformed.

randomize=1. Produce random deformations.

rotate=.3. Sample uniformly from $\pm .3$ radians (about 18) degrees (0 - do not rotate.)

scale=3. Sample uniformly from log-scale $\pm(.5/3)$ giving approximately $\pm 20\%$ (0 - do not scale.)

skew=3. Sample uniformly from log-skew (ratio of x-scaling and y-scaling) $\pm(.5/3)$ giving approximately $\pm 20\%$ (0 - do not skew.)

numdigits=50. Number of random samples per prototype.

synthesize=1. Generate random non-linear deformations. Using random Gaussian wavelet coefficients with exponential decay in variances (0 - do not deform.)

base=3. Daubechies wavelet basis to use, the coefficients for 2-6 are stored in the program.

LASTDIM=4. $2 \times LASTDIM \times LASTDIM$ is the number of randomly generated coefficients to use in wavelet expansion.

latexdim. Dimension of image in which to put the randomly perturbed prototypes. Defaults is 32.

The images are written to the directory in which *face* is called. The above parameters would create 50 randomly deformed images for all 293 classes. These can be stored in a directory *trainall.*

The script *makedet* is called to create a directory for a particular class, create training and test sets for this class, train the detector using *doitob* and run the classification on false positive obtained from the dataset *../trainall* using *findnewp*, that is called from *doitob.*

makedet latex_n n 32 20 3 2 3 .5 1.

The parameters to *makedet* are:

1. Name of directory which contains the object information.
2. Index number of prototype from *protos* to use.
3. Number of synthetic training data to produce.
4. Maximal number of local features to create in model.
5. Complexity of local features (number of edges).
6. Type of features for the classification trees (0,2,3).
7. Size of disjoint regions in each of which at most one local feature is found.
8. Proportion of training data required to keep a local feature.
9. Scaling to apply to each training image before edge extraction.

The parameters determining the type of deformations to apply to create training and test sets are written in the file *makedet* and can be changed.

The steps taken in *makedet* are the following. Assume directory name is *object*

1. *mkdir object. cd object.*
2. Creates a *train* directory. *cd* to *train.* Apply *make_train. cd* back to *object*
3. Creates a *test* directory. *cd* to *test.* Applys *make_train. cd*'s back to *object*
4. Runs *doitob* with the parameters given to *makedet*

12.8 Other objects with synthesized training sets

We outline the steps for making object models for a collection of objects where we assume one image of each object is taken on a flat background. (This prevents training to incorporate background information in the object model.)

1. Make a base directory *objects.*

   mkdir objects
   cd objects

2. Take images of objects such as the clip or the chess pieces shown in Chapters 8, 10. Convert to *pic0.pgm, pic1.pgm ...picN.pgm* (assuming N objects).

3. Make a protos directory

   mkdir protos
   mv pic.pgm protos*
   cd protos

4. Convert to *Images_* format.

   pgmtoIm numdigits=N

5. Choose anchor points by clicking on three points in each prototype image with the left mouse button. The coordinates of the three points get stored in *coords*.

   face opt=40 numdigits=N dataset=./ >> coords

6. Go back to *objects* directory and make *trainall* directory.

   cd ..,

   and prepare a *trainall* directory with say 50 randomly deformed images of each prototype:

   mkdir trainall

   cd trainall

   make_train 50 first=0 last=N latexdim=40 dataset=../protos rotate=.3 skew=3

   cd ..

   For parameters of random deformations see above.

7. To make a model for object 'n' with 32 training images:

   makedet object_n n 32 20 3 2 3 .5 1.

12.9 Shape recognition

In *base/NIST* there is a subdirectory *trees* with 100 classification trees trained on the NIST data base with absolute arrangements. The directory *test* has 10,000 NIST test images on which to try the trees.
face par.online
will classifiy the 10,000 images using the trees, computing the landmarks on the fly.

Similarily in *base/LATEX/CLASS* is a subdirectory with trees made with relational arrangements. The data for testing can be made by going to subdirectory *test* and running *make_lands*. The data is classified by returning to *base/LATEX/CLASS* and running
face par.rel
To see some of the arrangements detected on an image and other images in the same terminal node run
face par.show.

Extracting features

The first step is creating the feature files for the training data and the test data.
cd train (or cd test)
face pars opt=44

- Generic edge based features.

  The file *pars* should contain the parameters defining the type of features.

  ratio=1. What downscaling to apply to image before feature extraction.

  onlyedges=1 Extract only edges. 0 - Extract edge arrangements defined in *base/all.*

  merge=1 Merge the edge features with the edge arrangements.

  ltatreedir=base/all Directory with definition of predetermined collection of edge arrangements.

  dataset=./

  numdigits=10 Number of data points.

  EdgeParameters: See above for parameters defining the type of edges being extracted.

- Micro image codes.

  First a cart tree describing quantization of the micro-images must be made. It is written to *codes.tree.* The training images are assumed binary. For the NIST data set they are stored with 1 bit per pixel and *datadepth* must be set to 1.

  opt=48

  datadepth=1 For NIST image set to 1.

  codesize=4 Size of subimages to code.

  maxdepth=5 Maximum depth of tree.

  numdigits=100 Number of images from which to extract random *codesize* subimages.

  numperts=5 Number of random subimages to extract from each image.

  nist=1 Preprocess binary image:
  - Run median filter to eliminate little spots.
  - Blur image with Gaussian filter.
  - If any dimension of image is greater than *scale* downsample to size *scale.* Default value for scale is 32.
  - Threshold image at fraction *threshold* of maximal value. Default value is .2.

  For extracting code labels for all subimages of all training images.

  opt=1000

  numltas=62 Total number of microcodes.

  datadepth=1 Binary images.

  ltacluster=6 Size of blocks for clustering features of the same type.

Training and testing trees

Assume the training landmark files are in subdirectory *train*

opt=148.

nist=48. Class labels of NIST data are in ascii code so need to substract 48.

althistsize=10. How many classes to keep in terminal node. This is useful for problems with many class (i.e. LaTeX). Stores only top *althistsize* classes in the terminal node.

numcarts=100. Number of trees to grow.

class_tree=2. Absolute arrangements. 1 - Relative arrangements.

cartsizex=32. Dimension of grid on which absolute locations are defined. Must be the same for all if *class_tree=2* (absolute arrangements). cartsizey=32

cartspread=1. If absolute arrangements are used locations of features can be downsampled to coarser grid by factor *cartspread.* Warning *cartspread*cartsizex* should equal true dimension of images.

near=8. Radius of box of spread for each feature, $\mathbf{s} = 2 * (near - 1) + 1$.

numland=62. Number of local feature types.

maxdepth=20. Maximum depth of the tree.

numtreedata=100000. Number of data to use.

landdir=train. Directory with landmark data.

numclass=10. Number of classes.

numquesask=200. Number of random questions to sample at each node.

mindata=0. Minimum amount of data for splitting a node.

modetwo=3. Minimum amount of data at second largest class to stop splitting.

boost=1. Use boosting.

alpha=1. Extra factor to multiply boosting factor.

carttreedir=./. Directory to write the trees.

useques(5)=1. In case of relational trees use questions 5 and 6. 5 - asks for the first two features with a relation. 6 - asks for an additional feature in relation to one existing in the pending graph.

useques(6)=1.

Testing is achieved as follows

opt=147. 147 - loops through test points and then through all the trees, and reports aggregate classification rate at the end. 149 - Loops by tree and reports aggregate rate after each tree.

nist=48.

numtreedata=10000. Number of test data points.

landdir=test. Directory with landmark data.

numclass=10. Number of classes.

carttreedir=./. Directory with trees.

12.10 Combining detection and recognition

The same scripts used above for creating a detector for a particular class and classifiers for the object against false positives obtained from *trainall* will produce classifiers among the classes detected by the detector. The only change is to run
findnewp 0 cartstyle -1 and set the parameters inside *findnewp*.

nd=3100. Number of data to use from *trainall*

ncl=62. Total number of classes.

npcl=50. Number of elements per class (usually 50).

acf=.8. The factor to apply to the threshold *hits* to lower it and thereby obtain more detections in other classes.

numcarts=50. Number of classification trees.

cartdepth=10. Maximal depth of a trees.

nq=200. Number of random questions to sample.

crtsp=2. The radius of spreading for a binary feature used in the classification trees.

The program creates a file *good* showing how many of each class are detected. In order for the program to know how to label detected and classified objects a file with labels for the classes must be appended to *pars*
facenames(0) = marx
facenames(1) = lenin.

When classification trees for multiple classes are produced they are written to a directory called *facecart*, and are used if the parameter *fnumcarts* is positive. Otherwise they are ignored and regular un-classified detection is performed.

Bibliography

Abbott, L. F. & Nelson, S. B. (2000), 'Synaptic plasticity: taming the beast', *Nature Neuroscience* **3**, 1178–1183. Supplement.

Amit, D. & Brunel, N. (1995), 'Learning internal representations in an attractor neural network with analogue neurons', *Network* **6**, 261.

Amit, D. J. (1989), *Modelling Brain Function: The world of attractor neural networks*, Cambridge University Press.

Amit, D. J. & Brunel, N. (1997), 'Model of global spontaneous activity and local structured (learned) delay activity during delay periods in cerebral cortex', *Cerebral Cortex* **7**, 237–252.

Amit, D. J. & Fusi, S. (1994), 'Dynamic learning in neural networks with material synapses', *Neural Computation* **6**, 957.

Amit, Y. (1994), 'A non-linear variational problem for image matching', *SIAM Journal on Scientific Computing* **15**(1), 207–224.

Amit, Y. (1997), 'Graphical shape templates for automatic anatomy detection: application to mri brain scans', *IEEE Trans. Medical Imaging* **16**, 28–40.

Amit, Y. (2000), 'A neural network architecture for visual selection', *Neural Computation* **12**, 1059–1082.

Amit, Y. & Blanchard, G. (2001), Multiple randomized classifiers: MRCL, Technical report, Dept. of Statistics, University of Chicago.

Amit, Y. & Geman, D. (1997), 'Shape quantization and recognition with randomized trees', *Neural Computation* **9**, 1545–1588.

Amit, Y. & Geman, D. (1999), 'A computational model for visual selection', *Neural Computation* **11**, 1691–1715.

Amit, Y. & Kong, A. (1996), 'Graphical templates for model registration', *IEEE PAMI* **18**, 225–236.

Amit, Y. & Mascaro, M. (2001), 'Attractor networks for shape recognition', *Neural Computation* **13**, 1415–1442.

Amit, Y., Geman, D. & Jedynak, B. (1998), Efficient focusing and face detection, *in* H. Wechsler & J. Phillips, eds, 'Face Recognition: From Theory to Applications, NATO ASI Series F', Springer-Verlag, Berlin.

Amit, Y., Geman, D. & Wilder, K. (1997), 'Joint induction of shape features and tree classifiers', *IEEE Trans. on Patt. Anal. and Mach. Intel.* **19**, 1300–1306.

Amit, Y., Grenander, U. & Piccioni, M. (1991), 'Structural image restoration through deformable template', *Journal of the American Statistical Association* **86**(414), 376–387.

Arbter, K., Snyder, W. E., Burkhardt, H. & Hirzinger, G. (1990), 'Application of affine invariant fourier descriptors to recognition of 3d objects', *IEEE Trans. PAMI* **12**, 640–647.

Atkinson, R. (1975), 'Mnemotechnics in second-language learning', *American Psycholigsts* **30**, 821–828.

Bajcsy, R. & Kovacic, S. (1988), 'Multiresolution elastic matching', *Computer Vision, Graphics, and Image Processing* **46**, 1–21.

Ballard, D. H. (1981), 'Generalizing the hough transform to detect arbitrary shapes', *Pattern Recognition* **13**, 111–122.

Bartlett, M. S. & Sejnowski, T. J. (1998), 'Learning viewpoint-invariant face representations from visual experience in an attractor network', *Network: Comput. Neural. Syst.* **9**, 399–417.

Bertele, U. & Brioschi, F. (1969), 'A new algorithm for the solution of the secondary optimization problem in non-serial dynamic programming', *J. Math. Anal. Appl.* **27**, 565–57.

Biederman, I. (1995), Visual object recognition, *in* S. M. Kosslyn & D. N. Osherson, eds, 'Visual Cognition', MIT Press, Cambride, Mass., pp. 121–166.

Bienenstock, E., S., G. & Potter, D. (1997), Compositionality, mdl priors, and object recognition, *in* M. C. Mozer, M. I. Jordan & T. Petsche, eds, 'Advances in Neural Information and Processing Systems', Vol. 9, MIT Press, Cambridge, Mass., pp. 834–844.

Binford, T. O. & Levitt, T. S. (1993), Quasi-invariants: Theory and exploitation, *in* 'Proc. Image Understanding Workshop', Washington D.C., pp. 819–828.

Bishop, C. M. (1995), *Neural Networks for Pattern Recognition*, Oxford University Press, New York.

Blake, A. & Issard, M. (1998), *Active Contours*, Springer-Verlag, New York.

Blake, A. & Yuille, A. (1992), *Active Vision*, MIT Press, Cambridge, Massachusetts.

Bliss, T. V. P. & Collingridge, G. L. (1993), 'A synaptic model of memory: long term potentiation in the hippocampus', *Nature* **361**, 31.

Bookstein, L. F. (1991), *Morphometric Tools for Landmark Data : Geometry and Biology*, Cambridge University Press, Cambridg.

Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Jackel, L. D., LeCun, Y., Muller, U. A., Sackinger, E., Simard, P. & Vapnik, V. (1994), Comparison of classifier methods: a case study in handwritten digit recognition, *in* 'Proc. IEEE Inter. Conf. on Pattern Recognition', pp. 77–82.

Breiman, L. (1994), Bagging predictors, Technical Report 451, Department of Statistics, University of California, Berkeley.

Breiman, L. (1998), 'Arcing classifiers (with discussion)', *The Annals of Statistics* **26**, 801–849.

Breiman, L., Friedman, J., Olshen, R. & Stone, C. (1984), *Classification and Regression Trees*, Wadsworth, Belmont, CA.

Brooks, R. A. (1981), 'Symbolic reasoning among 3d models and 2d images', *Artificial Intelligence* **17**, 285–348.

Brunel, N., Carusi, F. & Fusi, S. (1998), 'Slow stochastic hebbian learning of classes of stimuli in a recurrent neural network', *Network* **9**, 123–152.

Burl, M. C., Leung, T. K. & Perona, P. (1995), Face localization via shape statistics, *in* M. Bichsel, ed., 'Proc. Intl. Workshop on Automatic Face and Gesture Recognition', pp. 154–159.

Burl, M., Weber, M. & Perona, P. (1998), A probabilistic approach to object recognition using local photometry and global geometry, *in* 'Proc. of the 5th European Conf. on Computer Vision, ECCV 98', pp. 628–641.

Camion, V. & Younes, L. (2001), Geodesic interpolating splines, *in* 'EEMVCPR'.

Caselles, V., Kimmel, R. & Sapiro, G. (2000), 'Aeodesic active countuors', *to appear International Journal of Computer Vision*.

Caselles, V., Kimmel, R., Sapiro, G. & Sbert, C. (1997), 'Minimal surfaces based object segmentation', *IEEE PAMI* **19**, 394–398.

Chalidabhongse, J. & Kuo, C.-C. J. (1997), 'Fast motion vector estimation using multiresolution-spatio-temporal correlations', *IEEE Trans. on Circuits and Systems for Video Technology* **7**, 477–488.

Chelazzi, L., Miller, E. K., Duncan, J. & Desimone, R. (1993), 'A neural basis for visual search in inferior temporal cortex', *Nature* **363**, 345–347.

Chesnaud, C., Réfrégier, P. & Boulet, V. (1999), 'Statistical region snake-based segmentation adapted to different physical noise models', *IEEE-PAMI* **21**, 1145–1157.

Christensen, G., Rabbitt, R. D. & Miller, M. I. (1996), 'Deformable templates using large deformation kinematics', *IEEE Transactions on Image Processing* **5**, 1435–1447.

Chuang, G. & Kuo, C. (1996), 'Wavelet description of planar curves: theory and applications', *IEEE trans. on Image Processing* **5**, 56–70.

Cohen, I. & Cohen, L. D. (1993), 'Finite element methods for active contour models and balloons for 2d and 3d images', *IEEE Trans. on Patt. Anal. and Mach. Intel.* **15**, 1131–1147.

Cohen, I., Cohen, L. D. & Ayache, N. (1992), 'Using deformable surfaces to segment 3-d images and infer differential structures', *CVGIP: Image Understanding* **56**(2), 242–263.

Cohen, L. D. (1991), 'On active contour models and balloons', *CVGIP: Image Understanding* **53**, 211–218.

Cootes, T. F. & Taylor, C. J. (1992), Active shape models - smart snakes, *in* 'Proceedings of BMVC', pp. 267–275.

Cootes, T. F. & Taylor, C. J. (1996), Locating faces using statistical feature detectors, *in* 'Proc., Second Intl. Workshop on Automatic Face and Gesture Recognition', IEEE Computer Society Press, pp. 204–210.

Cover, T. M. & Thomas, J. A. (1991), *Elements of Information Theory*, John Wiley, New York.

Daubechies, I. (1988), 'Orthonormal bases of compactly supported wavelets', *Comm. Pure and Appl. Math.* **XLI**, 909–996.

Desimone, R. & Duncan, J. (1995), 'Neural mechanisms of selective visual attention', *Annu. Rev. Neurosci.* **18**, 193–222.

Desimone, R., Miller, E. K., Chelazzi, L. & Lueschow, A. (1995), Multiple memory systems in visual cortex, *in* M. S. Gazzaniga, ed., 'The Cognitive Neurosciences', MIT Press, Cambridge, Massachusetts, pp. 475–486.

Diederich, S. & Opper, M. (1987), 'Learning correlated patterns in spin-glass networks by local learning rules', *Phys. Rev. Lett.* **58**, 949–952.

Dryden, I. L. & Mardia, K. V. (1998), *Statistical Shape Analysis*, Wiley.

Duda, R. O. & Hart, P. E. (1973), *Pattern Classification and Scene Analysis*, John Wiley, New York.

Durtewitz, D., Seamans, J. K. & Sejnowski, T. J. (2000), 'Neurocomputational models of working memory', *Nature Neuroscience* **3**, 1192–1198. supplement.

Elder, J. & Zucker, S. W. (1996), Computing contour closure, *in* 'Computer Vision - ECCV', Springer, New York, pp. 399–412.

Figueiredo, M. & Leitao, J. (1992), 'Bayesian estimation of ventricular contours in angiographic images', **11**, 416–429.

Fischler, M. A. & Elschlager, R. A. (1973), 'The representation and matching of pictorial structures', *IEEE Trans. on Computers* **22**, 67–92.

Fleuret, F. (2000), Détection hiérarchique de visages par apprentissage statistique, PhD thesis, Lúniversite Paris 6.

Fleuret, F. & Geman, D. (2001), 'Coarse-to-fine face detection', *International Journal of Computer Vision* **41**, 85–107.

Forsyth, D., Mundy, J. L., Zisserman, A., Coelho, C., Heller, A. & Rothwell, C. (1991), 'Invariant descriptors for 3-d object recognition and pose', *IEEE Trans. PAMI* **13**, 971–991.

Freund, Y. & Shapire, R. E. (1997), 'A decision-theoretic generalization of on-line learning and an application to boosting', *Journal of computer and system sciences* **55**, 119–139.

Friedman, J., Hastie, T. & Tibshirani, R. (1998), Additive logistic regression: a statistical view of boosting, Technical report, Department of Statistics, Stanford University.

Friston, K., Ashburner, J., Frith, C. D., Poline, J.-B., Heather & J.D., Frackowiak, R. (1995), 'Spatial registration and normalization of images', *Human Brain Mapping* **3**, 165–189.

Fukushima, K. (1986), 'A neural network model for selective attention in visual pattern recognition', *Biol. Cyber.* **55**, 5–15.

Fukushima, K. & Wake, N. (1991), 'Handwritten alphanumeric character recognition by the neocognitron', *IEEE Trans. Neural Networks.*

Fusi, S., Annunziato, M., Badoni, D., Salamon, A. & Amit, D. J. (2000), 'Spike-driven synaptic plasticity: theory, simulation, vlsi implementation', *Neural Computation* **12**, 2227.

Garris, M. D. & Wilkinson, R. A. (1996), *NIST special database 3. Handwritten segmented characters*, NIST, Gaithersburg, MD.

Geiger, D., Gupta, A., Costa, L. & Vlontzos, J. (1995), 'Dynamic programming for detecting, tracking and matching defomrable contours', **17**, 294–302.

Geman, D. (1990), *Random fields and inverse problems in imaging*, number 1427 *in* 'Lecture Notes in Mathematics', Springer Verlag.

Geman, D. & Jedynak, B. (1996), 'An active testing model for tracking roads from satellite images', *IEEE Trans. PAMI* **18**, 1–15.

Geman, S. & Geman, D. (1984), 'Stochastic relaxation, gibbs distributions, and the bayesian restoration of images', *IEEE Trans. PAMI* **6**, 721–741.

Geman, S., Potter, D. F. & Chi, Z. (to appear 2002), 'Composition systems', *Quarterly of Applied Mathematics*.

Gersho, A. & Gray, R. M. (1992), *Vector Quantization and Signal Compression*, Kluwer Academic, Boston.

Gold, S. & Rangarajan, A. (1996), 'A graduated assignment algorithm for graph matching', *IEEE PAMI* **18**, 377–388.

Grenander, U. (1970), 'A unified approach to pattern analysis', *Adv. Comput.* **10**, 175–216.

Grenander, U. (1978), *Pattern Analysis: Lectures in Pattern Theory I-III*, Springer Verlag, New York.

Grenander, U. (1993), *General Pattern Theory*, Oxford University Press, Oxford.

Grenander, U. & Miller, I. M. (1998), 'Computational anatomy: an emerging discipline', *Quarterly of Applied Mathematics* **LVI**(4), 617–694.

Grenander, U., Chow, Y. & Keenan, D. (1991), *A Pattern Theoretical Study of Biological Shape*, Springer Verlag, New York.

Grimson, W. E. L. (1990), *Object Recognition by Computer: The Role of Geometric Constraints*, MIT Press, Cambridge, Massachusetts.

Hallinan, P. L., Gordon, G. G., L., Y. A., Giblin, P. & Mumford, D. (1999), *Two- and Three- Dimensional Patterns of the Face*, A. K. Peters, Natick, MA.

Haralick, R. M. & Shapiro, G. L. (1992), *Computer and Robot Vision*, Vol. 1,2, Addison Wesley, Reading, MA.

Hastie, T. & Simard, P. Y. (1998), 'Metrics and models for handwritten character recognition', *Statistical Science*.

Hastie, T., Buja, A. & Tibshirani, R. (1995), 'Penalized discriminant analysis', *Annals of Statistics* **23**, 73–103.

Hebb, D. O. (1949), *The Organization of Behavior*, Wiley, New York.

Hedgé, J. & Van Essen, D. C. (2000), 'Selectivity for complex shapes in primate visual area v2', *The Journal of Neuroscience*.

Hermosillo, G., Chefd'Hotel, C. & Faugeras, O. (2001), A variational approach to multi-modal image matching, Technical Report RR 4117, INRIA.

Hinton, G. E., Dayan, P., Frey, B. J. & Neal, R. (1995), 'The wake-sleep algorithm for unsupervised neural networks', *Science* **268**, 1158–1161.

Ho, T. K., Hull, J. J. & Srihari, S. N. (1994), 'Decision combination in multiple classifier systems', *IEEE Trans. PAMI* **16**, 66–75.

Hopfield, J. J. (1982), 'Neural networks and physical systems with emergent selective computational abilities', *Proc. Natl. Acad. Sci., USA*.

Horn, B. K. P. & Schunck, B. G. (1981), 'Determining optical flow', *Artificial Intelligence* **17**, 185–203.

Hough, P. V. C. (1962), 'Methods and means for recognizing complex patterns', *U.S. Patent, 3069654*.

Huang, T. S. & Tsai, R. Y. (1981), Image sequence analysis: Motion estimation, *in* T. S. Huang, ed., 'Image Sequence Analysis', Springer-Verlag, NewYork.

Hubel, H. D. (1988), *Eye, Brain, and Vision*, Scientific American Library, New York.

Ishikawa, H. & Geiger, D. (1998), Segmentation by grouping junctions, *in* 'Proc. of the IEEE Comp. Vision and Pattern Recognition,'.

Ishikawa, H. & Geiger, D. (1999), Mapping image restoration to a graph problem, *in* 'Proc. IEEE-EURASIP Workshop on Non-linear and Signal and Image Processing'.

Jermyn, I. & Ishikawa, H. (1999), Globally optimal regions and boundaries, *in* 'Proc. of 7'th IEEE Intl. Conference on Computer Vision (ICCV'99)'.

Joshi, S. (1997), Large Deformation Diffeomorphisms and Gaussian Random Fields for Statistical Characterization of Brain Submanifolds, PhD thesis, Dept. of Electrical Engineering, Washington University.

Kass, M., Witkin, A. & Terzopoulos, D. (1987), 'Snakes: active contour models', *International Journal of Computer Vision* pp. 321–331.

Kim, B., Boes, J. L., Frey, K. A. & Meyer, C. R. (1997), 'Mutual information for automated unwarping of rat brain autoradiographs', *NeuroImage* **5**, 31–40.

Kohonen, T. (1984), *Self-organization and Associative memory*, Springer Verlag, Berlin.

Kwok, S. W. & Carter, C. (1990,), Multiple decision trees, *in* R. D. Shachter, T. S. Levitt, L. Kanal & J. F. Lemmer, eds, 'Uncertainty and Artificial Intelligence', Elsevier Science Publishers, North-Holland, Amsterdam.

Lambdan, Y., Schwartz, J. T. & Wolfson, H. J. (1988), Object recognition by affine invariant matching, *in* 'IEEE Int. Conf. Computer Vision and Pattern Rec.', pp. 335–344.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998), 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* **86**(11), 2278–2324.

Levenex, P. & Schenk, F. (1997), 'Olfactory cues potentiate learning of distant visuospatial information', *Neurobiology of Learning and Memory* **68**, 140–153.

Malik, J. & Perona, P. (1990), 'Preattentive texture discrimination with early vision mechanisms', *Journal of the Optical Society of America - A* **7**, 923–932.

Malladi, R., Sethian, J. A. & Vemuri, B. C. (1995), 'Shape modeling with front propagation', *IEEE PAMI* **17**, 158–176.

Mallat, S. (1989), 'A theory for multiresolution signal decomposition: the wavelet representation', *IEEE Trans. Pattern Anal. and Machine Intell.* pp. 674–693.

Markram, H., Lubke, J., Frotscher, M. & Sakmann, B. (1997), 'Regulation of synaptic efficacy by coincidence of postsynaptic ap's and epsp's', *Science* **375**, 213.

Marr, D. (1982), *Vision*, W. H. Freeman and Company, New York.

Marr, D. & Hilderith, E. (1980), 'Theory of edge detection', *Proc R. Soc. Lond. B* **207**, 187–217.

Marr, D. & Nishihara, H. K. (1978), 'Representation and recognition of the spatial organization of three-dimensional shapes', *Proc. R. Soc. Lond. B Biol. Sci.* **200**, 269–294.

Mascaro, M. & Amit, D. J. (1999), 'Effective neural response function for collective population states', *Network* **10**, 351–373.

Mattia, M. & Del Giudice, P. (1999), Asynchronous simulation of large networks of spiking neurons and dynamical synapses, Submitted for publication to Neural Computation.

Meyer, Y. (1990), *Ondelettes et Operateurs*, Herman, Paris.

Miller, M., Christensen, G., Amit, Y. & Grenander, U. (1993), 'A mathematical textbook of deformable neuro-anatomies', *Proc. of the National Academy of Science* **R90**, 11944–11948.

Minsky, M. & Papert, S. (1969), *Perceptrons*, MIT Press, Cambridge, Mass.

Mumford, D. (1994), Pattern theory: a unifying perspective, *in* 'First European Congress of Mathematics, Vol 1.', Birkhausër, pp. 187–224.

Mundy, J. L. & Zisserman, A. (1992), *Geometric Invariance in Computer Vision*, MIT Press, Cambridge.

Nagel, H. H. (1983), 'Displacement vectors derived from second-order intensity variations in image sequences', *Computer Vision, Graphics and Image Processing* **21**, 85–117.

Nagy, G. (2000), 'Twenty years of document analysis in pami', *IEEE PAMI* **22**, 38–62.

Oja, E. (1989), 'Neural networks, principle components, and subspaces', *International Journal of Neural Systems* **1**, 62–68.

Olshausen, B. A., Anderson, C. H. & Van Essen, D. C. V. (1993), 'A neurobiological model of visual attention and invariant pattern recognition based on dynamic routing of information', *Journal of Neuroscience* **13**, 4700–4719.

Parida, L., Geiger, D. & Hummel, R. (1998), 'Junctions: Detection, classification, and reconstruction.', *IEE PAMI*.

Petrocelli, R. R., Elion, J. L. & Manbeck, K. M. (1992), A new method for structure recognition in unsubstracted digital angiograms, *in* 'Proceedings of Computers in Cardiology', IEEE Computer Society, pp. 207–210.

Plamondon, R. & Srihari, S. N. (2000), 'On-line and off-line handwritten recognition', *IEEE PAMI* **22**, 63–84.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. (1995), *Numerical Recipes in C, The Art of Scientific Computing*, 2nd edn, Cambridge University Press, Cambridge.

Quinlan, J. R. (1986), 'Induction of decision trees', *MachineLearning* **1**, 81–106.

Rabiner, L. & Juang, B.-H. (1993), *Fundamentals of Speech Recognition*, Prentice Hall, Englewood Cliffs, NJ.

Rangarajan, A., Chui, H. & Bookstein, F. (1997), The softassign procrustes matching algorithm, *in* J. Duncan & G. Gindi, eds, 'Information Processing in Medical Imaging', Springer, pp. 29–42.

Reiss, T. H. (1993), *Recognizing Planar Objects Using Invariant Image Features*, Lecture Notes in Computer Science no. 676, Springer Verlag, Berlin.

Revow, M., Williams, C. K. I. & Hinton, G. E. (1996), 'Using generative models for handwritten digit recognition', *IEEE PAMI* **18**, 592–606.

Rice, J. A. (1995), *Mathematical Statistics and Data Analysis*, 2nd edn, Duxbury Press, Belmont, California.

Riesenhuber, M. & Poggio, T. (1999), 'Hierarchical models of object recognition in cortex', *Nature Neuroscience* **2**, 1019–1025.

Riesenhuber, M. & Poggio, T. (2000), 'Models of object recognition', *Nature Neuroscience* **3**, 1199–1204. Supplement.

Ripley, B. D. (1994), 'Neural networks and related methods for classification', *J. Royal Statist. Soc., B* **56**, 409–437.

Rojer, A. S. & Schwartz, E. L. (1992), A quotient space hough transform for scpae-variant visual attention, *in* G. A. Carpenter & S. Grossberg, eds, 'Neural Networks for Vision and Image Processing', MIT Press.

Rolls, E. T. (2000), 'Memory systems in the brain', *Annu. Rev. Psychol.* **51**, 599–630.

Rose, D. J., Tarjan, R. E. & Leuker, G. S. (1976), 'Algorithmic aspects of vertex elimination on graphs', *Siam J. Comput.* pp. 266–283.

Rowley, H. A., Baluja, S. & Kanade, T. (1998), 'Neural network-based face detection', *IEEE Trans. PAMI* **20**, 23–38.

Sandor, S. E. & Leahy, R. M. (1995), Towards automated labelling of the cerebral cortex using a deformable atlas, *in* Y. e. a. Bizais, ed., 'Information Processing in Medical Imaging', Kluwer Academic Press, Netherlands, pp. 127–138.

Schapire, R. E., Freund, Y., Bartlett, P. & Lee, W. S. (1998), 'Boosting the margins: a new explanation for the effectiveness of voting methods', *Annals of Statistics* **26**(5), 1651–1686.

Schapire, R. E., Fruend, Y., Bartlett, P. & Lee, W. S. (1998), 'Boosting the margin: a new explanation for the effectiveness of voting methods', *The Annals of Statistics* **26**, 1651–1686.

Shapiro, L. G. (1980), 'A structural model of shape', *IEEE PAMI* **2**, 111–126.

Shi, J. & Malik, J. (2000), 'Normalized cuts and image segmentation', *IEEE PAMI* **22**, 888–905.

Simard, P. Y., LeCun, Y., Denker, J. S. & Victorri, B. (2000), 'Transformation invariance in pattern recognition – tangent distance and tangent propagation', *International Journal of Imaging Systems & Technology.*

Sung, K. K. & Poggio, T. (1998), 'Example-based learning for view-based face detection', *IEEE Trans. PAMI* **20**, 39–51.

Tanaka, K., Saito, H. A., Fukada, Y. & Moriya, M. (1991), 'Coding visual images of objects and the inferotemporal cortex of the macaque monkey', *Jour. Neuroscience* **66**(1), 170–189.

Tarr, M. & Bülthoff, H. (1998), 'Image-based object recognition in man, monkey and machine', *Cognition* **67**, 1–20.

Terzopolous, D., Platt, J., Barr, A. & Fleisher, K. (1987), 'Elastically deformable models', *Comp. Graph.* **21**, 205–214.

Tovee, M. J. (1996), *An introduction to the visual system*, Cambridge University Press, Cambride.

Trouvé, A. (1998), 'Diffeomorphism groups and pattern matching in image analysis', *International Jounral of Computer Vision.*

Ullman, S. (1996), *High-Level Vision*, M.I.T. Press, Cambridge, MA.

Van Rullen, R., Gautrais, J., Delorme, A. & Thorpe, S. (1998), 'Face processing using one spike per neuron.', *Biosystems* **48**, 229–239.

Vapnik, V. N. (1995), *The Nature of Statistical Learning Theory*, Springer Verlag, New York.

Viola, P. & Jones, M. J. (2002), 'Robust real time object detection', *Intl. Jour. Comp. Vis.*

Viola, P. & Wells, W. M. I. (1997), 'Alignment by maximization of mutual information', *IJCV* **24**, 137–154.

von der Heydt, R. (1995), Form analysis in visual cortex, *in* M. S. Gazzaniga, ed., 'The Cognitive Neurosciences', MIT Press, Cambridge, Massachusetts, pp. 365–382.

Wang, S. C. (1998), A statistical model for computer recognition of sequences of handwritten digits, with applications to zip codes, PhD thesis, University of Chicago.

Wang, Y. & Staib, L. H. (2000), 'Boundary finding with prior shapes and smoothness models', *IEEE PAMI* **22**, 738–743.

Wickerhauser, M. V. (1994), *Adapted Wavelet Analysis from Theory to Software*, IEEE Press, Wellesley, Massachusetts.

Wiskott, L., Fellous, J.-M., Kruger, N. & von der Marlsburg, C. (1997), 'Face recognition by elastic bunch graph matching', *IEEE Trans. on Patt. Anal. and Mach. Intel.* **7**, 775–779.

Zeki, S. (1993), *A Vision of the Brain*, Blackwell Scientific Publications, Oxford.

Zhu, S. & Yuille, A. (1996), 'Region competition: Unifying snakes, region growing, energy/bayes/mdl for multi-band image segmentation', *IEEE Trans. on Patt. Anal. and Mach. Intel.* **18**, 884–900.

Zhu, S. C. & Mumford, D. (1997), 'Prior learning and gibbs reaction-diffusion', *IEEE. Trans. PAMI* **19**, 1236–1250.

Index