CSE 252C: Computer Vision III

Lecturer: Serge Belongie
Scribes: Andrew Rabinovich and Vincent Rabaud
Edited by: Catherine Wah

LECTURE 13
Matching with Constellations of Parts

13.1. Constellation based models

The last two lectures covered bag of features, which had distinguished features but no location/geometric information, and shape matching, which used undistinguished features (*e.g.* equally spaced samples along contours) and strong geometric information (*e.g.* least squares estimates of the regularized TPS transform).

Now we will examine a class of approaches that fall somewhere in between; these methods are sometimes called "constellation based" or "parts based models," in which the model accounts for two key properties:

- relative locations between parts
- appearance of parts

Some of the questions we have to answer:

- What is a "part," and how do we find it?
- How do we model location?
- How do we handle occlusions/clutter?

[1]Department of Computer Science and Engineering, University of California, San Diego.

December 31, 2009

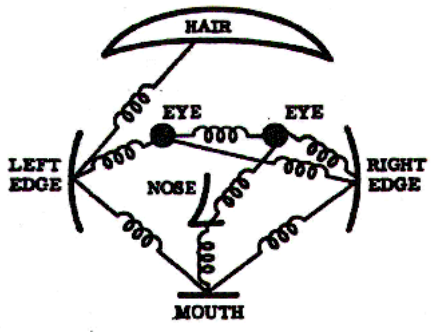

Figure 1. Description of a human face with pictorial structures (Fischler & Elschlager, 1973).

This part-based model matching is a highly active area in computer vision, with roots going back at least as far as Fischler & Elschlager's classic work on "the representation and matching of pictorial structures" (1973) (Figure 1). As an introduction to this type of approach, we will study a method developed by Geman, Amit, & Wilder (1997) and applied to digit recognition.

As we've seen, the use of small bitmaps (binary) of digits simplifies the image processing problems, and allows us to focus on other matters pertaining to recognition (*e.g.* using SIFT on a 28×28 image would be overkill). The method of Geman *et al.* is based on a specific kind of decision tree that examines arrangements of small (4×4) tags. The arrangements encode loose relative position information, allowing for invariance to substantial affine/non-linear deformation.

13.1.1. Tag creation

We'll start by looking at how the features (or "tags") are extracted. They look like oriented edge fragments, but they're simpler, in that they exploit the binary nature of the pixels in the digit images.

The following is done to create the tags:

(1) Extract a large sample of 4×4 sub-images from the dataset at random (without regard for class labels, which in this case are "0," "1,", ..., "9").
(2) Cluster these sub-images using a decision tree. Each node considers 16 possible questions: "is site (i, j) black?" for $i, j = 1, 2, 3, 4$. The question chosen is the one that most nearly splits the set of subimages in half. There is a tag type for each node of the resulting tree, except for the root. For example, a depth 3 tag tree, has $2 + 4 + 8$ tags (Figure 2). (Geman *et al.* use a depth 5 tree, with 62 tags.)

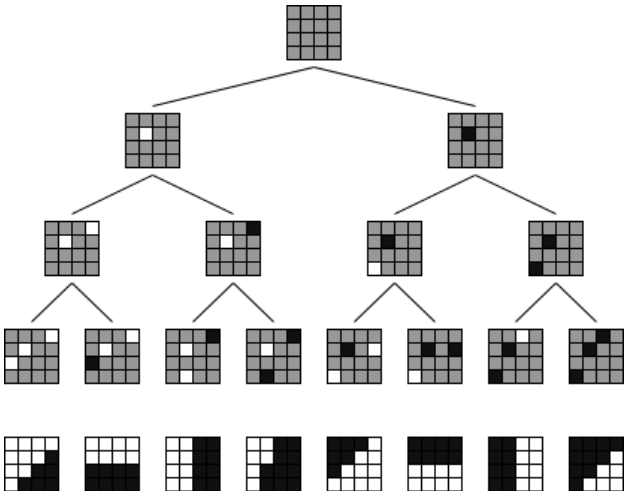

Figure 2. First three tag levels (Geman, Amit, & Wilder, 1997).

(3) Each pixel in the image is assigned all the tags encountered as the 4×4 sub-image containing that pixel at $(1, 1)$ (top-left corner).

In step 1, they restrict the population to sub-images that are not all white or all black; the result is concentrated processing around boundaries (Figure 3). They refer to generalizations from binary to grayscale patches (*ca.* 1995), but the best bet is to use modern methods as described in the BoF lecture.

The resolution 4×4 sounds arbitrary, but they found empirically that for resolutions of 10×10 to 70×70 for digits, the resulting tags work roughly equally well.

13.1.2. Tag arrangement

Now the more interesting part is the analysis of tag arrangements – these are the real "features" used for recognition. A *tag arrangement* is an attributed graph; vertices are labeled by tag type and edges by angles (Figure 4).

The angle relationships are quantized into the 8 compass directions N, NE, E, ... (or $\frac{k\pi}{4}$-sized angular intervals). Note in the "8" in Figure 4 that the same "feature" (*i.e.* tag arrangement) can appear multiple times in the same image, since the coarse quantization of tag types and angles offers some leeway. In this sense, this "binning" provides some invariance to distortion, in a manner similar to shape contexts and order structure (Carlsson, 1999).

13.1.3. Recursive partitioning of the shape space

Now that we've defined these features, how do we decide what number of tags and which arrangements to use? Clearly some will be more useful than others in a given task such as digit recognition.

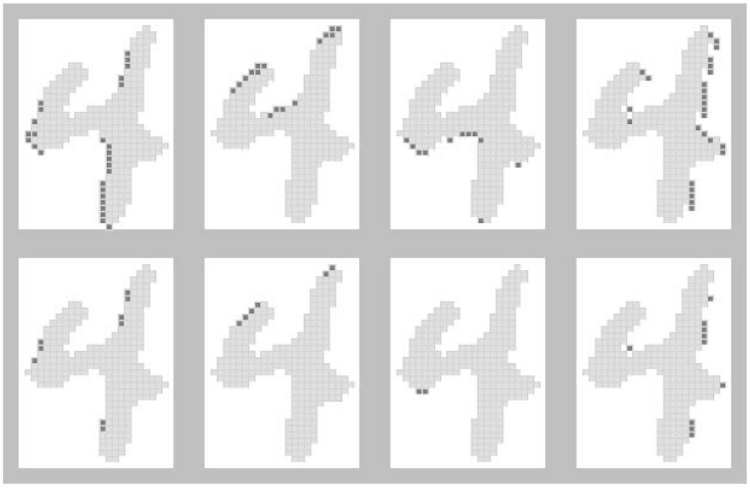

Figure 3. All instances of four depth 3 tags (top), and their refinements as all instances of 4 depth 5 tags (bottom) (Geman, Amit, & Wilder, 1997).

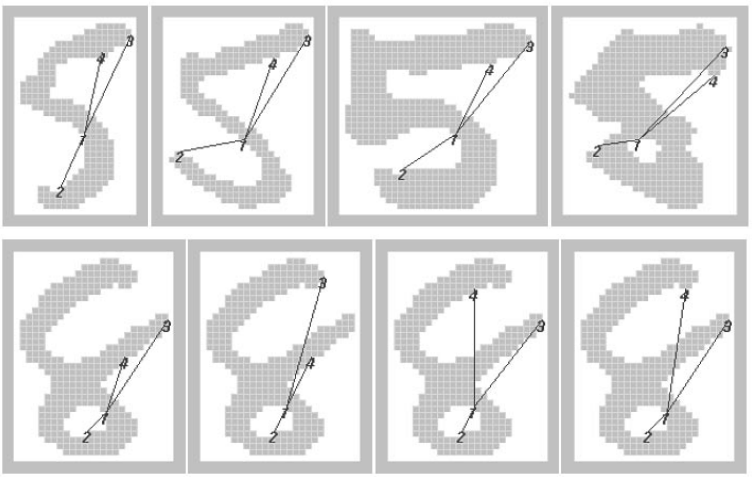

Figure 4. Examples of an arrangement in 5's (top) and the arrangement in one 8 (Geman, Amit, & Wilder, 1997).

To build the tree for tag arrangements (via recursive partitioning of the shape space):

(1) Start at the root and loop through all arrangements involving two tags and a relation. Each one splits the data into 2 groups: those that have the arrangement and those that don't.
(2) Choose the arrangement using an "entropy-based splitting rule" – one of many splitting rules, which we'll study in more detail later in the course.

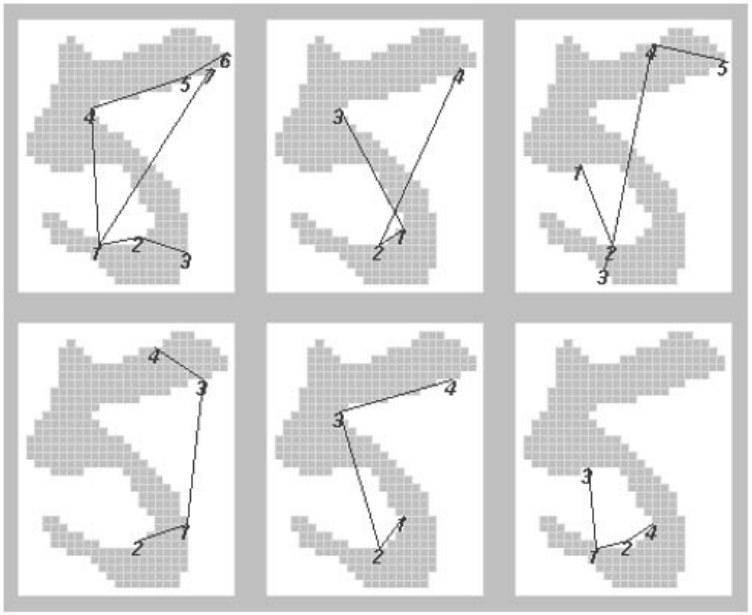

Figure 5. Example of digit 5 passed through six different trees – each tree provides its own structural description, or "point of view" on the shape (Geman, Amit, & Wilder, 1997).

The entropy of a set (of digits, in this case) can be thought of as how random the class distribution is in that set. We want the split that most reduces the uncertainty, that is, we want the greatest "information gain."

(3) Call this chosen feature A_0.
(4) Repeat this splitting procedure at the "yes" child nodes using a so-called "minimal extension" to A_0, *i.e.* A_0 plus one more tag and relation. This is a useful heuristic that greatly reduces the number of arrangements one must consider.

This is continued recursively until some stopping criterion is satisfied, *e.g.* the number of data points falls belows a threshold.

13.1.4. Multiple randomized trees

There is a problem, though, when we look at the practicality of the above method. Even though the minimal extension of pending arrangements helps matters, we still have a huge number ($62 \times 62 \times 8 = 30,752$) of arrangements at the root node, and similar problems at internal nodes.

To solve this, we look to randomization, in particular, multiple randomized trees. This is not only more efficient, but it also provides insights into different types of structural descriptions (Figure 5).

If we use multiple trees, then we need to aggregate the results somehow. There are many ways to do this; Geman *et al.* use a simple method:

- Think of each tree as a discrete random variable T on the space of digit images X. Each terminal node corresponds to a different value of T. Let $T_1, \ldots, T_N$ be the set of N trees.
- Use the training data to compute the empirical distribution (histogram) on the classes $\{0, 1, \ldots, 9\}$ based on the training data that reaches each terminal node.
- For the n^{th} tree T_n and an image x. Let

$$\mu_n(x) = (\mu_n(x, 0), \ldots, \mu_n(x, 9)) \tag{13.1}$$

be the distribution stored at the leaf of T_n reached by the image x, and let:

$$\overline{\mu}(x) = \frac{1}{N} \sum_{n=1}^{N} \mu_n(x). \tag{13.2}$$

- To classify an image in the test set, we simply compute $\overline{\mu}(x)$ and take the mode of this distribution as the estimated class.

There are many more sophisticated ways of aggregating multiple trees, but this one is about as simple as it gets, and works well even with limited amounts of training data.

13.1.5. NIST experiments

Geman *et al.* performed experiments on the NIST Special Database 3 of handwritten digits, which involved 2,000 writers who produced 100,000 training digits and 50,000 testing digits, with no overlap in writers.

Some simple preprocessing (slant and scale correction) were done; without this, the error goes up a few tenths of a percentage. The single tree error rate is $\approx 7\%$. They used 25 trees $T_1, \ldots, T_{25}$, with an average depth of about 9 and a maximum depth of 20 (*i.e.* no questions), and with about 600 terminal nodes on average. With aggregation, the error rate is about 0.8%, and with various rejection criteria in place, as low as 0.2% (*e.g.* based on ratio between the mode and next highest value). In general, this method is easy to train, using $\approx$ 25MB of memory during training and negligible memory during testing.