Stroke Extraction For Handprinted Digit Recognition

J.R. Parker
Laboratory for Computer Vision
Department of Computer Science
University of Calgary
Calgary, Alberta, CANADA

Abstract:

Many structural pattern recognition methods use straight line segments as features. Others depend on the ability to identify a stroke, or a curve drawn with a single pen down and a single pen up motion. Here a method is presented for identifying linear strokes that does not depend on a structural thinning step. It can be used for character and symbol recognition, and the features can be connected to form curves.

A feature is any measurement that is made on an image or a region. Area, perimeter, circularity, orientation, and number of holes are all features, and there are infinitely many more. Features are used to characterize objects in an image. As it is impractical to use an infinite number of features, the first task is to determine which features should be used for the problem at hand - not all features are useful. We want features that will discriminate between object classes that are likely to appear. If the problem is to count the cattle in an image containing cattle, sheep, and horses then it is of no help to use the number of legs as a feature.

The basic idea behind *structural* pattern recognition is that objects are constructed from smaller components using a set of rules. Characterizing an object in an image is a matter of locating the components, which at the lowest level are features, and constructing some representation for storing the relationships between them. This representation is then checked against known patterns to see if a match can be identified. Structural pattern recognition is, in fact, a sophisticated variation on template matching, one which must match relationships between objects as well as the objects themselves.The problems involved in structural pattern recognition are two: locating the components, and finding a good representation for storing the relationships between the components.

Lines and curves are quite commonly used as structural primitives or features. This makes sense, because line drawings can contain the same basic shape information as can a grey level raster representation. For the structural recognition of isolated symbols the basic problem is the exatrction of the linear features from the original raster image.

1 Shape Tracing

The vector template [PARK94] approach essentially attempts to compare a set of standard line drawings of characters against the incoming data. A related method, which will be referred to as *shape tracing*, attempts to draw a character over top of the incoming data to see how well the drawing matches the data. This was suggested to me by my six year old daughter, who is already an expert at digit recognition. I noticed that when I asked her how she knew a particular digit was a six she traced over it with a pencil or her finger. It is possible that the pattern implicit in a six was stored by her as a pen motion.

With this in mind, an effort was made to characterize the motions involved in drawing all ten digits. Each motion begins at a start point, which is a point at which the pen would first touch the paper in making a stroke. Motion is specified by giving a direction only, or a series of directions for continuing the stroke. Directions are crudely specified as being one of eight possible: right-up (0-45 degrees), up-right (45-90), up-left (90-135), left-up (135-180), left-down (180-225), down-left (225-270), down-right (270-315) or right-down (315-0). Motion is approximated by straight lines; of course, curves exist in handprinted characters, but most computer software would plot a curve as a sequence of straight lines. The lines used in this recognizer are longer than would be used to plot the curve, but the basic shape information remains after the linear approximation.

An example of how to draw a digit appears in Figure 1. The digit, in this case a ‘6’, begins in the upper right of the canvas., where the pen is put to paper. Drawing the digit proceeds as follows: left-down, down-left, down-right, right-down, up-right, up-left, left-up, and left-down. Not all sixes follow exactly this path, so the tracing of the path must have a certain flexibility. For ex-

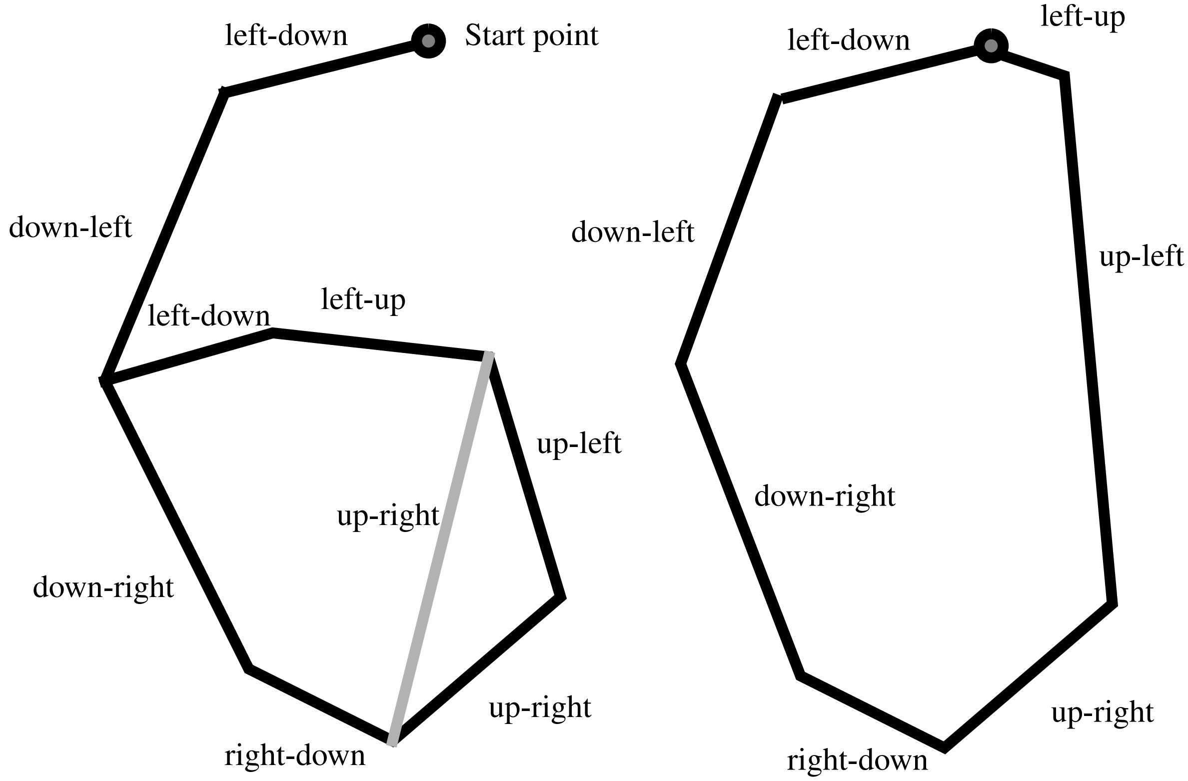


Figure 1 - Drawing a six by specifying the start point and pen motions. A zero can be described by identical pen motions, but some of the lines are different lengths.

ample, the light grey line represents an alternate path that could just as easily occur, and should also be allowed in a six. The line lengths are not all crucial, but if all of the lines were the same length, the result would have been a zero instead of a six. Thus some effort must be made to characterize the key lines and their relative lengths.

Thus, the templates in this particular matching scheme consist of sets of pen motions, and the recognizer must attempt to measure how well each set of motions matches what is actually seen in the input image. The matching step turns out to be hard to do in software, and seven versions of the program were tested before coming up with a strategy that worked.

The first step is to identify the points where the pen first touched the paper. A bad *start point* can result in a very poor match with a perfectly good input glyph. From there, straight lines are traced from each begin point, then from the ends of those lines, and so on until all possible lines (linear features) have been located. Then the recognizer attempts to trace each possible digit over the extracted linear features, starting at each of the identified start points. Each of the three steps above is sufficiently complex to warrant a more detailed explanation.

2 Locating Start Points

Given a raster image, the question that must be answered here is “At what point did the pen most likely first touch the paper (or lift from it)?” Once again, here is a problem that the human visual system has no trouble with, but that is difficult to solve in an algorithmic fashion. It may be better to ask what the properties of such a region are in terms of local pixel measurements, since pixels are the primitives most easily accessed. When looked at this way, two solutions come to mind: the first uses the endpoints of the skeleton of the image, and the second uses a small window and looks for the characteristic ‘bump’ seen at a start point.

Using the skeleton endpoints seems obvious enough, but given the kinds of artifacts that thinning can produce it is likely to produce spurious results. Consider, for example, the raster glyphs seen in Figure 2. The first set (a-c) have rather good skeletons, and the start points can be found quite simply by looking for 1-connected pixels in the skeleton. However, the second set (d-f) possess artifacts created by the thinning process that have resulted in an extra 1-connected pixel at the end of a short line segment. Some of these can be detected and removed simply by noting that an end pixel should not appear within a small distance of a point where two lines join - a join point is identified by the fact that it has 3 or more neighbors. This works for Figure 2d and 2e, but not for 2f.

In addition to artifacts, the method can be fooled by join points having extensions from both ends. Serifs come to mind as examples of this sort of problem, by in hand printed characters it is more common to have a simple overrun of a line with the point where it was supposed to join, causing a small extension. These points will legitimately be labelled as start points, although they are not by the original definition. However, if the later recognition steps are aware of the sort of ‘false starts’ that are likely to occur then, for example, the drawing stage can take many of them into account.

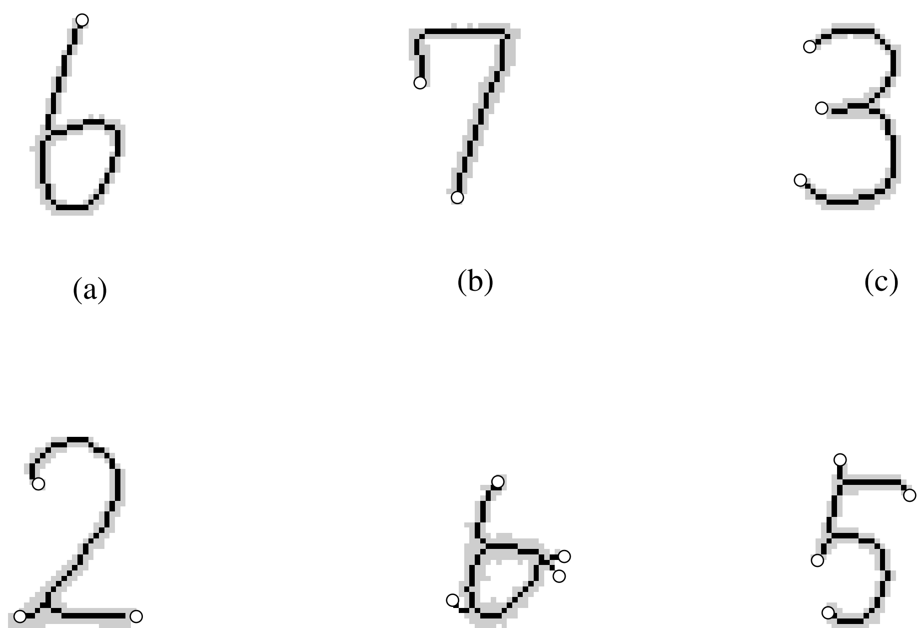
The second method for identifying start points is to use a small square window into the raster input image. The window is moved to all possible locations and the number of black to white transitions seen on the boundary of the window are counted. If a stroke endpoint is within the window then the number of such transitions should be one, since the stroke ends within the window. To make sure that the window covers enough of the stroke to be significant the center pixel in the window must also be black. This will be called the *moving boxes* algorithm for obvious reasons.

The moving boxes algorithm begins by estimating the width of the strokes; this is done by counting consecutive black pixels in the horizontal and vertical directions and taking the median of all such measurements as the width. Then, for each black pixel in the image, a box is constructed extending for *WIDTH* pixels horizontally and vertically. The boundary of this box is traversed in a counter-clockwise fashion and the number of black to white transitions (changes in pixel colour) are counted. A single transition implies a possible start point at or near the selected pixel. After all pixels are tested the largest clusters of possible start pixels are selected as actual starts; the centre or mass of each select cluster is used as the actual start point.

This method is illustrated in Figure 2, and some results from it are shown. This method is not perfect: some spurious start points are allowed, and some actual start points are missed. Figure 2a-2f represent the same glyphs used previously to test the method based on thinning. As before, the rough outline of 2e results in an incorrect start point at the bottom of the six digit, and there is one extra start point in the five digit in Figure 2f. It is interesting to note that if only the starts detected by **both** algorithms are used the number of false starts is reduced from 6 to 2, in the case of the thinning method, and from 3 to 2 in the case of the moving boxes method. It was therefore decided to use both methods to detect start points.

3 Tracing Linear Features

Tracing lines in a thinned glyph is relatively easy, but the result is not good enough for recognition due to irregularities in the skeleton introduced mainly by defects in the outline. A trac-



(a) (b) (c)

(d) (e) (f)

Figure 2 - Locating start points by finding 1-connected pixels in the skeleton. The cases in (a)-(c) work fine. (d) shows a ‘necking’ effect, creating a start in the lower left. (e) shows thinning artifacts caused by a rough outline. (f) suffers from line extension on the two left-most starts.

ing scheme that avoids thinning was devised, and is based on knowing the start points in advance. From each start point, the *longest line that consists of only black pixels* is located. Its start point is marked as now being used, and the pixels near the line are also marked. Pixels on the boundary of the glyph are not allowed to be end points, and so the tracing of the next line resumes from an interior black pixel connected to the previous line.

Continuing from a line endpoint is done as before, by drawing the longest line on black pixels starting at that endpoint. However, pixels that are marked are counted *against* the length of the line being traced. This means that a line on black pixels is to be preferred over one containing marked ones. In addition, line endpoints will be connected to each other when possible; that is, when the connection is on black pixels. This is done even if the connection is not the longest possible black line.

The detailed method for tracing linear features from a raster digit image is as follows:

1. Determine the bounding box for the image, and get a width estimate for the strokes.
2. Locate the stroke start points, as described in section 5.4.1
3. Dilate the image by 2 pixels, and erode by 1 pixel. Now mark boundary pixels so that they can be easily identified.
4. From each start point, determine the longest line that consists of black pixels only, and save it as a stroke.
5. For each line found in step 4, mark the pixels near the extracted line as used. Also mark the start points as used.
6. For each unmarked stroke endpoint, determine whether a black line can be drawn to another unused endpoint. If so, save the resulting linear feature, mark the pixels near the extracted line as used, and mark all relevant endpoints as used.
7. For each unmarked stroke endpoint, determine the longest line that consists of black pixels that starts at that endpoint. Save the resulting linear feature, mark the near the extracted line as used, and mark all relevant endpoints as used.
8. Repeat from step 6 until either no unused endpoints remain or no black pixels remain.

The difficult part of the algorithm above is actually the marking of the used pixels. The linear feature is a thin line, consisting of relatively few pixels. However, if the pixels neighboring the line are not properly marked then the longest line from an end pixel may very well be another line back to a pixel very near the original start. The answer was to create a rectangle oriented along the direction of the linear feature and having the same length. The width was varied until the rectangle covered the boundary of the digit in that local region as well as possible. Then all pixels that were within that polygon (the rectangle) were marked, as were any pixels around the start points, and any small, isolated black regions caught between the rectangle and the boundary. One example of the feature tracing process can be seen in Figure 3, and it can be seen from this that the pixel marking process seems to be successful.

The procedure followed for zeros and eights is only slightly different, due to the fact that neither digit generally has a start point. If no start point is found then one is created by slicing through the glyph vertically along its own axis and using the black pixel in the centre of the first stroke encountered. A start point found in this way is special, and a note is made of this fact for later use.

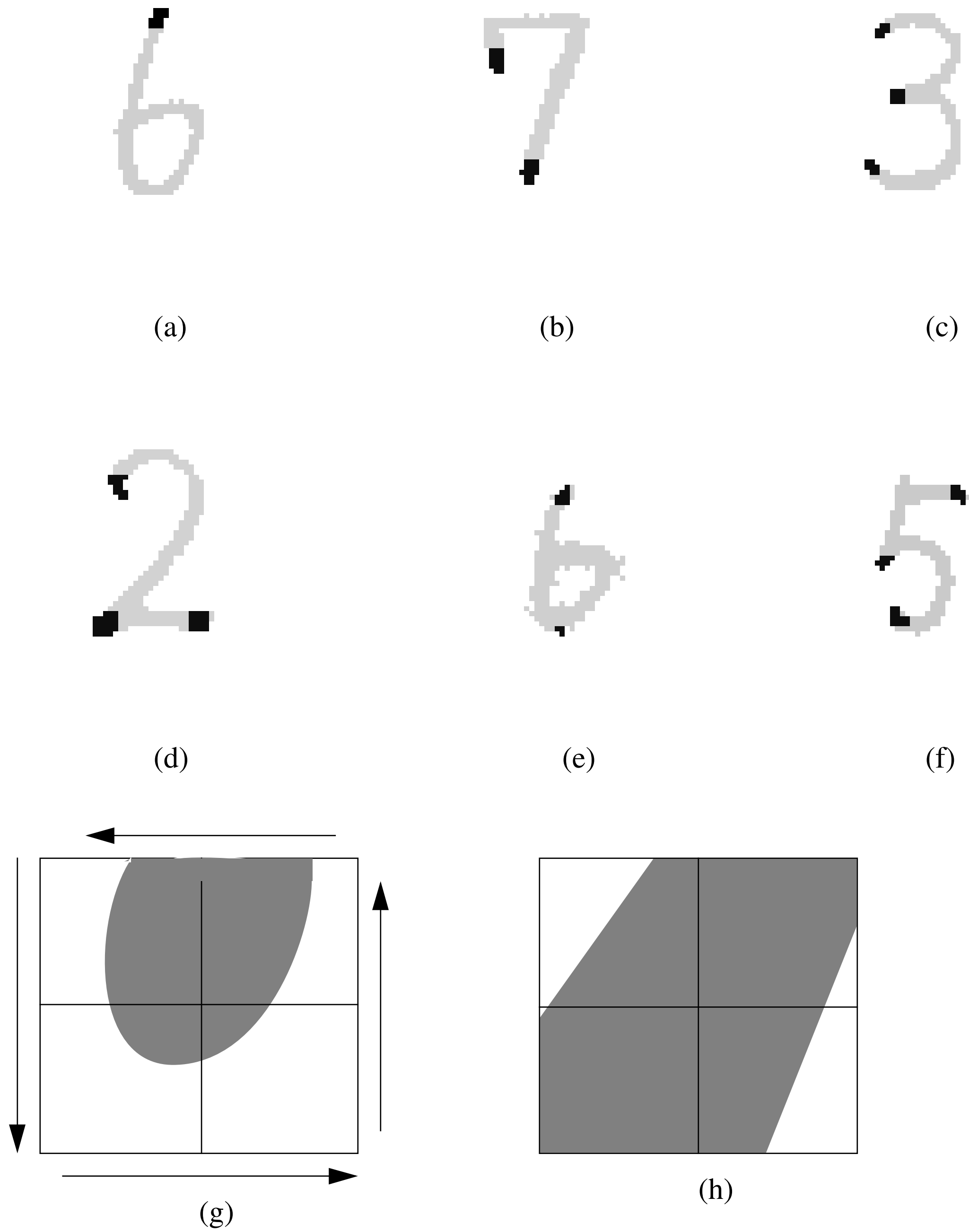
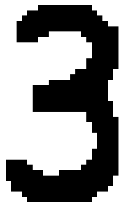
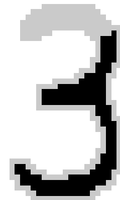


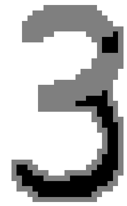
Figure 3 - The moving box algorithm for identifying stroke starting points. (a)-(f) The results of the method in identifying the start points for the same glyphs as seen in Figure 5.11. (g) The moving box placed over an obvious start point - there is one black to white transition. (h) An obvious non-starting point, where there are two such transitions.



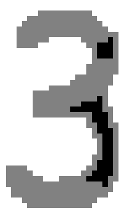
(a)



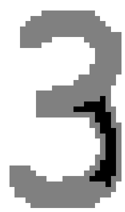
(b)



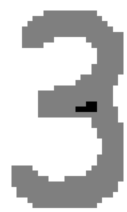
(c)



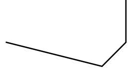
(d)



(e)



(f)



(g)

Figure 4 - (a) The original raster glyph. (b) After tracing the longest black line from the upper left start point. (c) After tracing a line from the middle start point. (d) After tracing a line from the lower start point. (e) After connecting the end points of the lines found in b and c. (f) After growing a line from the end of the middle line. (g) The extracted linear features, including a small connection between the lower and middle portions of the glyph which contained no black pixels.

The linear features found in this step are saved, as is the number of start points and their positions. This will be needed in the final stage of the recognizer.

4 Drawing The Digits

At the point where the recognizer attempts to draw digits, the data has been distilled down to a few line segments and start points. The templates are stored as code, at least in the initial version of the program; that is, the procedure that attempts to recognize a six has the template represented as a sequence of procedure calls, each designed to trace a part of the digit over the linear features. Training this recognizer is therefore a bit difficult. Strokes are traced from a specific start point, and each digit has a different collection of legal starts. This fact can be used in the classification; a character having four start points is unlikely to be a zero, for example. The number of features can also be used in this way. The only digit allowed to have a single feature is a ‘1’.

A C-code description for each of the digits exists, and the code resembles the above descriptions. A better way to describe the templates might be to display the lines allowed by each template, and label each line d0, d1, d2 ... d_n as seen in Figure 5.14. The lines associated with each label are now clearly seen, as are the reasons for the constraints placed on the various values of d_i.

Eights presented unique difficulties in tracing. There was a larger than expected variety in the shape of the eights seen in the sample data set, and that combined with the fact that an eight has a larger number of linear features than any other digit made them impossible to trace both unambiguously and reliably. The solution was, arguably, a ‘hack’, but nonetheless a reliable one. An eight will be a character that more or less agrees with a zero, but has a black region in the middle. This simple process correctly classifies 95% of the ‘8’ digits.

The classifier described above was used to classify thea set of 1000 digit images, and gave the following results:

	0	1	2	3	4	5	6	7	8	9
Correct	100%	94%	92%	99%	90%	94%	100%	88%	99%	98%
Reject	0	0	5%	1%	4%	3%	0	0	0	0

This gives an overall error rate of 3.3%. The low recognition rate for seven digits might be improved on in a number of ways, but is not a concern provided that it does not result in an overall low recognition rate for sevens in the multiple/parallel system.

5 Conclusions

Five very different methods have been presented for classifying raster digit images, each giving acceptable recognition rates alone. Combining these methods to give a single multiple algorithm for digit recognition should result in even higher recognition rates; this is the final task remaining.

6. References

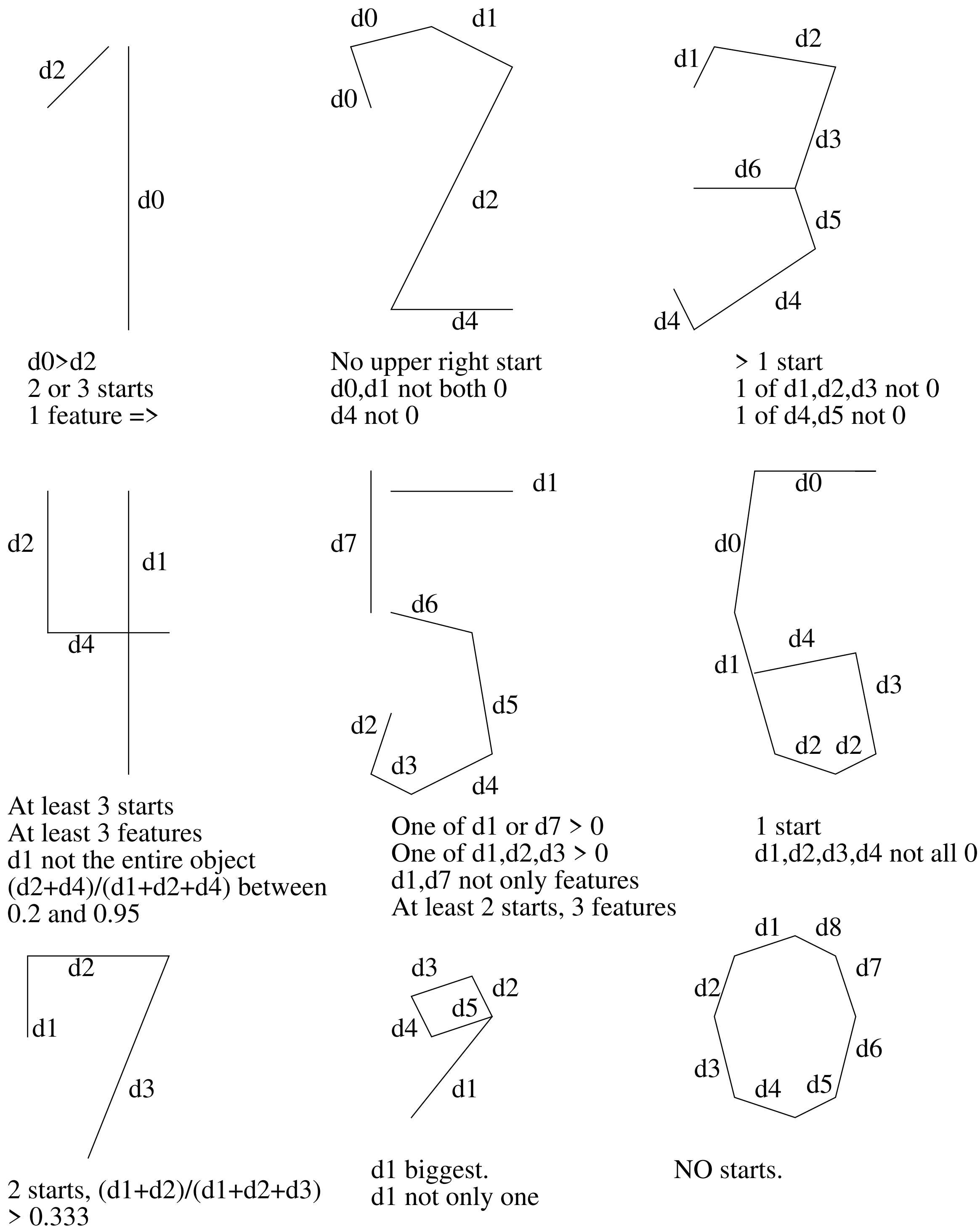


Figure 5.14 - Tracing patterns for digit recognition.