Optical Character Recognition using Neural Networks

Deepayan Sarkar
University of Wisconsin – Madison

ECE 539 Project, Fall 2003

Goal: Optical Character Recognition

The problem of OCR is fairly simple:

- Input: scanned images of printed text
- Output: Computer readable version of input contents

There are several existing solutions to perform this task for English text. The potential benefits of this approach is its flexibility, since it makes no prior assumptions on the language of the text, and it should be possible to extend it to other alphabets.

Tasks involved

From the computational point of view, there are three major tasks involved in our approach to performing OCR.

- Segmentation Given input image, identify individual glyphs
- Feature Extraction From each glyph image, extract features to be used as input of ANN. This is the most critical part of this approach, since it is not at all clear how this can be done
- Classification Train the ANN using training sample. Then, given new glyph, classify it.

Segmentation

Segmentation is important in two phases of the process.

- Obtaining training samples
  The easiest way to obtain training samples is to segment an image and ask a human supervisor to classify each glyph

- Recognizing new image after training
  As a first step for trying to recognize a new input image, it must be segmented into glyphs. An additional requirement here is to obtain the glyphs in correct order as well.

To make this easier, the input image is first divided into lines and then segmented into glyphs (details in project report).

Feature Extraction: Why do it ?

- Segmented glyphs are binary image matrices (very high dimension)

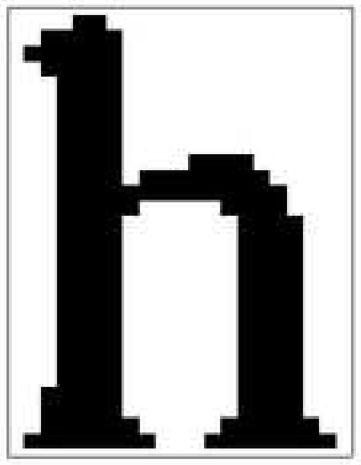
- MLP needs moderately low-dimensional input feature vector

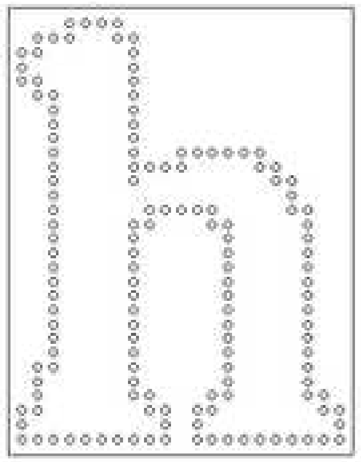
Unfortunately, there is no obvious way to reduce the dimensionality in a way guaranteed to preserve the distinctiveness of glyphs.

Feature Extraction: How to do it ?

There is no single obvious choice of features. I decided to base my features on identifiable regular parabolic curves in the image.

Step 1: Obtain boundary of image From image matrix, flag as *boundary points* backgroud pixels (0) in the image which have at least one neighbour in the foreground (1)





Feature Extraction: How to do it ?

Step 2 Loop through each of these boundary points, and figure out the 'best' parabola passing through that point fitting the boundary locally. For each point, this involves

- Decide the 'orientation' of the boundary at that point by fitting a straight line through points in a small neighbourhood of that point.

Feature Extraction: How to do it ?

Step 2 Loop through each of these boundary points, and figure out the ‘best’ parabola passing through that point fitting the boundary locally. For each point, this involves

- Rotate the image by an angle to make this line horizontal, and fit a quadratic regression line to the previously identified neighbouring points.

Feature Extraction: How to do it ?

Step 2 Loop through each of these boundary points, and figure out the 'best' parabola passing through that point fitting the boundary locally. For each point, this involves

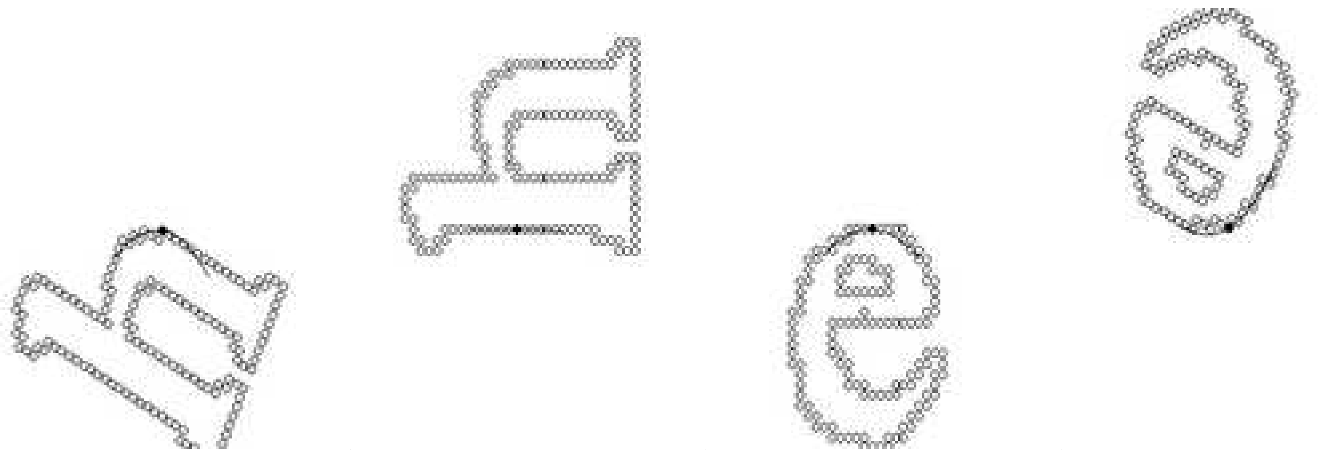
- Determine points in the boundary that are 'close' to this fitted quadratic curve (using a predetermined threshold).

- Update the quadratic curve by refitting it using all the points thus identified.

- Repeat this update using 'close' points 2 more times.

Feature Extraction: How to do it ?

Hope that the curve thus identified closely approximates the curvature of the boundary at that point. Note that it is perfectly all right if this doesn't work as expected for all points, since for points common to a single curve, it is enough that this works for at least some of these points.



Feature Extraction: How to do it ?

Step 3: Identify ‘strongest’ curve Determine the three curves that are the ‘strongest’ in terms of how many points are ‘close’ to those curves. To do this,

- order the points by the number of other boundary points ‘close’ to the best curve through that point
- record the angle of rotation and the quadratic coefficient of the curve as features (the linear coefficient is close to 0 because of the rotation)

Feature Extraction: How to do it ?

Step 3: Identify ‘strongest’ curve Determine the three curves that are the ‘strongest’ in terms of how many points are ‘close’ to those curves. To do this,

- We don’t want the same curve to be identified again, so we leave out all points identified as being ‘close’ to the first curve, and re-evaluate the ‘strengths’ of the remaining points based on the remaining points. Again choose the best curve and record the angle of rotation and the quadratic coefficient

- Repeat this once more to get a total of 6 features

Feature Extraction: How to do it ?

Step 4: Finally, add the aspect ratio (width/height) of glyph as another feature, making a total of 7

Classification

Once the features are extracted, we can go ahead and train a neural network using the training data for which we already know the true classes. After training, recognizing a new scanned image involves

- reading in the image
- segmenting the image into lines
- segmenting each line into glyphs
- classify each glyph by extracting the feature set and using the already trained neural network to predict its class

Software Implementation

Implemented as an add-on package for a MATLAB-like programming environment called R (`http://www.r-project.org`). R is

- Popular among statisticians (like me :-))
- Open source, freely downloadable
- Runs on Linux, UNIX, Mac, Windows

Sample session: Loading OCR package

Once R is started, the OCR package can be loaded by

```
> library(rocr)
```

Sample session: Creating Training data

A new collection of glyphs for training can be created, or an existing one updated, by calling

```
> updateTrainingSet(imagefile, datafile = "file.rda")
```

where imagefile is the name of the scanned image which is to be used to obtain the candidate glyphs, and datafile is the name of the file on disk used to store this set (so that the same collection can be used and updated across different sessions).

Sample session: Creating Training data

For each glyph identified in the image, the user is shown an image of that glyph and prompted to provide a class label for it.

Once the class is specified, the corresponding matrix is added to the collection along with its class label.

There is also the possibility of not specifying a class, in which case that glyph is ignored. This is useful for bad (spurious) segments as well as for very frequent letters that would otherwise dominate the list.

Sample session: Creating Training data

Problem: Typically, in a sample of actual text, some letters of the alphabet occur far more than others, and some (especially capital letters) occur very rarely. This makes collecting a balanced training set fairly laborious.

For my experiments, I manually classified a sample of 528 glyphs, which had the following distribution:

‘	,	.	a	A	b	c	d	D	e	f	fi	g
4	11	15	51	2	4	12	32	1	54	6	1	14
h	H	i	I	k	l	m	M	n	N	o	O	p
22	1	16	7	8	13	14	3	40	1	33	1	6
r	s	S	t	T	u	v	w	W	x	y		
25	37	1	36	1	13	9	16	3	1	14		

Sample session: Extracting Features

To extract the features from the glyphs in the training set, we can call

```
> featurelist = featuresFromTrainingData(datafile = "file.rda")
```

where, as before, datafile is the file on disk that stores the training glyphs. This stores the features in the variable featurelist.

Problem: This is currently very slow. However, it should be possible to speed it up by reimplementing it so that it uses C code internally

Sample session: Training MLP

A neural network can be fit to these training data by

```
> library(nnet)
> fittednet = fitNeuralNet(featurelist)
```

which stores the fitted network along with some other information in the variable fittednet.

Sample session: OCR new image

After the training is done, a new image can be processed by calling

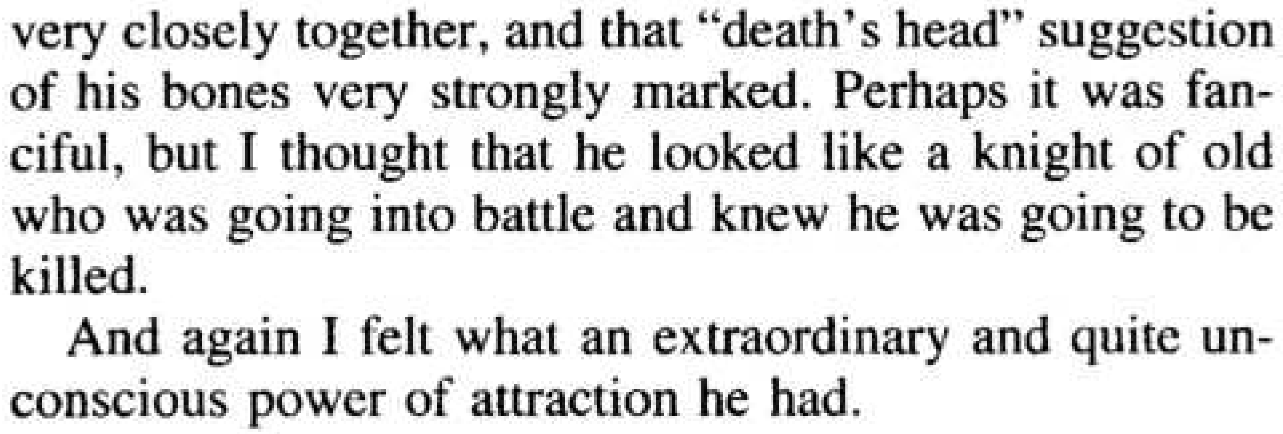
```
> ocrNewImage(imagefile, fnet = fittednet)
```

where imagefile is the name of the image to be processed, and fnet is the network to be used. Again, this process is currently quite slow because of the time required for feature extraction from each glyph.

Results

Not very impressive :-(

Results: section of OCR'd image


very closely together, and that "death's head" suggestion of his bones very strongly marked. Perhaps it was fanciful, but I thought that he looked like a knight of old who was going into battle and knew he was going to be killed.

And again I felt what an extraordinary and quite unconscious power of attraction he had.

Results: OCR results

```
[1] "veo  etoyels Ioke oer, end net  r‘deao,k   head fg suhgestion"
     very closely together, and that ‘‘deaths’s head ’’ suggestion

[2] "oI gtd genep eaw  stsougly markod. serhass lI was scnw"
     of his bones very strongly marked. Perhaps it was fan-

[3] "cifol, hmt I tho0phi ihat ha looser lthe a knight of old"
     ciful, but I thought that he looked like a knight of old

[4] "wk, was goine into  batIta anr snew he Wak geing so he"
     who was going into  battle and knew he was going to be

[5] "... apxhn  I tcit what an enH aorhi.Mwy ans suiie unm"
     ... again, I felt what an extraordinary and quite un-

[6] "eoaserouk poWer nf attracigHn he had."
     conscious power of attraction he had.
```

Conclusions

- Although results are not good, they are not that bad either, indicating that the technique is not flawed
- More training data may improve robustness and accuracy
- Speed needs to be improved (via C code) for serious testing

Several other possibilities for improvemnet are discussed in the project report.