Genetical Engineering of Handwriting Representations

Alexandre Lemieux, Christian Gagné and *Marc Parizeau*

Laboratoire de vision et systèmes numériques (LVSN),
Département de génie électrique et de génie informatique,
Université Laval, Ste-Foy (Qc), Canada, G1K 7P4.
E-mail: {alemieux, cgagne, parizeau}@gel.ulaval.ca

Abstract

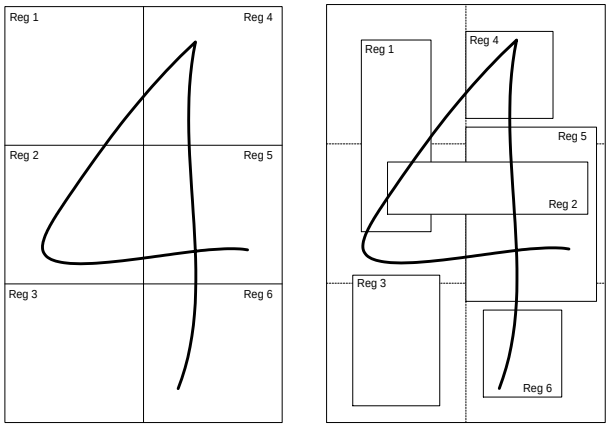
This paper presents experiments with genetically engineered feature sets for recognition of on-line handwritten characters. These representations stem from a nondescript decomposition of the character frame into a set of rectangular regions, possibly overlapping, each represented by a vector of 7 fuzzy variables. Efficient new feature sets are automatically discovered using genetic programming techniques. Recognition experiments conducted on isolated digits of the Unipen database yield improvements of more than 3% *over a previously manually designed representation where region positions and sizes were fixed.*

1 Introduction

Many handwriting representations have been developed over the past decade [10]. These sophisticated methods were however created and fine tuned with human knowledge and expertise, using an expensive, time consuming trial and error process. This paper reports on ways of automatically building efficient input feature sets for character classifiers, using Genetic Programming (GP) [1, 6].

GP has been found very useful for discovering creative solutions in a wide range of problems [7]. It is based on a Darwinian search process where populations of solutions (programs) evolve over time (generations) using, on the one hand, selection operators based on solution fitness to decide which solutions survive into the next generation, and on the second hand, genetic operators that modify current solutions. The application of GP to handwriting recognition is relatively new. To the authors knowledge, the only other previous work has been published very recently for off-line scripts [11]. This work is similar to ours, but was applied at the pixel level, whereas we make use of higher level information extracted from on-line scripts.

The paper is organized as follows. Section 2 first reviews an existing character representation that will serve as a basis for comparison. Then, Section 3 gives a description of the genetically engineered representations. Section 4 contains the experimental results followed by their analysis in Section 5. Finally, Section 6 concludes this paper.



a) b)

Figure 1. A region base representation: a) basic regular grid; b) floating regions.

2 Fuzzy-Regional Representation

One representation for handwriting recognition is based on a window of attention [4], as illustrated in Figure 1. This representation is said to be "regional", because the window of attention is subdivided into a grid of distinct regions (3×2 for Figure 1a). Within each of these regions, 7 fuzzy variables are extracted [4]: the first three measure the degree to which region content is *rectilinear*, *curved clockwise*, and *curved counterclockwise*, while the other four measure the degree to which region content is *horizontal*, *vertical*, and *oblique* with either positive or negative slopes. These 7

variables are then assembled to form a fuzzy vector, and the basic feature set is simply the concatenation of all regional fuzzy vectors plus a certain number of global features (see [4] for more details).

One limitation of this input representation is that the topology of the regions is fixed on a regular grid that stems only from the intuition of the designer. More recent work devoted to improving on this representation [9] has shown very good performance on the Unipen database using a 3×2 or a 3×3 configuration. However, it has also raised the question about how good the same features could be on some other non regular topology. Since then, many new ideas led to small improvements, but other attempts were in vain, wasting precious development resources. The object of this paper is to experiment with new automatic search techniques, namely genetic programming, that can be exploited to genetically "discover" a better handwriting representation.

3 Genetically Engineered Representations

In the 19th century, Charles Darwin proposed his revolutionary theory of natural selection which later inspired many persons in various fields of research like Holland [5] in 1975, who stated for the first time the concept of Genetic Algorithms (GA). This interesting scheme led to another more general concept sketched by Koza [6] in the late 1980s: Genetic Programming (GP). It is an evolutionary computation technique that can automatically find computer programs to solve various problems without explicit programming.

GP, just like GA, is based on a Darwinian process where populations of individuals evolve in an environment that favors the survival of the fittest. The living individuals transmit their good genes (those that make them fit) to their offspring through reproduction. Reproduction itself is based on principles of cellular reproduction through genetic operations like mutation and crossover. The fundamental difference between GA and GP resides on the genotype used by each strategy. GA individuals are coded with bit strings whereas GP uses a tree-based genotype. This impacts directly on the type of problems that can be solved. While GA are often used to optimize system parameters, GP takes advantage of an increased versatility to create and improve structures, not only values. The branches of the trees are used to code elementary operations relevant to the problem domain, while the leaves represent the inputs. Moreover, these trees can be interpreted in various ways [6] that lead to programs as algorithmic and sequential group of operations.

Figure 2 illustrates the basic algorithm behind our specific evolutionary paradigm. The first step consists in initializing a population with randomly generated individuals. These individuals are programs that are able to construct feature spaces for handwriting recognition, using a fuzzy-regional representation (§2). Genetic operators such as crossovers and mutations are then applied on this population in order to beget new individuals (see [6] for more details on genetic operators). Next, the fitness of these new individuals need to be evaluated in order to guide the selection process that engenders the next generation. This fitness evaluation translates into measuring the performance of the feature space, that is in estimating the recognition rate of some classifier based on that feature space.

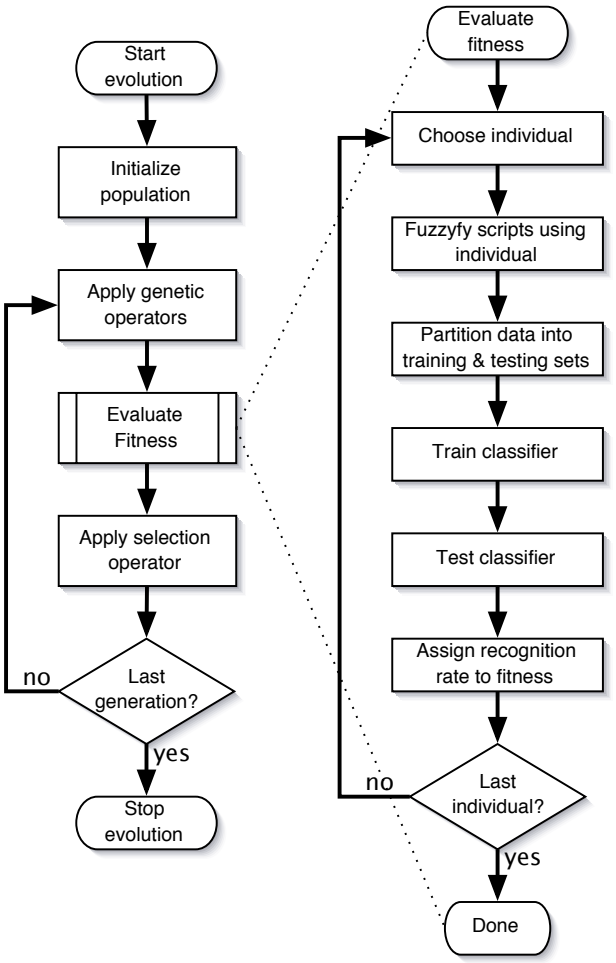


Figure 2. Basic evolutionary paradigm.

A classifier must be trained and tested on some data set, partitioned into training and testing data. This partitioning can be fixed throughout the evolution, or can be changed at each generation, or even for each individual. We change the partitioning for each individual in order to seek out more robust solutions. Part of the training set is used as a validation set in order to halt training. Correct classification rate on the testing set is used as individual fitness.

To use GP, one must specify two critical components. The first is a fitness function such as the above. The second is the set of elementary primitives that can be used to construct tree nodes. These primitives should generally be designed to transform an input of a specific data type, and produce an output of the same type. They represent prior functional knowledge of the problem domain. The following subsections describes the different families of primitives for which we give results in Section 4.

3.1 Floating regions

The starting point of the work presented in this paper is a character representation based on a regular grid of regions that spans evenly the surface covered by the character [4] (Figure 1a). It is likely, however, that this approach is not optimal. For example, some noise may result from an oversized region, or important discriminant features may not be well extracted, because of an undersized or ill-positioned region. Thus, the position and size of each region should not be fixed arbitrarily, but should be determined experimentally (Figure 1b).

Continuous values. As a first attempt to improve upon the base representation, a genetically engineered grid maker is developed. The fixed grid of the original representation is replaced by a variable grid composed of floating regions that can have different sizes and aspect ratio. The genotype of this grid maker is composed of the following primitives:

1. Continuous ephemeral constants (see [6]) randomly generated and defined over interval $[0, 1]$;
2. Basic arithmetic operators $(+, -, \times, \div)$ for computing new values;
3. A *REG* operator for extracting the fuzzy content of a region defined by four arguments: the first two specify the position of the region's lower left corner; the last two specify the position of the upper right corner.

Constants are used as tree leaves. They can serve either directly as inputs to the *REG* operator, or they can be combined with other values through arithmetic operators. The *REG* operator itself returns the lower left corner of the region, which can eventually be reused as a value to define another region. There are two different versions of the *REG* operators: the first returns the X coordinate, while the second returns the Y coordinate. A *REG* operator extracts the region content as a side effect, and concatenates the corresponding fuzzy vector to the final feature set. Inputs of this operator are interpreted as virtual coordinates into the character frame, where (0,0) is the lower left corner of the frame, and (1,1) is the upper right corner. Any value outside interval $[0, 1]$ is simply folded back into the interval.

Discrete values. A second experiment in this family slightly modifies the first grid maker by reducing the search space dimensionality. The idea is simply to modify the distribution of constants in order to better limit the search space of the evolution process. Hence, constants are again uniformly generated over interval $[0, 1]$, but with 0.05 increments. This leads to a discrete distribution of initial values. Also, operators $(\times, \div)$ are removed from the primitive set so that values remain discrete. Finally, any values exceeding the limits are folded back into the interval $[0, 1]$, as with continuous values.

Differential values. The third experiment for floating regions uses differential positioning relative to the 3×2 grid. In this way, the evolution process begins with a reduced search space solidly anchored on this regular grid. Thus, individuals define changes to the 3x2 grid. It is important to note that both position and size of the regions are modified in this model. The *REG* operators are modified so that, for instance, an argument of 0.1 is interpreted as an increment to the coordinate of the default corresponding region. Six different *REG* operators are defined, one for each region of the 3×2 grid. Constants for this experiment are uniformly generated over interval $[-0.35, 0.35]$ with discrete increments.

3.2 Fuzzy operators

A second family of experiments, similar to the previous one, tries to exploit fuzzy operators to act on the regional vectors before their concatenation to the final feature set. With these operators, the trees now have to deal with 2 different types of data: floating point values and fuzzy vectors. This particular tree organization is called strongly-typed genetic programming [8]. The following operators are added to the primitive set in order to deal with fuzzyfied regions defined by the *REG* primitives:

1. Unary logical *NOT* operator that returns the fuzzy complement of its input vector;
2. Binary logical *AND* and *OR* operators that return respectively the fuzzy conjunction and disjunction of its input vectors;
3. Finally, a *CAT* operator that simply returns its input vector, after concatenating it to the output feature set as a side effect.

The *REG* operator is also modified to extract and return the fuzzy content of a given region, but without any side effect (its side effect is transfered to the *CAT* operator). Figure 3 gives a synthetic example of a tree that uses these operators. In this example, the triangles represent subtrees containing only arithmetic operators and constants (not detailed). For a given *REG* operator, its arguments $(\mathbf{X1}, \mathbf{Y1})$ and $(\mathbf{X2}, \mathbf{Y2})$

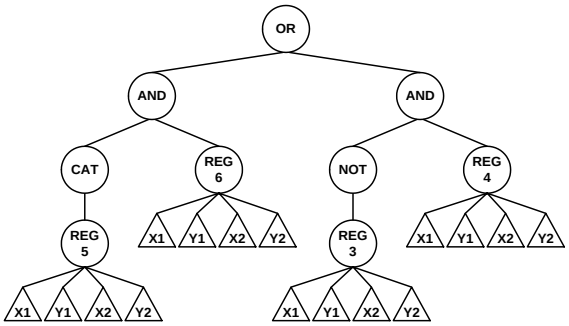


Figure 3. Example of a tree for floating regions using fuzzy operators.

Table 1. GP experimental parameters.

Parameter	Value
Number of generations	65 to 250
Population size	$5 \times 4\,000$
Migration type	one-way ring
Number of migrants	80
Tournament selection	5 participants
Minimum initial tree height	5
Maximum initial tree height	9
Maximum tree height	15
Crossover probability	90%
Shrink/swap mutation probability	50%
Standard mutation probability	5%

specify respectively the lower left and upper right corners of the region. This particular example illustrates the case of differential values, where six different *REG* operators numbered 1 to 6 exist, one for each of the six regions of the basic 3×2 grid. This tree can be interpreted as the following feature set: $\mathbf{R5} + \big[(\mathbf{R5} \wedge \mathbf{R6}) \vee (\overline{\mathbf{R3}} \wedge \mathbf{R4})\big]$, where $\mathbf{R}n$ represents the fuzzy vector of the modified nth region, $+$ is concatenation, $\wedge$ is logical *AND*, and $\vee$ is logical *OR*. Note that an implicit *CAT* is always executed at the tree root.

3.3 Fitness Evaluation

To build our fitness function, because GP is very CPU intensive, especially with large populations of individuals, we have used a relatively small development set of isolated digits extracted from the Unipen Train-R01/V07 database [3]. Overall, 375 digits per class were randomly chosen and subdivided further into training, validation and testing subsets of respectively 125, 65, and 185. The training subset is used to train a multilayer perceptron classifier using standard back-propagation, the validation subset to halt training, and the testing subset to compute fitness.

Furthermore, to avoid code bloating and limit the size of the feature space, a penalty of 1% per *CAT* operator in excess of 8 is also imposed. In practice, this makes the evolution converge towards individuals who acquire the maximum number of possible regions, benefiting from the highest feature size without penalty.

4 Experimental Results

The experiments were conducted using a development version of Open BEAGLE[1] [2], a new versatile C++ framework that implements GP and other evolutionary algorithms. Like many other techniques, GP requires the adjustment of several parameters that are summarized in Table 1.

[1] *http://www.gel.ulaval.ca/~beagle*

Evolutions were distributed on a Beowulf cluster of 19 PCs based on AMD Thunderbird Athlon processors clocked at 1.2 Ghz. A typical run that evolves a population of 20 000 individuals for about 250 generations can require over 4 weeks of CPU time. Table 2 summarizes the recognition rates obtained for both families of experiments. It also compares these rates with the performance of a basic 3×2 and 3×3 grid. These recognition rates stem from the complete section 1a of the Unipen DevTest-R02/V02 data set which contains 8598 digits (including some unreadable characters). The classifiers were trained on the specified feature set extracted from the same data set that was used for GP (i.e. $3\,750$ randomly chosen digits from Train-R01/V07).

The results of Table 2 should not be interpreted in an absolute way. The corresponding classifiers are not trained on the complete Unipen training set, and feature sets based only on fuzzy regions are not discriminant enough to achieve very high performance[2]. These results, however, do have a relative significance that is discussed in the next section.

The first column of Table 2 specifies the feature set. Basic 3×2 and 3×3 represent two regular 3×2 and 3×3 grids, while Float Reg and Fuzzy Ops denote respectively the two families of experiments: floating regions (§3.1) and fuzzy operators (§3.2). Letters C, D, and Δ specify respectively continuous, discrete, and differential values (§3.1). The second column states the total number of fuzzy vectors in the final feature set. The total feature set size is thus 7 times this number. Column 3 gives the average recognition rates obtained over 5 different training runs. The maximum over these runs is given in column 4. Finally, column 5 shows the increase in performance compared to the 3×2 grid.

Figure 4 shows a graphical view of the different regions contained in one individual that evolved from the Fuzzy Ops family. Each rectangular region in that figure corresponds

[2] Our current best feature set/classifier combination performs at the 97% level [9].

Table 2. Experimental results for Unipen DevTest-R02/V02, Section 1a (isolated digits).

Representation Type	Fuzzy Vectors	Mean Rec. Rate (%)	Max Rec. Rate (%)	Max Shift Basic 3x2
Basic 3x2	6	91.4	92.3	–
Basic 3x3	9	92.1	93.0	+0.7%
Float Reg (C)	8	95.2	95.6	+3.3%
Float Reg (D)	8	93.0	93.6	+1.3%
Float Reg (Δ)	8	92.1	92.8	+0.5%
Fuzzy Ops (C)	8	93.0	93.4	+1.1%
Fuzzy Ops (D)	8	92.2	93.2	+0.9%
Fuzzy Ops (Δ)	8	92.2	93.2	+0.9%

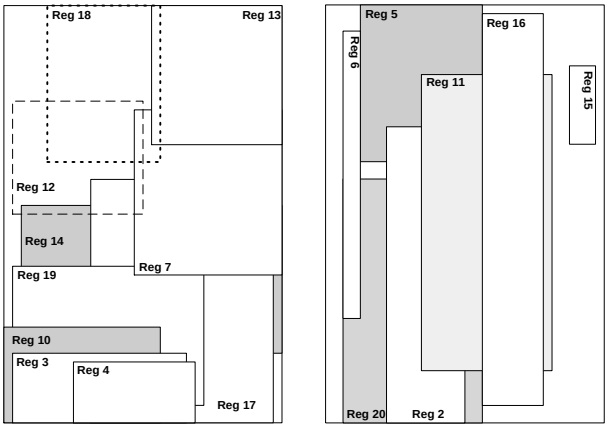


Figure 4. Genetically engineered representation: a) First and b) second sets of regions.

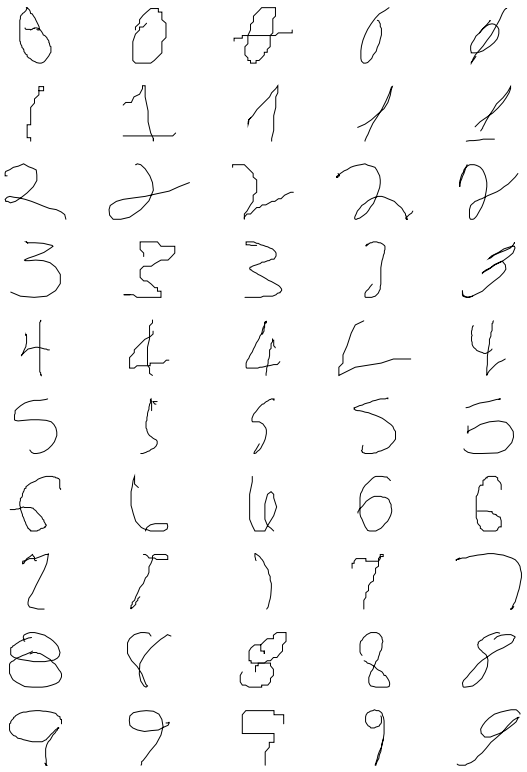


Figure 5. Samples of recognized digits.

to one *REG* primitive. That particular individual contains a total of 20 such primitives (3 could not be shown).

5 Analysis

Results obtained with the two families of experiments yield an improvement of at most $+3.3\%$ over the basic 3×2 grid. Figure 5 gives a few samples of recognized digits taken from DevTest-R02/V02. These digits are all confused when using the simple 3×2 grid. While looking at these digits, one should not forget that training was conducted on a somewhat limited data set. Curiously, if we train with the best feature set using the complete Unipen Train-R01/V07 database (15 953 digits instead of 3 750), we obtain very similar performances ($\approx 95.6\%$). If we do the same on the 3×2 grid, the recognition rate goes up from 92.3% to 95.1%. This could mean that the evolution process has converged to a representation which is not so much characterized by more discrimination power, but rather by better generalization capability.

This best result is for the first family of experiments, using continuous values. It shows the potential of GP for automatic feature set extraction. Moreover, this representation outperforms by $+2.6\%$ the basic 3x3 grid that has a higher feature count. Results also show that the addition of fuzzy operators, in the second family of experiments, does not seem to succeed in increasing performance. A possible explanation for this is that it contributes to a phenomenon known as intron proliferation [1] (similar to ”code bloat-

ing"). For example, the individual of Figure 4 contains 20 *REG* primitives, but only 8 *CAT*s. And 7 of these *CAT*s are applied directly on the output of a *REG*, thus in effect bypassing the purpose of the fuzzy operators that need to be positioned in between a *REG* and a *CAT* to be of any use. These 7 *CAT*s were applied on regions 2, 4, 7, 12, 13, 17, and 18 (see Figure 4). The 8th one, which stems from the tree root, is a complex expression that cannot be readily visualized.

Another interesting fact about the generated individuals is that many of them perform very well. Thus, GP has found several different feature sets with equivalent performance. As for the use of continuous, discrete, or differential values, results are inconclusive. It appears that GP is able to deal with any type of values, although best results were achieved with continuous values. But this could be fortuitous.

One major improvement that could be done is in the area of the *REG* primitive. Instead of extracting the seven fuzzy variables simultaneously and outputing a fuzzy vector, we are considering using specialized primitives that would extract only one variable at a time. Also, we intend to use Automatically Defined Functions (ADF) [6] for evolving sub-programs (sub-genotypes) that would define these regions. Then, the main program (genotype) could concentrate on the fuzzy operators between single variables, "calling" sub-programs to obtain regions. Another possible improvement would be to use a faster classifier like, for instance, a K-Nearest Neighbor (KNN). Indeed, although the MLP is quite efficient in testing mode, it's training phase is usually slower than the training/testing cycle of a KNN. And since the fitness function of GP individuals requires a single test run, the MLP that was used in these experiments is quite inefficient. Preliminary tests have shown that by using a simple 1-NN classifier instead of an MLP, processing time could be cut at least in half, even for large training and testing sets. But the area where we expect the greatest improvements lies in the discovery of per class specialized feature sets, as well as feature sets specialized for discriminating between ambiguous character pairs.

6 Conclusion

This paper has presented a method for automatic extraction of efficient handwriting representations through genetic programming. Although this work is exploratory in nature, results are very encouraging: a basic feature set was improved upon by more than 3.3% without any explicit programming.

Genetic programming techniques have the faculty of discovering novel solutions to problems, using a subtle mixture of user defined elementary operations (instructions), fitness driven selection, pure luck, and brute force. We feel that it holds great promises for both handwriting recognition and pattern recognition in general. One of its main characteristic is also that it is not limited to working in a finite vectorial space, contrary to neural networks, for instance.

But it also has shortcomings. One of them is the huge computer resources needed to discover interesting problem solutions. With rapid progress in microprocessor design and manufacturing efficiency, however, this problem is diminishing rapidly. Moreover, evolutions over (very) long period of time can easely be envisioned. After all, if a typical research project can span several years, why would we not be willing to wait for several months of GP?!

Acknowledgments. This work was supported by an NSERC-Canada scholarship to C. Gagné, and an NSERC-Canada grant to M. Parizeau.

References

[1] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, dpunkt.verlag, 1998.

[2] C. Gagné and M. Parizeau. Open BEAGLE: A new C++ evolutionary computation framework. *Late breaking papers, Genetic and Evolutionary Computation COnference (GECCO)*, 2002.

[3] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet. Unipen project of on-line data exchange and recognizer benchmarks. *Proc. of the 14th Int. Conf. on Pattern Recognition (ICPR)*, pages 29–33, 1994.

[4] J. Hébert, M. Parizeau, and N. Ghazzali. A new fuzzy geometric representation for on-line isolated character recognition. *Proc. of the 14th International Conference on Pattern Recognition*, pages 33–40, 1998.

[5] J. M. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[6] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, USA, 1992.

[7] J. R. Koza, D. Andre, F. H. B. III, and M. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufman, 1999.

[8] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[9] M. Parizeau, A. Lemieux, and C. Gagné. Character recognition experiments using unipen data. *Proc. of 6th Int. Conf. on Document Analisys and Recognition (ICDAR)*, pages 481–485, 2001.

[10] R. Plamondon, D. Lopresti, L. Schomaker, and R. Srihari. *Online Handwriting Recognition*, volume 15, pages 123–146. John Wiley & Sons, 1999. In John G.Webster, Wiley Encyclopedia of Electrical and Electronics Engineering.

[11] A. Teredesai, J. Park, and V. Govindaraju. Active handwritten character recognition using genetic programming. *European Conf. on Genetic Programming (EuroGP)*, April 2001.