mCALI: A Multipurpose Sketch Recognizer

João Miguel Calado Vieira
Computer Science and Engineering
IST/UL, Lisbon, Portugal
E-mail: joao.m.calado.vieira@tecnico.ulisboa.pt

Abstract—In this paper we present mCALI, a trainable sketch recognizer designed to work in several devices, supporting multi-stroke sketches, tolerant to different drawing orders, and independent of sketch position, scale and rotation. It supports big datasets with miscellaneous symbols, and has the ability of distinguishing sketches that depend on their rotation for correct classification, when wanted. Our recognizer has a 95% recognition rate with 20 training examples, and 88% with 5 training examples. mCALI is a multipurpose and consistent recognizer on its results, with a standard deviation of 8-3% between 1 and 30 training examples, on an average of results in 9 different datasets. The recognizer has two classifiers, in order to allow the user to choose the most suitable classifier, according to his preferences. The first classifier has a training time of 703 ms per sketch on a smartphone and 3 ms per sketch on a laptop. The second one has a training time of 165 ms per sketch on a smartphone and 0.35 ms per sketch on a laptop. Both classifiers have an average execution time of 0.1 ms on a laptop.

Index Terms—recognizer; sketch; shape; recognition rate; classifier; features; training time; execution time.

I. INTRODUCTION

Personal computers, like desktops, laptops, and recently smartphones and tablets, are becoming essential in a person’s life due to the high number of tasks that can be simplified with their usage. Thanks to the appearance of devices with touch screens, the user’s need of expressing himself or to perform tasks with gestures or sketches has been increasing. However, there is not a solution in the sketch recognition area that could easily be integrated into any of these devices.

To develop a new recognizer we have to take into account several aspects: Diversity - It must be capable of supporting a wide variety of symbols, to guarantee that even if the symbols used in each context are very different from each other, the recognizer still gets good recognition rates. To support several symbols, it must be also trainable; Restrictions - To ensure that the gathering of training sketches is as natural as possible, the recognizer must not impose restrictions on the sketch drawing, i.e., it should not limit things such as, the number of strokes in a sketch, the position, scale and rotation of the sketches, or even the drawing order of the sketch to be always the same; Performance - It should have good recognition rates without needing a lot of training examples; Vocabulary size - The recognizer must be able to handle big training datasets and minimize the results loss with the growing of the symbol database; Resources - To work in several devices that have different hardware limitations, the recognizer must minimize the computational resources used.

Although there are already many recognizers with good recognition rates, some of them don’t meet a number of the requirements stated above. Some only recognize single-stroke sketches, others are not trainable, some are not tolerant to different drawing orders, and others have a high execution time. In this work we propose to develop a solution that meets all or most of the requirements listed.

Our sketch recognizer is multi-stroke, independent of sketch position, scale and rotation, and tolerant to different drawing orders, thanks to the features based on geometric properties, like ratios between area, perimeter, width, height, and filling. It is able of handling several and big datasets due to the high number of individual features (24) that help to describe the sketches in several geometric aspects. For the recognizer being trainable, we used classifiers as a method of learning and grouping the sketches, to be able to support several training datasets with different symbols. To guarantee a good time performance in several devices, and also to meet the needs of several users, the recognizer has two built-in classifiers. The first has the best recognition rates (95%) but takes longer to train, and the second is a lot faster to train (less 3/4 of time) but has lower recognition rates (94%). Both classifiers have an execution time of 0.1 ms.

With the chosen features, the recognizer is able to distinguish different sketches that have the same convex hull, and has the ability of differentiating sketches that are dependent of their rotation. This last aspect was not as good as we wanted, but nonetheless it has that ability. Also, the choice of the features influenced the computational resources needed to run the recognizer. Because the features are based in numerical calculations of euclidean points, their calculation is very fast (0.01 ms) and low on computational power.

Our work has contributed with a new recognizer, but also with several studies that could be helpful to other people. We made a comparative study of several classifiers regarding their recognition rates and training time, in several datasets. Another study was made to identify the best features to use in mCALI, to analyse their positive or negative contribution on distinguishing sketches. We also studied different subsets of features to choose the subset that provided the best results, and also analysed the efficency of the features that are dependent of sketch rotation. To study the performance of our recognizer, we gathered sketches to create a new dataset, with a total of 8160 examples from 24 symbols, made by 17 users. With the new dataset and the datasets of other works, we made a comparative study of mCALI against other recognizers.
relatively to recognition rates, and training and execution times.

This document is organized as follows: In Section II we describe the most relevant works we studied, in Section III we present the mCALI recognizer, in Section IV we describe the selection process of the classifiers and features to integrate with mCALI, in Section V we show the experimental evaluation we performed and the results, and in Section VI we present the main conclusions of this work.

II. RELATED WORK

We began this work by studying other recognizers on the area of sketch recognition. The first one was the predecessor of this work, CALI [9], [10], [29] a multi-stroke recognizer of geometric shapes and commands, with a recognition rate of 95.8%. The recognizer is based on three main ideas: using global geometric properties extracted from input shapes; using a set of filters to either identify or remove unwanted shapes; and fuzzy logic to overcome imprecision and uncertainty. Thanks to the features based on geometric properties, CALI has the ability of distinguishing different types of stroke (solid, dashed, bold), and the drawing order doesn’t affect the correct classification of a sketch. Although there is a trainable version of CALI that uses the Naive Bayes classifier, in order to use that algorithm, it is mandatory to discretize the features values.

The second recognizer studied was HHReco [16], [17], a recognizer based on statistical approach to sketched symbol recognition using Zernike moments as features. The Zernike moments are polynomials that represent a class of orthogonal moments on a unit circle, that have been used in image recognition with good results. The moments are independent of rotation and reflection and can be constructed to an arbitrary order. Although higher order moments are more precise to perform image reconstruction, they are also more susceptible to noise. As Zernike moments are not invariant to scale and translation, the recognizer has to do some pre-processing first. Each sketch is scaled to a 100x100 pixel model and its centroids are translated to origin. Next, all sketch strokes are approximated and interpolated to produce a more evenly distributed data points for moment calculation. At the end, a sketch is represented by a multidimensional feature vector composed of Zernike moments. The authors then made a study to choose the classifier and the Zernike moments order that provided the best results. They concluded that the SVM classifier and Zernike moments of order between 8 and 10 obtained a recognition rate of 96.9% on dependent tests and 96.7% on independent tests.

Another recognizer studied was $N$ [2], [3], an extension of $1$ [35], that can recognize multi-stroke sketches and has the ability of distinguishing symbols that differ only by orientation ($A$ and $A'$) with a recognition rate of 96.6% on 15 training examples. The algorithm starts by generalizing from one multi-stroke to all possible uni-strokes using alternative stroke orders and directions (i.e. the letter X drawn with two strokes, generates 8 uni-strokes). Then, each uni-stroke is preprocessed by the same steps used in $1$. First, it is resampled to 64 points that are spread equidistantly, then rotates the uni-stroke to its "indicative angle", defined by the centroid (x,y) to the first point, so that it is at 0°. Third, it scales the stroke non-uniformly to match a reference square. Fourth, it translates the stroke so that its centroid is at the origin. This steps result in a template. On the recognition phase, the candidate template is compared to each of the stored templates to calculate the average distance between their points. The template that has the minimum average distance is the result of the recognition. However, the approach of creating uni-stroke permutations to represent multi-stroke templates has the problem of resulting in a combinatoric explosion of the number of templates stored. The number of uni-strokes generated for a multi-stroke sketch with $S$ strokes is: $U = S! \times 2^S$; which means that, for example, a cube made with 9 strokes results in 185,794,560 templates. Another problem that this approach has is the possible confusion between different symbols, like "=" and "z", that could be the same because of the uni-stroke conversions.

The most promising recognizer that was studied is the latest member of the $S$ family, $SP$ [34] is an improved version of $SN$ that was aimed to solve the problem of combinatoric explosion resulting from the uni-stroke permutations to represent multi-stroke sketches. This recognizer was based on the concept that a sketch is nothing more than a "cloud of points", in which direction, number of strokes and drawing order, become irrelevant to represent a sketch. This is possible because a cloud of points is represented in a vector on which the ordering of the points is not chronological. The algorithm of $SP$ contains part of the $1$ algorithm (resampling, scale and origin translation) along with a new matching technique. The matching is made using a Greedy heuristic that calculates the minimum sum of distances between all pairs of points, by encoding the confidence degree between each pair of points. With this approach, the $SP$ recognizer achieves recognition rates of 99% with 5 training examples on independent and dependent tests with an execution time of 5.5 ms.

In the end of this study, we noticed that there are two approaches the recognizers use in their features: features based on geometric properties or vector related, and features based on sketch images. Each of these approaches has its advantages and disadvantages, but generally the recognizers that use features from the first type have simpler and faster algorithms, while the ones that use features from the second type have more complex algorithms that, in some cases, increase the execution time.

III. MCALI RECOGNIZER

On this section we describe all the details related to mCALI. Its architecture, the integrated classifiers and the features used.

A. Architecture

Our recognizer works mainly in two modules, training and recognition. On the training mode, the recognizer receives a sketch (set of points), extracts the features and builds a feature vector that is then inserted on the classifier. Similarly, on the recognition mode (which means that the recognizer
was trained before) the recognizer receives as input a sketch, extracts its features, and builds the feature vector. But instead of inserting it on the classifier, the classifier analyses it and identifies the group it belongs as result. The result can be only the highest value, or a distribution of values for each class of symbol that was used as training.

B. Classifiers

The recognizer has two built-in Vote classifiers. This classifier is a learning algorithm that implements a voting system between the several classifiers that are inside it, in which each classifier has a vote about the class a new element belongs. The most voted class is then the result. The two classifiers that mCALI contains are the ones that obtained the best results in a study of several classifiers in 9 datasets, that is described on the next section.

The first one is Vote 1, that contains the KNN, Random Subspace and LogitBoost classifiers, has the best recognition rates (95%), but with slower training times (703 ms per sketch on smartphone and 3 ms per sketch on a laptop). The second classifier is Vote 2, that contains the KNN, Random Subspace and Naive Bayes classifiers, has lower recognition rates (94%) but is substantially faster to train (165 ms per sketch on a smartphone and 0.35 ms per sketch on a laptop). These values were obtained from the average results in all datasets, with 20 training examples. Both classifiers have an execution time of 0.1 ms, on a laptop, to classify a sketch. The classifier that the recognizer uses is selected by the user, according to his preferences or context in which the recognizer is going to be used.

C. Features

The first set of features we use in our recognizer are the features used in CALI, mCALI’s predecessor. The features are based on special polygons calculated for each sketch, and then their geometric properties are used to build several ratios. The special polygons are the convex hull, the largest quadrilateral and the largest triangle inscribed in the convex hull (Figure 1).

\[
\frac{P_{CH}^2}{A_{CH}} \quad \frac{A_{LT}}{A_{CH}} \quad \frac{P_{LT}}{P_{CH}} \quad \frac{A_{EQ}}{A_{LQ}} \quad \frac{TotalLength_{shape}}{P_{CH}}
\]

We added another special polygon to help to distinguish symbols that depend on their rotation, the extreme quadrilateral. It’s a polygon that has the maximum and minimum x and y coordinates as its vertices (Figure 2).

With the area and perimeter of this polygon, we build a ratio that varies with the sketch’s rotation.

\[
\frac{P_{EQ}^2}{A_{EQ}}
\]

Other features that are also dependent of the sketch rotation are the horizontal and vertical movements of the points in a sketch, identified by Malaviya and Peters [23]:

\[
HM = \frac{x_{\text{max}} - x_{\text{min}}}{N} \quad VM = \frac{y_{\text{max}} - y_{\text{min}}}{N}
\]

and the Pearson correlation, that gives the linear dependence between the x and y coordinates:

\[
\rho_{X,Y} = \frac{COV(X,Y)}{\sigma_X \sigma_Y}
\]

where \(COV\) is the covariance and \(\sigma_X \sigma_Y\) are the standard deviations of the x and y.

To be able to distinguish different sketches that have the same convex hull, we added a set of features we called "intersections", that gives us spatial information of the sketch points. Basically, we scale the convex hull to create two smaller versions of it, with their centroids in the same coordinates. Then, we search for the points of the sketch that intersect those polygons and we calculate new convex hulls for the intersecting points (Figure 3).

With those new convex hulls, we build ratios regarding their areas and the original convex hull. These features are invariant to position, scale and rotation.

\[
\frac{A_{\text{IntersectCH1}}}{A_{CH}} \quad \frac{A_{\text{IntersectCH2}}}{A_{CH}} \quad \frac{A_{\text{IntersectCH3}}}{A_{CH}}
\]
The last set of features were obtained from another special polygon, the bounding box of the sketch. With the bounding box, we divide it in four quadrants and then calculate the length of the sketch in each quadrant (Figure 4). Basically, it’s the filling ratio for each of the quadrants. With those ratios, we have a perception of the percentage of the sketch in each quadrant, which is good to characterize the sketch, but also to know its rotation.

![Fig. 4: Feature based on quadrants.](image)

We also use the bounding box to create a ratio between its area and the convex hull area.

\[
\frac{A_{CH}}{A_{BB}}
\]

IV. Choosing Classifiers and Features

To be able to choose the classifiers and the features to use in mCALI, we had to make several tests to guarantee that the recognizer has the best possible combination of recognition rates and training times. The tests were made on 8 datasets, from the CALI, $S1$, $SN$, $SP$ and HHReco recognizers, which are represented in Figure 5.

A. Classifier Analysis

We analysed several classifiers from different groups: the simple classifiers (Random Forest [5], Naive Bayes [18], KNN [1] and SMO [30]), classifiers that are based on other classifier (AdaBoost [11], Bagging [4], Attribute Select, Random Subspace [15] and LogitBoost [12]), and classifiers that use multiple classifiers (Vote [20]). In the simple classifiers, we observed that KNN and SMO were the best ones with fewer training examples ($\leq 7$), and Random Forest was the best with more training examples ($> 7$). On the classifiers of the second group, we discovered that Random Subspace and LogitBoost had better results than all the others (including the simple classifiers). Although, with few training examples, KNN and SMO were still the best ones. As the final step we analysed two Vote classifiers, Vote 1 (with KNN, Random Subspace and LogitBoost) and Vote 2 (with KNN, Random Subspace and Naive Bayes). We observed that Vote 1 had the best results, followed by LogitBoost and Vote 2. The results are presented in Figure 6.

Then, we analysed the training times of all classifiers on a laptop (Toshiba A300; CPU-Intel Core2 Duo T9400 2.53 GHz; RAM-4GB) and on a smartphone (Samsung Galaxy Mini GT-S5570; CPU-600 MHz; RAM-384 MB). We discovered that although Vote 1 has the best recognition rates, it is also the slower classifier of them all. From observing the charts in Figure 7, LogitBoost seems to be the cause of this longer training time. Based on this, we decided to use two classifiers on mCALI, Vote 1 that has the best recognition rates (95%) but is slower to train (3 minutes on the smartphone and 0.5 seconds on the laptop, with 20 training examples), and Vote 2 that has slightly lower recognition rates (94%) but is faster to
train (40 seconds on the smartphone and 0.1 seconds on the laptop, with 20 training examples).

![Training times of several classifiers on a laptop (above) and a smartphone (below).](image)

Fig. 7: Training times of several classifiers on a laptop (above) and a smartphone (below).

**B. Feature Analysis**

One of the aspects that we wanted to analyse from the set of features used, was to discover if the features that depend on the sketch’s rotation should be used separately because of those properties. This could be reducing the recognition rates, and we wanted to know if they were. To test this, we collected the recognition rates for different subsets of features, the original CALI features and all the features without each or all of the rotation dependent features. The tests were made with a model validation technique, the k-fold cross validation, in this case 10-fold cross validation. This technique takes the original dataset and randomly partitions it into 10 equal size subsets, which 9 are used for training and 1 is used for testing. The cross-validation process is then repeated 10 times, with each of the subsets used exactly once as the testing data. The $1 dataset was the one selected to make these tests, because it has 4 pairs of symbols that are the same but are dependent of their rotations. The results are presented in Figure 8.

From this analysis, we discovered that having all the features result in higher recognition rates than with other subsets of features. With all the features we have an increase of 28% on the recognition rates relatively to the original features of CALI, and an increase of 23% with the features dependent of the sketch’s rotation. Although these features change with the sketch rotation, they are also good to describe the sketch and differentiate it from others.

**V. EXPERIMENTAL EVALUATION**

To thoroughly analyse the performance of mCALI, already with the chosen classifiers and features, we made a set of tests that are described in this section. The results for mCALI are relative to the Vote 1 classifier.

**A. The Procedures**

The first thing we did before the testing, was to collect a new dataset of 24 symbols (geometric shapes, generic symbols and letters). The dataset has a total of 8160 sketches made by 17 users, that had to draw each symbol 20 times. The symbols that are in this dataset were chosen to ensure that we had some symbols from the other recognizers datasets (CALI, $1, $N, $P and HHReco), as well as to have new symbols. With this selection we guarantee that our dataset is diversified and also familiar to the other datasets. We also had 3 pairs of symbols that are rotation dependent, particularly "<" and ">", "H" and "I", "A" and "v" (Figure 9).

![mCALI's dataset.](image)

Fig. 9: mCALI’s dataset.

Afterwards, we made dependent and independent tests relatively to the new dataset, to analyse the performance of our recognizer in the two scenarios in which a recognizer can be used. A scenario in which a user can train the recognizer with his or her own data, and another in which the user can...
use the pre-trained recognizer which is adaptive. Next, we observe the recognition rates of several recognizers against ours (CALI, $P$ and HHReco), in 9 different datasets, including the new one. The tests are relative to the number of training examples, which were selected randomly, and the remainder examples were used as testing cases. We made 10 iterations to create each of the sets of training examples, in order to get the average results. Additionally, in order to analyse the performance with the growing of the symbols dataset, we make tests regarding the vocabulary size in the new dataset. Then, we present two confusion matrices of mCALI and $P$ using the new dataset for 10 training examples. Finally, we analyse the training and execution times of the several recognizers.

B. Results

a) Dependent and Independent Tests: Regarding the dependent and independent tests, we observed that our recognizer has better results on a dependent scenario, in which it achieves a 93% recognition rate with 10 training examples. A lot higher than in the independent scenario, in which it achieved a 79% recognition rate (Figure 10). In the independent tests, the users that train and test the recognizer are different, and so, we believe that the low results are due to differences in the users drawing style. As we have several symbols in our dataset that could be drawn in several ways (arrow, smileys, stars), the results between users can be worse.

b) Recognition Rates: Next, we studied the recognition rates of CALI, $P$, HHReco and mCALI in all 9 datasets we gathered. As can be seen in Figure 11, on average and with 20 training examples, mCALI has a 95% recognition rate, $P$ has 83%, CALI has 79% and HHReco has 78%. mCALI is also the most consistent of them, with a standard deviation of 8-12% in its results, while the standard deviation of $P$ is 24-12%. Generally, mCALI performed 10-15% better than the second best recognizer, $P$.

We tried to include $SN$ in our analysis, but we gave up on trying to collect the results because the tests took 3 days to complete, only for one iteration on the new dataset. This is due to the combinatoric explosion resulting from the calculation of all the uni-stroke permutations of a multi-stroke sketch. Obviously, this affects the performance of the recognizer on
either the execution time and the memory consumption, and makes it nearly impossible to use. On the iteration we were able to register, it had very similar results compared to mCALI, that were better than $P$.

It’s important to note that the values we gathered for the CALI, $P$ and HHReco recognizers are different from the ones described in their works, because our tests were performed in 9 different datasets and the results presented are the average, relatively to the number of training examples per class. For example, $P$ had very good results in the $ family datasets (between 96-98% with 5 training examples) but had worst recognition rates for the HHReco and CALI datasets. HHReco reported to have an accuracy rate of $≈ 94\%$ with 10 training examples per symbol, while we obtained (using their recognizer and dataset) recognition rates between 49-89% between 1 and 30 training examples per symbol. This is because their value was obtained with dependent tests, that use the set of each user individually for training and testing. Our value was obtained by using the whole dataset and selecting randomly the sketches that were used for training.

c) Vocabulary Size: We also made another study to analyse the recognition rates regarding the vocabulary size in the new dataset, i.e. the number of symbols that the recognizer has to distinguish. The tests consider between 2 and 24 different symbols and 10 training examples are selected randomly.
It was expected that the recognition rates would decrease as the number of different symbols increase, so our goal with this test was to identify the recognizer with the least possible variation. By analysing the results presented in Figure 12, we observed that mCALI and SP were the ones that had the smaller value of 16%. However, mCALI was the only one that had a range where the recognition rates didn’t decrease, between 10 and 15 training examples.

d) Confusion Matrices: With the previous tests, we discovered that mCALI and SP were the two best recognizers of the four we analysed. So we decided to build confusion matrices for these two recognizers on the new dataset, in order to thoroughly analyse which symbols were being confused with others (Figure 13).

![Confusion Matrices](image)

Fig. 13: Confusion matrices (values in percentages).

On mCALI, there are 19.3% misclassified symbols, in which 6% are on the symbols that are dependent of their original rotation. The features we used to distinguish these types of symbols are good, but still aren’t as good as we wanted. On SP, there are 21% misclassified shapes, specially on confusable symbols like rectangles, parallelograms and diamonds, in which mCALI performs better. However, on the sketches that are dependent of their rotation, the results are almost perfect. If mCALI had the results on those symbols as we expected, we could have had an increase of 6% on the recognition rates with fewer training examples.

e) Training and Execution Times: Finally, the last tests that we performed were aimed at observing the training and execution times of the several recognizers. These tests were made with the new dataset on the same laptop used in Section IV. The results are presented in seconds on Figure 14.

![Training Times](image)

Fig. 14: Training times (above) and execution times (below).

Relatively to the training time, Vote 1 is the slowest one, taking 3.33 ms per sketch, followed closely by HHReco with 3.15 ms. Vote 2 decreased greatly the training time by taking 0.66 ms per sketch, while SP took 0.2 ms per sketch and CALI has a training time of practically 0.

On execution times, CALI, HHReco and mCALI were the fastest ones with a time of less than 0.05 ms, while SP takes 32 ms per sketch. Even if this is still a low value, there is a big difference from SP to the other recognizers. This is because SP compares the candidate sketch to all the templates it stored in the training phase, in execution time. This could affect the performance of the recognizer on devices with lower computational power. Even though Vote 1 takes the longest time to train, the training process is only done one time, as the execution time represents every classification made.

VI. CONCLUSIONS AND FUTURE WORK

At the end of this work, we have a new recognizer that meets several goals that we defined in the beginning. Our recognizer is multi-stroke, invariant to the sketch’s position, scale and rotation, and tolerant to different drawing orders, thanks to our features based on geometric properties, with
ratios of area, perimeter, width, height and filling. In order to be trainable, our recognizer uses classifiers as learning and classification methods, to be able to support several datasets. mCALI supports a wide variety of datasets, even if they are big, due to the high number of individual features (24) that help to describe sketches in several geometric aspects. In order to work in several devices, and also to meet the needs of all users, the recognizer has two classifiers. The first one obtains the best recognition rates (95%) but is slower to train, and the second is much faster (less 3/4 time than the first) but has slightly lower recognition rates (94%).

To create this new recognizer we made several studies relatively to the performance of several classifiers in their recognition rates and training times, and of analysis of the features contribution that can help others in similar areas. We hope these studies will help in building an informative perspective about the classifiers performance, and what should the criteria be to select the most adequate features.

Additionally we gathered a new dataset with 8160 examples from 24 symbols, made by 17 users. With this new dataset and other datasets from related works, we made a comparison of several recognizers relatively to their recognition rates, training and execution times. So, in this work we summarize an analysis of several classifiers, several sets of features, several recognizers and several datasets.

Other of the features we wanted our recognizer to have was the ability of distinguishing the same sketch with different rotation, in order for the recognizer to be able of being integrated in several datasets with symbols from different contexts. We made that possible with the set of features we used, but we wanted the results to be better in that area. If the performance in that area was how we wanted it to be, we would have add an increase of 6% in our recognition rates with fewer training examples.

With this aspect, it would be interesting to pursue as future work, to improve the classification of sketches that are dependent of their original rotation, by adding or changing some features that would better describe a sketch’s orientation. Currently, some of our features are based on special polygons like convex hulls. However, a convex hull fails to represent an outline of a sketch, because it encapsulates empty space on its area. Adding features based on "Alpha Shapes" [8] and "Gamma Shapes" [6] could be a promising approach to successfully outline a sketch. If this approach is possible to integrate on 2D sketch recognition, there would be no more problems with the same convext hull for different shapes, and would also help in identifying its rotation. As there is a great diversity of training datasets that can be used on a sketch recognizer, its normal for some classifiers to work better than others on certain datasets. As the choice of a classifier to use is not obvious, it would be interesting to find a solution that selects the best classifier to use according to its training set. In a similar way, the choice of features to use on a recognizer could also be made automatically. If there was a solution with these abilities, there would be a recognizer that is totally adaptive to its training set, whichever it is.


