Generative Learning of Visual Concepts using Multiobjective Genetic Programming

Krzysztof Krawiec

Institute of Computing Science, Poznan University of Technology
Piotrowo 2, 60965 Poznan, Poland

Abstract

This paper introduces a novel method of visual learning based on Genetic Programming, which evolves a population of individuals (image analysis programs) that process attributed visual primitives derived from raw raster images. The goal is to evolve an image analysis program that correctly recognizes the training concept (shape). The approach uses generative evaluation scheme: individuals are rewarded for *reproducing* the shape of the object being recognized using graphical primitives and elementary background knowledge encoded in predefined operators. Evolutionary run is driven by a multiobjective fitness function to prevent premature convergence and enable effective exploration of the space of solutions. We present the method in detail and verify it experimentally on the task of learning two visual concepts from examples.

Key words: Visual learning, genetic programming, generative pattern recognition, evolutionary synthesis.

1 Introduction

Visual learning seems to be the most promising way of building scalable and adaptive image analysis systems. Unfortunately, learning in computer vision is usually limited to parameter optimization that concerns only a particular processing step, such as preprocessing, segmentation, feature extraction, etc. Reports on methods that synthesize complete object recognition systems starting from raw image data are rare. Most algorithms are also application-specific, which makes the acquired knowledge difficult to transfer to other applications.

The most popular way of equipping a vision system with learning capability consists in introducing an off-shelf machine learning (ML) algorithm into

the chain of image processing, analysis, and interpretation. Though usually straightforward, this approach implies serious simplifications in terms of representation of input data (commonly a fixed-length vector of image features) and the expected output (discrete, nominal decisions). Also, given the large number of features that can be derived from the input image, and conseqently high dimensionality of the input space (when compared to non-vision ML applications), the risk of overfitting becomes grave, unless human intervention constrains the search by, e.g., preselecting only a handfull of the most promising features.

In this paper, we hypothesize that visual learning may benefit from a novel way of assessing learner's ability to recognize (interpret) an input image. The proposed assessment method is more thorough than in conventional ML as, in a sense, it forces the learner to prove its 'understanding' of the input image. Technically, learners are encoded as a genetic programming (GP) individuals [1], i.e., as expression trees built of elementary operators that dwell in a population maintained by an evolutionary algorithm [2,3]. Each learner processes, analyzes, and interprets information given in a form of *visual primitives* (VPs) that represent local salient features derived from the input raster image. When exposed to an input image, the learner produces in response a simplified sketch of that image. An evolutionary fitness function examines the sketch, using multiple objectives to assess its different aspects, and appropriately rewards the individual. In such a way, the evolutionary process promotes individuals that provide best interpretations of the input image, in the sense detailed further in the paper.

Therefore, the primary contribution of this paper is an approach to image interpretation and object recognition that (i) guides visual learning by estimating learner's ability to reproduce the input image, (ii) engages multiple objectives for learner's evaluation (Section 4.3), (iii) uses visual primitives as basic 'granules' of information (see Section 4.1), and (iv) relies on evolutionary computation (GP in particular) to effectively search the hypothesis space (Section **??**).

The following Sections 2 and 3 detail our motivations and summarize the related work. In Section 4, we thoroughly describe our approach. Section 5 demonstrates the performance of the approach on a visual task of acquiring two visual concepts. In Section 6, we provide summary and draw conclusions for further research.

2 Motivations

Any machine learning algorithm requires guidance when searching the space of hypotheses (identified with *learners* and *individuals* in this paper) [4,5]. In supervised learning, this guidance is usually driven by the quality of discrimination of decision classes, technically expressed as classification accuracy, sensitivity, selectivity, or a similar measure. This approach is characteristic for, among others, the 'wrapper' approach to feature selection and construction in machine learning [6,7]. We claim that such way of evaluating learner's performance is rather superficial, as it focuses exclusively on the *final* output from the learner and does not examine the processing that led to it. If the number of possible hypotheses is high and the number of training examples low, which often the case in visual learning, many of hypotheses may perform well by pure chance, despite, for instance, relying on object features that are essentially irrelevant for the task being solved. Though such overfitting can be fought by constraining the hypothesis space, such an intervention is usually arbitrary and requires domain knowledge.

Overfitting becomes even more likely when, in addition to feature selection, learning includes also feature *synthesis*, meant as explicit construction of a more sophisticated mapping from the space of raster images into the space of image features. For instance, in our past experience with evolutionary design of pattern recognition systems [8,9], a recognition system might happen to evolve an irrelevant feature that was coincidentally correlated with decision class label. Overfitting is also likely to lead to false positive errors when one classifies objects from beyond the problem domain, i.e., examples that do not belong to any of decision classes that the system was trained on. For this reason, Revow *et al.* prefer generative methods to statistical ones when dealing with handwritten character recognition, stating that "statistical recognizers can occasionally confidently classify images that do not look like a character" [10, p. 593].

In other words, learner's evaluation in terms of class discrimination only does not necessarily reflect the 'appropriateness' of its entire decision making process, in particular of the image features being used. To circumvent this problem, in our approach learner is not expected to explicitly discriminate the positive examples from the negative ones. Rather than that, it should *learn the generative description of the positive class*. To attain that, it has to detect the important image features, and, based on them, produce a sketch that reproduces the input shape. The reproduction process is sequential and consists of a series of *drawing actions* that reproduce the input shape part by part. In such a way, learner undergoes a more thorough evaluation when compared to conventional supervised learning, where only decisions are considered.

We claim also that such procedural shape reproduction reflects to some extent the process of human image understanding, especially if the subject of reasoning is *shape*. For instance, a person asked to prove his/her understanding of the concept of a triangle is expected to formulate it as a specific arrangement of simpler visual objects – sections. When confronted with an example of a triangle, such a person should be not only able to recognize it, but also to produce a top-down decomposition, to identify the particular components (three sections), and to show how this triangle can be reproduced using the section concept. Similarly, our method is procedural (stepwise), and also generative: the learning agent is expected to *generate* drawing in response to the input stimulus.

In this paper we use genetic programming [1,11,12], a variant of evolutionary computation, a general bio-inspired template of performing global parallel search in high-dimensional spaces [2,3]. Its commonly quoted virtues include relatively little task-specific tailoring and low risk of being trapped in local minima of objective function. It has sound rationale in both computational biology and in optimization theory, and has proven effective in a wide spectrum of benchmarks and real-world applications.

Evolutionary computation is used here mostly because our space of genetic programs (i.e., the space of all possible learners-hypotheses) cannot be effectively searched by means of exact methods due to high dimensionality and enormous cardinality. Also, like in many other applications of genetic programming, our objective function lacks properties that could be used to speed up the search by, for instance, prunning the unpromising parts of the search space (as the Branch and Bound algorithm does). Heuristic or metaheuristic search is, therefore, the only plausible method that can yield reasonably good suboptimal solutions in a polynomial time. This is also consistent with the practical attitude chosen here, where we do not strain to necessarily find a globally optimal recognizer, being satisfied with a well-performing suboptimal one.

3 Related Research in Visual Learning

In most approaches to visual learning reported in literature, learning is limited to parameter optimization and usually concerns only a particular step, such as image preprocessing, segmentation, or feature extraction. Only a few methods close the feedback loop of the learning process at the outermost (e.g., recognition) level [13,14,15,16,17,18,19,20,8,21,22]. Learning methods that use raw image data and produce complete recognition systems are rare, often make use of domain-specific knowledge and/or predefined object models, and are usually specialized towards a particular application.

In [9,8] we proposed a method that evolved feature extraction procedures encoded as Genetic Programming [1] and Linear Genetic Programming [11] individuals. The approach implemented feature-based recognition, with each individual in population encoding a particular sequence of predefined image processing and feature extraction operations. When undergoing evaluation, an individual (feature extraction procedure) produced feature values for all images from the training set. Next, the discriminative ability of the resulting features was measured by performing an internal cross-validation test on the training data using a machine learning classifier.

The idea of symbolic processing of attributed visual primitives (VPs) using genetic programming was first explored in [23], where we applied it to recognize computer screens in indoor scenes. Despite interesting results, it was obvious that solving so specific task cannot lead to elaboration of more general visual concepts and generic understanding of images. This shortcoming motivated work presented in this paper, which shares the idea of VPs with [23], but aims at multi-objective generative learning.

The approach presented here may be considered as a special case of generative pattern recognition. In a representative study on that topic, Revow *et al.* used a predefined set of deformable models encoded as B-splines and an elastic matching algorithm based on expectation maximization to recognize handwritten characters [10]. In [24], an analogous approach proved useful for recognizing hand-drawn shapes. However, our method goes further, as it does not use explicit models of objects, but encodes them implicitly in the genetic programming code. Secondly, our 'implicit models' are not handcrafted but automatically derived from the training data in the process of evolutionary learning. And, last but not least, our learners have to reproduce the input image using *multiple* drawing actions, which forces them to discover how to break up the analyzed shapes into elementary components.

4 Visual Learning Driven by Image Reproduction

The proposed approach may be shortly characterized as *generative visual learning*, as our evolving learners try to *reproduce* the input image and are rewarded according to the quality of that reproduction. In that process, learners focus on a particular aspect of visual information, which is shape in this study. Other factors, like color, texture, shading, etc., are ignored.

Image reproduction takes place on a virtual canvas spanned over of the input image. On that canvas, a learner is allowed to perform elementary *drawing actions*, DAs for short. To enable reconstruction, DAs have to be compatible with the image aspect that is to be reconstructed. As in this paper we con-

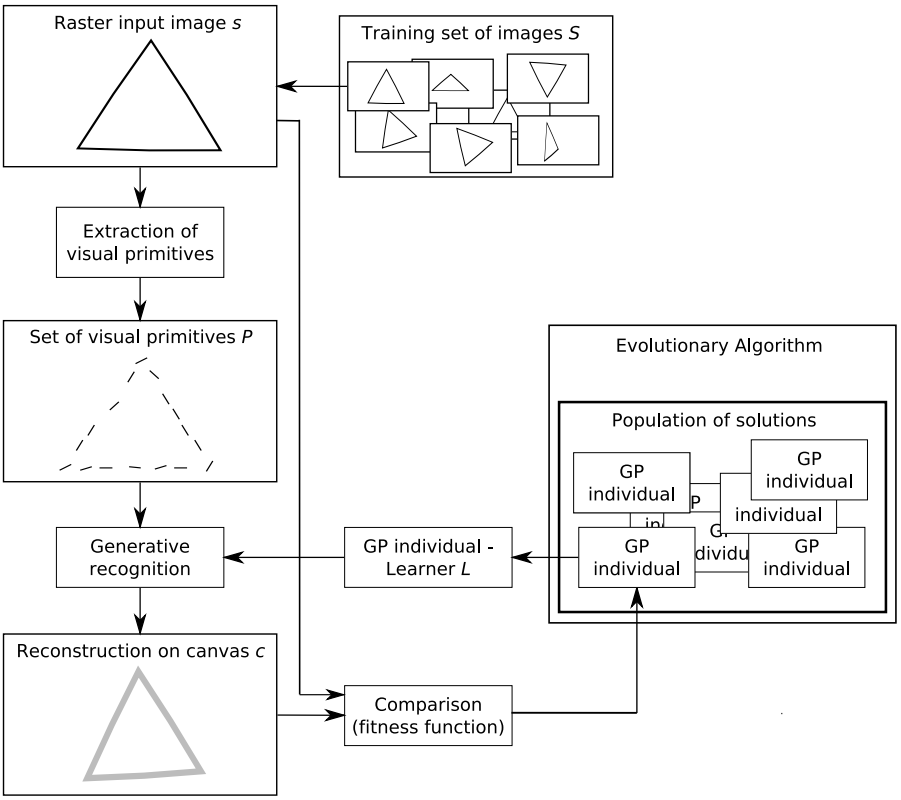

Figure 1. The outline of GP-based generative visual learning.

sider two-dimensional closed polygons, we implement DAs as *sections* of fixed width. For instance, reconstruction of a trianglular shape within this framework requires the following steps: (i) detecting conspicuous features – triangle corners, (ii) pairing the corners, and (iii) performing three DAs that connect the paired corners. It is worth emphasizing, that learners are not given a priori information about the concept of the corner nor about the expected number of them, but are expected to discover these on their own.

On the top level, the proposed method uses evolutionary algorithm [2,3] to maintain a population of individuals (solutions), each of them being a visual learner implemented as genetic programming tree (Fig. 1). The processing carried out by a learner L for an input image s (stimulus) may be split into three stages. First, we detect from s visual primitives and gather them in a set P. Next, L analyzes P and produces a drawing on a canvas c, attempting to reproduce the shape of object shown in s. Finally, canvas c is compared to the original input image s using two criteria, and the result of that comparison determines the fitness of the individual-learner L. The following three subsections describe these major steps of the approach.

4.1 Visual Primitives

To reduce the amount of data processed by learners and to bias the learning process towards the most relevant information, we consider only salient, localized image features. For each detected feature, we build an independent *visual primitive* (VP for short) and gather all of them in a set P. VPs are elementary 'granules' of our representation and play analogous role to terminal symbols in syntactic pattern recognition or terminal elements in cognitive image understanding.

In the left-hand part of Figure 1, we show an exemplary set of VPs obtained in this way from an image of a triangle. The learning algorithm described in the following sections does not make any assumption about the way VPs are created. Possible instances of VPs include edge fragments, regions, and texems. However, the type of detected feature determines the image aspect that is subject to analysis. As in this paper we focus on shape, we use VPs that reflect edges detected in the input image s by a straightforward gradient analysis procedure. First, we extract from s candidate VPs based on the gradient magnitude computed by means of Sobel edge detection filter. The candidates are sorted with respect to decreasing gradient magnitude, and are subsequently added to the resulting set P, one by one, with respect to this order. However, we impose a lower limit on VPs' mutual proximity: a candidate p may be added to P only if P does not contain any VPs that are too close, i.e., there is no $p' \in P$ such that $||p - p'|| < d_{min}$. Each VP in the resulting set P is described by three scalars called hereafter *attributes*: spatial coordinates of its location and gradient orientation.

As this procedure is not subject to learning, it always produces the same result for a given training image. Thus, we run it only once prior to evolutionary run, and cache the VPs to reduce the time complexity.

4.2 Visual Learners

To represent learners, we choose expressions used commonly in genetic programming (GP, [1,11,12]). Genetic programming is a variant of evolutionary computation with solutions (individuals) encoding definitions of functions, called alternatively procedures. In the evaluation phase of an evolutionary run, each individual processes a training example and returns a result which is compared with a pre-defined desired value. The outcome of that comparison, usually averaged over multiple training examples, determines individual's fitness. GP proved extremely successful in many real-world applications, providing human-competitive solutions for different problems, including re-discovery

Table 1
The GP operators

Type	Operators
$\Re$	Ephemeral random constant, +($\Re$,$\Re$), -($\Re$,$\Re$), *($\Re$,$\Re$), /($\Re$,$\Re$), sin($\Re$), cos($\Re$), abs($\Re$), sqr($\Re$), sgn($\Re$)
Ω	ImageNode() (returns P), SetIntersection(Ω,Ω), SetUnion(Ω,Ω), SetMinus(Ω,Ω), SetMinusSym(Ω,Ω), ForEach(Ω,Ω), AddAttribute(Ω,$\Re$), AddAttributeToEach(Ω,$\Re$), GroupProximity(Ω), Ungroup(Ω), ForEachCreatePair(Ω,Ω,Ω), SelectorMax(Ω,A), SelectorMin(Ω,A), SelectorCompareConstant(Ω,A,R,$\Re$), Draw(Ω)
A	p_x, p_y, p_o
R	Equals, EqualsPercent, Equals10Percent, Equals20Percent, LessThan, GreaterThan
G	Sum, Mean, Product, Median, Min, Max, Range

of patented and discovery of patentable designs [12].

Our approach relies on the most popular tree-based variant of GP [25]. Each visual learner L is a procedure written in a form of tree, with nodes representing predefined elementary *GP operators* that process VPs. The *terminal* nodes fetch the set of primitives P derived from the input image (see section 4.1), and the *non-terminal* (inner) tree nodes process these data all the way up to the root node. A non-terminal node may (i) group primitives, (ii) perform selection of primitives using constraints imposed on attributes, or (iii) define new attributes and attach them to primitives.

Table 1 presents the complete list of GP operators that our individuals are composed of. We use *strongly-typed* GP (cf. [25]): a node N_1 may accept node N_2 as a child only if the type of N_2's output matches the type of corresponding N_1's input. The list of types includes: numerical scalars ($\Re$ for short), sets of VPs (Ω), attribute labels (A), binary arithmetic relations (R), and aggregators (G). Importantly, sets of VPs (type Ω) may be nested, i.e., a set of VPs may include another set of VPs as an element.

The non-terminal GP operators may be divided into the following categories:

1) *Scalar operators.* Scalar operators accept arguments of type $\Re$ and return result of type $\Re$. They implement basic arithmetic and the most common unary functions, as in the famous symbolic regression benchmarks [1].

2) *Selectors.* A selector processes the set of VPs received from its leftmost child. From that set it filters out, according to some objective or condition, some of the VPs, and returns the filtered set. Thus, selectors accept at least

one argument of type Ω and return a result of type Ω. *Non-parametric selectors* expect two child nodes of type Ω and produce an output of type Ω. Operators that implement basic set algebra, like set union, intersection, or difference, belong to this category. *Parametric selectors* expect three child nodes of types Ω, A, and $\Re$, respectively, and produce output of type Ω. For instance, the operator *LessThan* applied to child nodes (R, p_o, 0.3) filters out from the set R all VPs for which the value of attribute p_o (orientation) is less than 0.3. Note that, as VP nesting is allowed, R may contain an element R' that is in fact not a VP but another set of VPs. When a GP operator queries R' for an attribute value, R' computes it by averaging the attribute values of its members.

3) *Iterators*. The role of an iterator is to process VPs one by one. Currently, we have two iterators: *ForEach* and *ForEachCreatePair*. The former of them iterates over all VPs from its left child and processes each of them using the GP code specified by its right child. The VPs resulting from all iterations are grouped into one set of primitives and returned. The *ForEachCreatePair* operator proceeds analogously, however, it has *two* children subtrees for processing: each iterated VP is processed independently by both subtrees and the VPs obtained from those subtrees are paired.

4) *Attribute constructors*. An attribute constructor assigns a new attribute to the VPs received from its leftmost child node, while the rightmost child subtree determines how the new attribute should be computed. Given a set of primitives R, an attribute constructor passes it through its rightmost subtree. This may be done once, where R is treated as a whole (operator *AddAttribute*), or iteratively, when the subtree is executed independently for all elements of R (operator *AddAtributeToEach*). For each iterated element $p \in R$, the subtree computes a scalar value v based on the existing attributes, and v is attached at the end of p's attribute list. Thus, attribute constructors accept one argument of type Ω and one of type $\Re$, and return result of type Ω.

Given elementary operators from the above four categories, a learner L applied to an input image s builds gradually a hierarchy of sets of primitives derived from s. Each application of selector (category 2 in the above list) or iterator (category 3) creates a new set of VPs that includes other elements of the hierarchy. Selector's output may be based on values of pre-defined VP attributes (coordinates and gradient orientation), or new attributes created by an attribute constructor (category 4). Additionally, some operators may take as arguments scalar values (category 1). In the end, the root node of our GP tree returns a hierarchy of nested sets of VPs built atop of P. The nodes of the hierarchy encapsulate VPs from P and/or other sets of VPs. The particular structure and contents of that hierarchy reflects the processing performed by L for image s.

Figure 2 illustrates selected steps of learner's processing, i.e., of building a

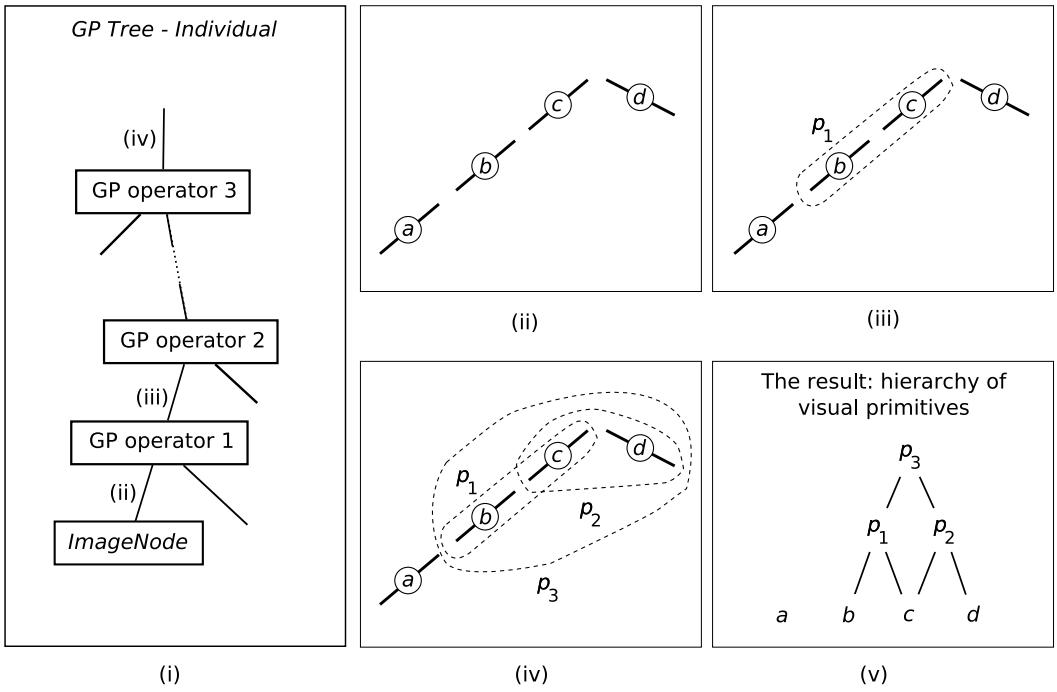

Figure 2. A fragment of learner's code (i) and its processing. (ii) A set of four visual primitives. (iii) A GP operator groups primitives b and c into group (set) p_1. (iv) Consequent GP operators build another two sets of VPs, p_2 and p_3. (v) The primitive hierarchy for (iv).

VP hierarchy in response to an input image. Figure 2(i) shows the fragment of learner's code – a path from one of its leaves to the root node. The remaining subfigures illustrate the outcomes of particular tree nodes on that path, in bottom-up order. Figure 2(ii) illustrates P, the input image representation returned by the *ImageNode* operator; it contains four VPs, each of them depicted as a short segment marked by a letter. This set of primitives is then fetched by GP operator 1, which groups primitives b and c into one subset named p_1, illustrated in Figure 2(iii) as a dashed-line oval. In Fig. 2(iv) we demonstrate the final result of processing, which includes three groups of primitives. In particular, each of the shapes denoted by p_1, p_2, and p_3, corresponds to a single VP group built on this processing path. In Fig. 2(v), this resulting hierarchy is shown in an abstract way, without referring to the actual placement of particular VPs in the input image. Note that the hierarchy does not have to contain all VPs from P, and that a particular VP from P may occur in more than one branch of the hierarchy. Importantly, the primitive hierarchy (v) should not be confused with the GP tree (i).

4.3 Drawing Actions and their Evaluation

As outlined in Section 4, in our approach learner is expected to reconstruct the essential features of the input image s. To meet that goal, it performs drawing actions (DAs) that boil down to insertions of elementary graphical elements (sections) into an output canvas c. To implement DAs within the GP framework, an extra GP operator called *Draw* is introduced (Tab. 1). *Draw* expects one argument – a set of VPs obtained from its only child node – and returns it unaffected. Its only function is to draw on c sections connecting each pair of VPs contained in the processed set of VPs. Learner's code may include multiple *Draw* nodes, located in different parts of the GP tree, not necessarily at the root. This allows selective drawing and gradual, part-by-part reproduction of complex objects.

After learner L finishes its processing, c stores the cumulative result of all DAs it has performed in response to s. A particular DA may contribute (exclusively or partially) to two types of errors:

(1) False positive error: Drawing a section in c at location where there is no edge in the input image s.
(2) False negative error: Not drawing a section in c at location where there is an edge in the input image s.

For images of closed polygons placed on a uniform background, which we consider in the experimental part, the number of pixels lit in the input image s is usually very small compared to the total number of image pixels. Thus, the a priori probability of false positive error is much higher than that of the true positive error. Moreover, they vary across images, so it would be difficult to aggregate them arbitrarily using, e.g., weighted sum. Thus, to avoid such aggregation we rely on multi-objective evaluation, penalizing learners separately for committing both types of errors by means of two maximized objectives: $f_p()$ and $f_n()$. The objective $f_p()$ measures the true positive ratio that depends on the cumulative brightness of pixels that are lit in the input image s *and* have been subject to any DA. The objective $f_n()$ computes the true negative ratio in an analogous way. Formally, for a set of training images S:

$$f_p(L) = \frac{1}{|S|} \sum_{s \in S} \frac{1}{|c|} \sum_{s[x,y]>0 \wedge c[x,y]>0} \max(s[x,y] - c[x,y], 0) \tag{1}$$

and

$$f_n(L) = 1 - \frac{1}{|S|} \sum_{s \in S} \frac{1}{|s| - |c|} \sum_{s[x,y]>0 \wedge c[x,y]>0} \max(c[x,y] - s[x,y], 0) \tag{2}$$

where $s[x, y]$ denotes the brightness of a pixel in the training image s, and $c[x, y]$ denotes the brightness of the corresponding canvas pixel. Terms preceding the summations provide normalization: $|S|$ is the number of training images, and $|c|$ and $|s|$ denote the total cumulative brightness (sum) of all pixels lit in c and s, respectively.

The following properties of $f_p()$ and $f_n()$ need emphasizing:

1) Computing $f_p()$ and $f_n()$ requires considering only pixels which have been affected by DAs (condition $c[x, y] > 0$ in formulas (1) and (2)). This immensely reduces computation time, as such pixels usually constitute a small fraction of the image.

2) DAs traversing *the same* pixel $[x, y]$ cumulatively increase its brightness $c[x, y]$. Thus, overlapping DAs decrease the $c[x, y] - s[x, y]$ term in formula (2). As $f_p()$ has an analogous property, the learner is penalized for superfluous DAs on both objectives. In this way, we promote simplicity by favoring learners which not only reproduce the shape, but also minimalize the number of DAs.

3) The objectives rely on pixel *brightness*, and not on their *numbers*. This makes them more continuous and improves the convergence of the learning process, especially when the input image is not binary but grayscale.

4) An ideal learner provides $[f_p, f_n] = [1, 1]$ and perfectly reproduces the shape using a minimal amount of drawing. On the other extreme there is an anti-ideal with $[f_p, f_n] = [0, 0]$, which produces an inverse (a negative) of the original input image s. Evaluation $[1, 0]$ is assigned to any individual that fills the entire canvas with DAs. Analogously, any learner that does not perform any DA receives fitness $[0, 1]$. In practice, these extreme cases are unlikely to occur.

5 Experimental Evaluation

5.1 Experiment Objectives and Training Data

In this part we use the proposed approach to recognize triangles and sections. Though straightforward for humans, these tasks are nontrivial, as learner's only input is a set P of a few dozens of VPs, each of them described by coordinates p_x, p_y and gradient orientation p_o. Learners have no a priori information on, e.g., spatial proximity of VPs, their collinear alignment, etc. The VPs located next to triangle vertices are not marked as special in any way; their importance has to be discovered in the learning process.

For the triangle task we prepared a training set containing images of 10 tri-

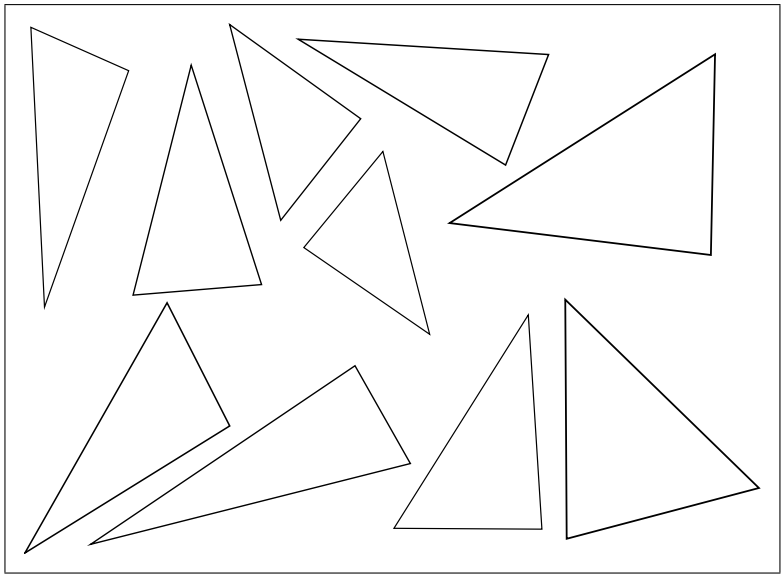

Figure 3. The training set of triangles.

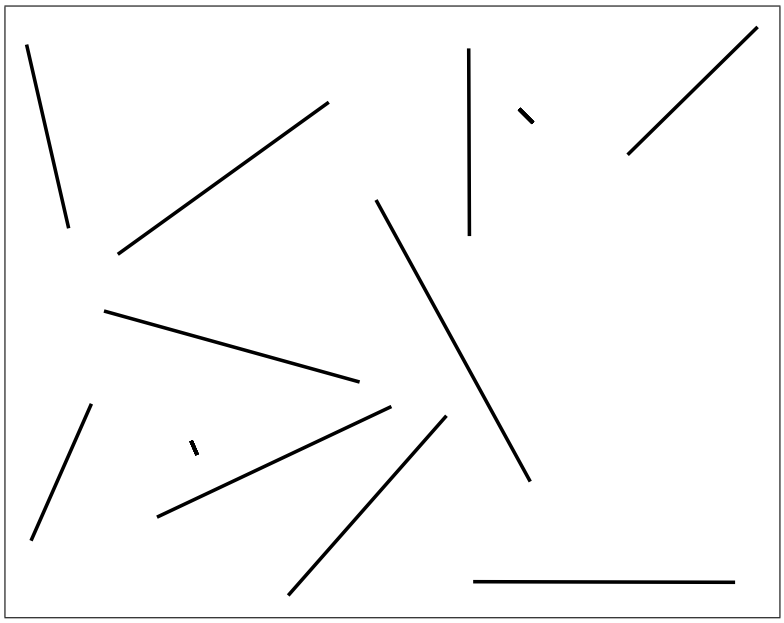

Figure 4. The training set of sections. Each long line represents one training example; the two short sections constitute noise that is included in *each* example.

angles of different sizes, shapes, and orientations, placed in a raster image of 640×480 pixels (Figure 3). Similarly, we prepared training data for the section task, composed of 10 examples shown together in Fig. 4. To make this task more demanding, we introduced some extra noise, represented by two very short sections. The noise is presented to the learner together with *each* training image. The task of the learner is to ignore that noise and reproduce the shape of the section.

Prior to the evolutionary runs, we extract VPs from all training examples using

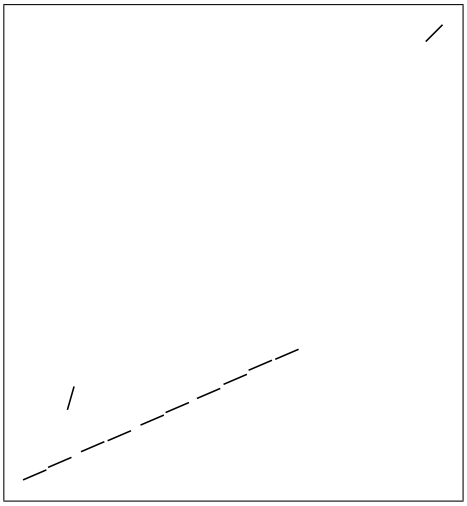

Figure 5. The VP representation P of one of the section examples (including noise).

the procedure described in section 4.1. Though, in terms of data volume, the resulting image representation P is usually several orders of magnitude smaller than the original image s, the essential sketch of the input image is usually well preserved. Figure 5 shows a visualization of P derived from one of the objects from Fig. 4. Each segment corresponds to a single VP, with orientation depicted by slant. Notice the two VPs resulting from the presence of noise in the original image.

Visual primitives have discrete coordinates that are not always perfectly consistent with the actual location of triangle pixels due to unavoidable rounding errors. Preliminary evolutionary runs have shown that this issue may severely impact the convergence of the algorithm: individuals that reproduce well the overall shape may receive low fitness due to minor shit of DAs with respect to the input image. Thus, when computing $f_p()$ and $f_n()$, we 'fuzzify' the input images by passing them through a lowpass filter, while for DAs we set section width to 3. This alleviates the problem of coordinate discretization and makes the objectives more continuous.

5.2 Parameter Settings

Our evolutionary algorithm maintains a population of 4000 individuals, initialized using Koza's standard ramped half-and-half operator with ramp from 2 to 6 [1]. Evolution runs for 100 generations and uses Pareto-ranking for selection. For this purpose, individuals are ranked from the best to the worst rank using dominance relation based on objectives $f_p()$ and $f_n()$, where an individual L_1 dominates individual L_2 if L_1 is at least as good as L_2 on all objectives and strictly better on at least one objective. Then, we randomly select an individual from r^{th} rank (r=1,2,...) with probability $\gamma^{r-1}/2$ (γ=0.6). Thus,

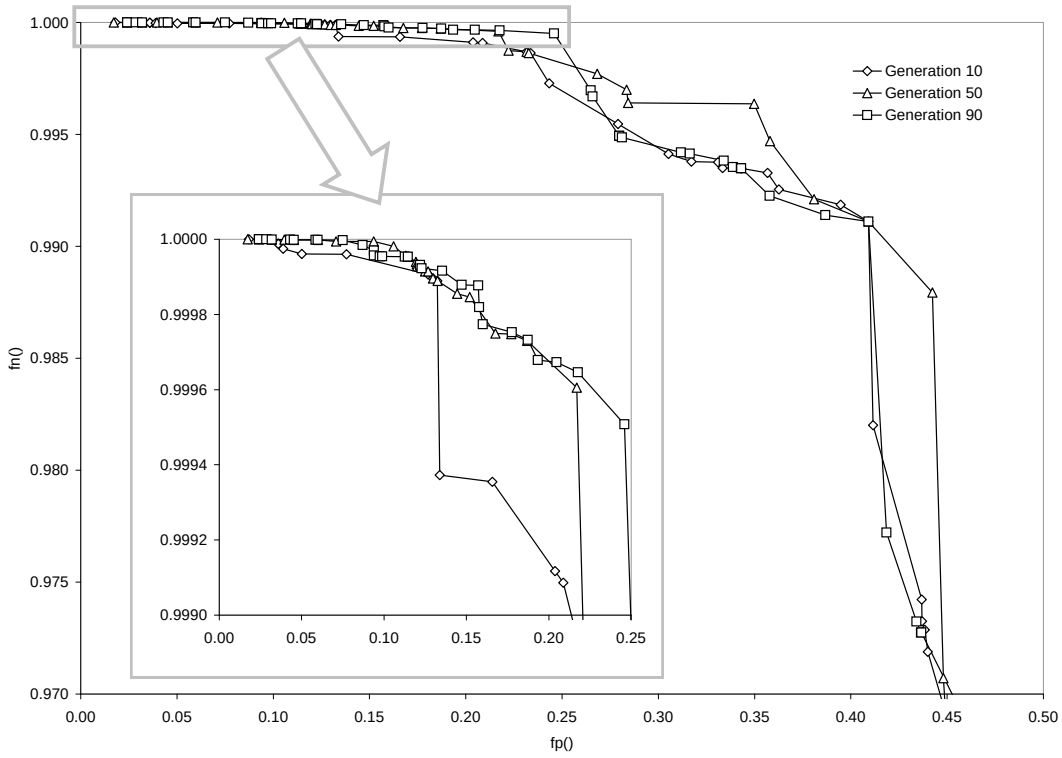

Figure 6. The Pareto non-dominated fronts of solutions for selected generations.

probability of selection drops exponentially with rank number. Each selected individual follows randomly one of three paths: with probability 0.5 it crosses over with another individual and produces offspring, or with probability 0.4 it undergoes mutation, or it is directly copied into the next generation with probability 0.1. For this purpose, we use standard GP crossover and mutation, which consist in, respectively, swapping randomly selected subtrees of GP code and replacing a randomly selected tree node with a randomly generated subtree. The tree depth limit is set to 8, so mutation or crossover that produces a deeper tree must be repeated. If feasible individual(s) cannot be produced within 5 such trials, the original individual(s) are passed as the result.

The method has been implemented with help of the ECJ package [17] and Java Advanced Imaging library [26]. For evolutionary parameters not mentioned here explicitly, ECJ's defaults have been used.

5.3 The Results

To illustrate progress of evolutionary learning, in Fig. 6 we show the front of Pareto non-dominated solutions for 10^{th}, 50^{th}, and 90^{th} generation of the triangle problem. The inset presents the selected fragment of the front in magnification. Though by definition both objectives range from 0.0 to 1.0 in-

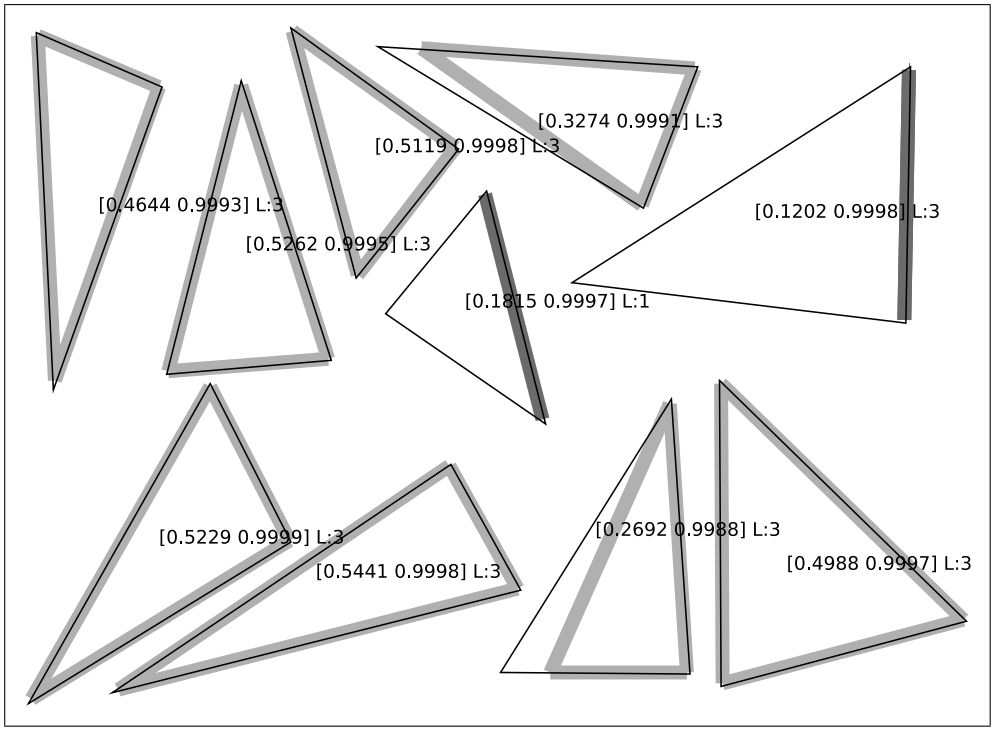

Figure 7. The output of a selected individual L_t for the triangle training data (overall fitness [0.3966,0.9995]).

clusive, the $f_p()$ objective usually does not draw close to 1.0. The reason for this, mentioned at the end of section 5.1, is that the VPs are rarely located exactly at the vertices of the recognized objects, which in turn makes it difficult for individuals to cover the object precisely with sections drawn on the canvas. Nevertheless, this does not seem to deteriorate the convergence of the algorithm, thanks to the fuzzyfication of objectives, and thanks to the fact that our dominance-based selection method takes into account only the *ordering* of the individuals on objectives $f_p()$ and $f_n()$, and not their values.

According to Fig. 6, evolution converges to solutions of a reasonable quality already in the middle of the run. The front of non-dominated solutions is non-convex for all depicted generations. This clearly indicates that relying on multiobjective fitness was a right choice: without it, many potentially valuable solutions would be depreciated by scalar evaluation.

At the end of evolution, we select the best representative from the non-dominated solutions by scalarizing the f_p and f_n objectives using weighted sum:

$$f(L) = wf_p(L) + (1 - w)f_n(L). \tag{3}$$

with $w = 0.01$ estimated experimentally. The best solution with respect to f for the triangle task, referred hereafter to as L_t, was found in 95^{th} generation

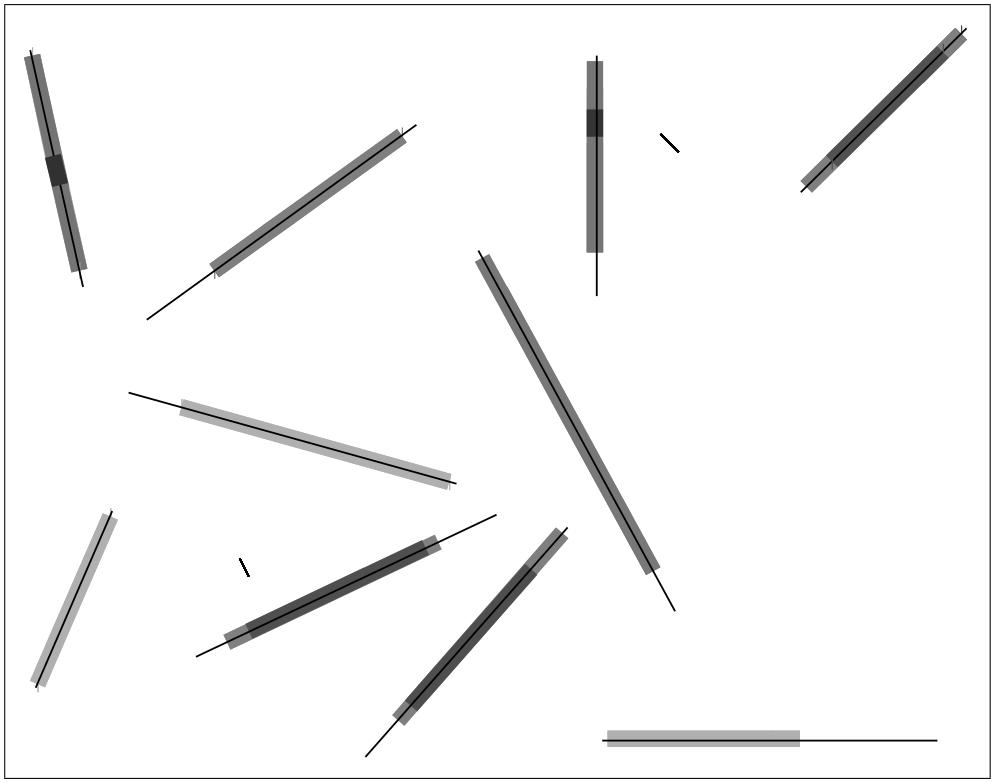

Figure 8. The output of a selected individual L_s for the section training data (overall fitness [0.3795,0.99997]).

of the evolutionary run. Similarly, the best solutions for the sections task, L_s, evolved in 92^{th} generation.

Figure 7 illustrates the resulting canvas c created by L_t for the training set of triangles. Similarly to Fig. 3, this and the following figures show the result for the entire training set superimposed in one figure. Thin lines represent input examples $s \in S$. Wide gray stripes illustrate drawing actions (DAs) issued by L_t. To improve legibility, DAs are depicted by very thick stripes, albeit in fact their width is smaller and amounts to 3 pixels. Darker stripes (or stripe fragments) reflect overlaping of multiple DAs, which negatively influences f_p and f_n (see formulas (1) and (2)). Text label attached to each triangle contains learner's evaluation for that example: numbers in square brackets correspond to f_p and f_n, while the number following 'L:' gives the actual number of DAs issued by the learner for that example. L_t identifies correctly 8 out of 10 triangles. In the incorrect cases, one triangle edge is reproduced thrice by three DAs (hence darker stripes).

Figure 8 shows an analogous result for L_s and the training set of sections. Similarly to L_t, L_s reproduces sections almost perfectly despite the presence of noise – the very short sections represented by two separate VPs, included in each training example.

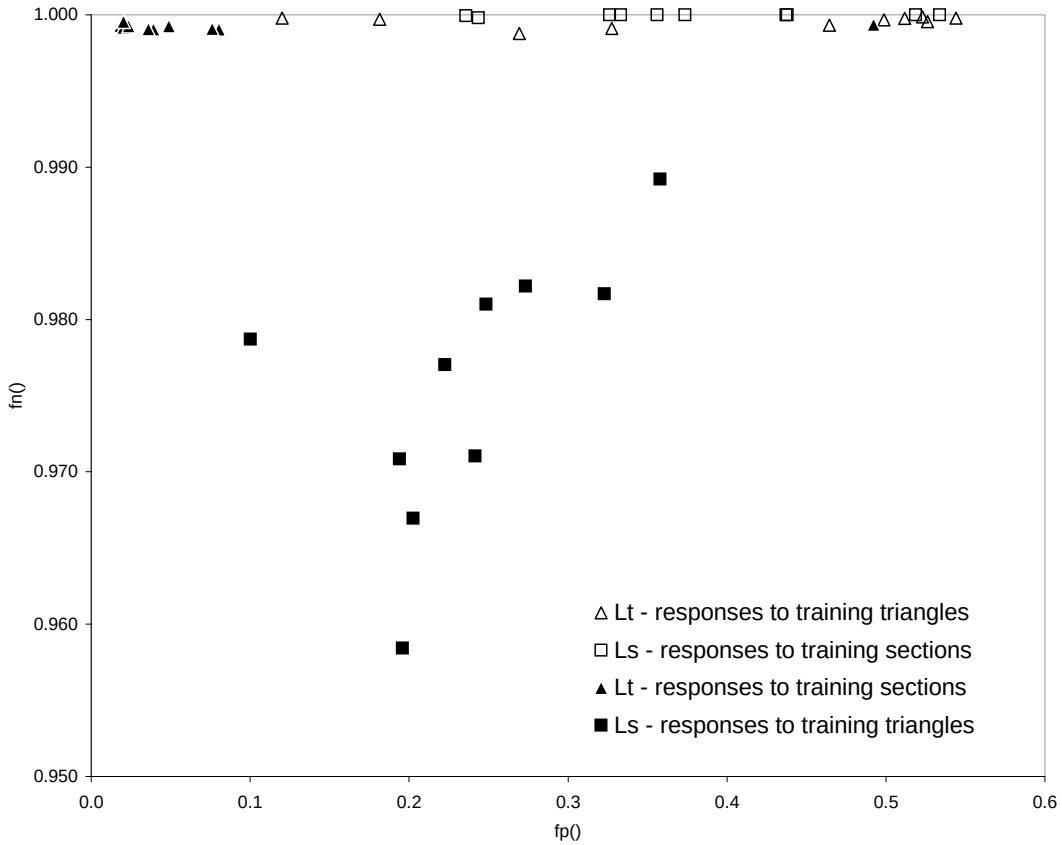

Figure 9. Cross-class evaluation: Responses of learners L_t and L_s to examples from the other decision class.

5.4 The Cross-Class Test

To verify the usefulness of the evolved solutions for the purpose of object recognition, we perform a *cross-test*, i.e., we apply L_t and L_s to examples from the other shape class (negative examples). Technically, we let L_t process all examples from the training set of sections, and, *vice versa*, we let L_s process all examples from the training set of triangles.

Figure 9 presents the quantitative interpretation of the obtained results in terms of f_p and f_n. As opposed to Fig. 6, where each point corresponds to mean performance on *all* images from the training set, here each point represents learner's response to *one* input image. Empty markers are used to depict learner's responses to examples from its positive class; for instance, empty triangular markers show the responses of L_t to training triangles. Filled markers denote learner's responses to images from its negative class; e.g., filled squares mark the responses of L_s to *triangle* images.

Figure 9 proves clearly that our learners are able to detect patterns that do not belong to their decision class. When faced with a negative example,

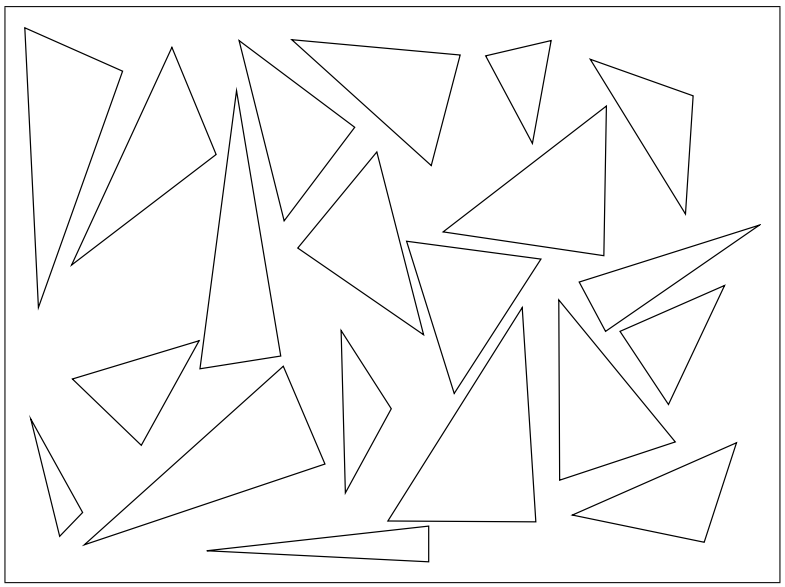

Figure 10. The testing set of triangles.

learners produce output which is, in quantitative terms, significantly inferior to the evaluation for the training (positive) examples. In particular, L_t fed with sections usually produces low f_p value (with one exception), whereas L_s produces relatively low f_n for triangles.

5.5 Test Set Verification

To verify L_t's ability to generalize beyond the training data, we evaluate it on a separate test set of 20 triangles shown in Fig. 10. Figure 11 shows L_t's responses to these examples, which allow us to conclude that 12 out of 20 testing triangles have been recognized correctly. In the remaining cases, L_t committed interpretation errors similar to those observed for two examples in the training set (see Fig. 7).

In Fig. 12 we present the quantitative results of test-set evaluation. These are presented in the same way as in Fig. 9, with each data pont marking the response of one learner to one input image. Except for two test examples, L_t's responses to testing triangles group clearly around the points corresponding to training triangles. On the contrary, L_s produces responses with significantly inferior f_n (less than 0.9973, compared to 0.99997 for training sections on the average). This clearly indicates, that, by introducing appropriate similarity measures (e.g., distance-based), or acceptance/rejection levels in the space spanned over f_p and f_n, one could perform successful recognition of these figures.

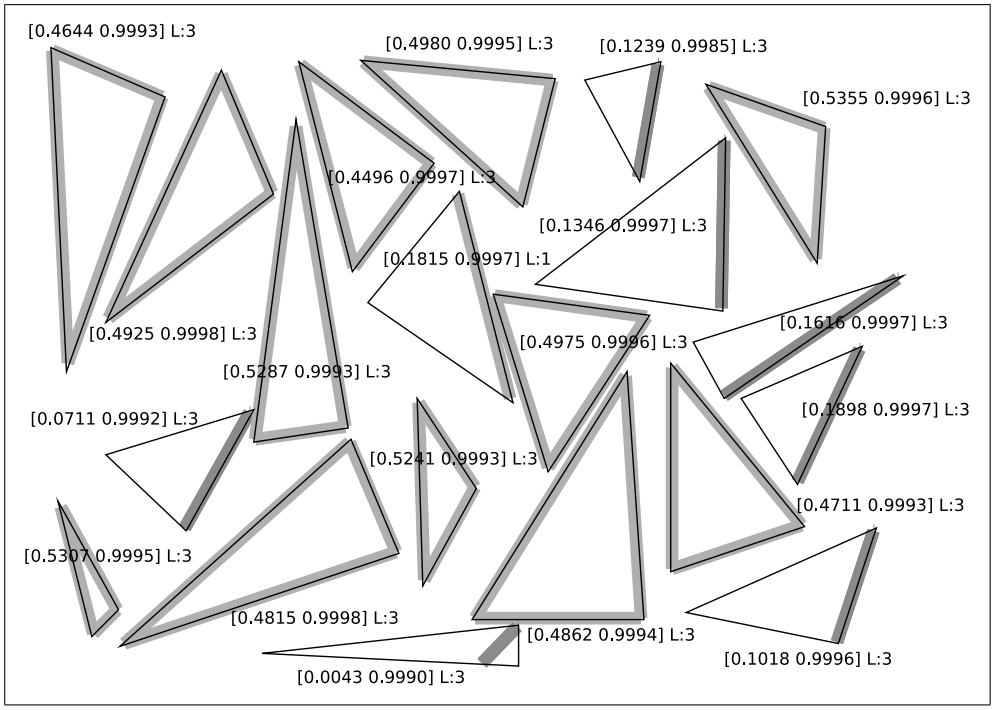

Figure 11. Test set evaluation: Responses of L_t individual to testing triangles.

5.6 Interpreting the Evolved Solutions

In this section we demonstrate that the code of evolved learners is interpretable, which makes our approach appealing when compared to non-symbolic paradigms like, e.g., neural networks. In Figures 13...16, we present the GP code of the L_t individual. To facilitate interpretation, the individual's code is divided into four parts, with Fig. 13 presenting the root part of the tree, and Figures 14, 15, and 16 depicting the left, middle, and right sub-branch, respectively, referred by oval components in Fig. 13.

The figures show both the individual's nodes (text boxes containing names of GP operators) as well as the processing it performs for an exemplary triangle. For each node, a rectangle placed atop of it illustrates the returned value – a nested set of VPs – in a graphical way (this does not apply to scalar values). Within each such box, short segments depict VPs, whereas small rectangles illustrate grouping of primitives (sets of primitives). Primitive groups are sometimes embedded in each other, which results in multiple nested rectangles. The terminal nodes labeled *ImageNode* return the original set of primitives P derived from the input image s. Terminal nodes named $v[<attribute>]$ return attribute value computed for the processed set of primitives, whereas nodes named $n[<attribute>]$ return attribute name (e.g., *Orientation*). To improve readability, two tree branches that compute scalar numerical values have been removed (nodes marked by ellipsis '...'). Finally, for the *Draw* function, placed by evolution in L_t's root node, dotted line segments reflect drawing actions.

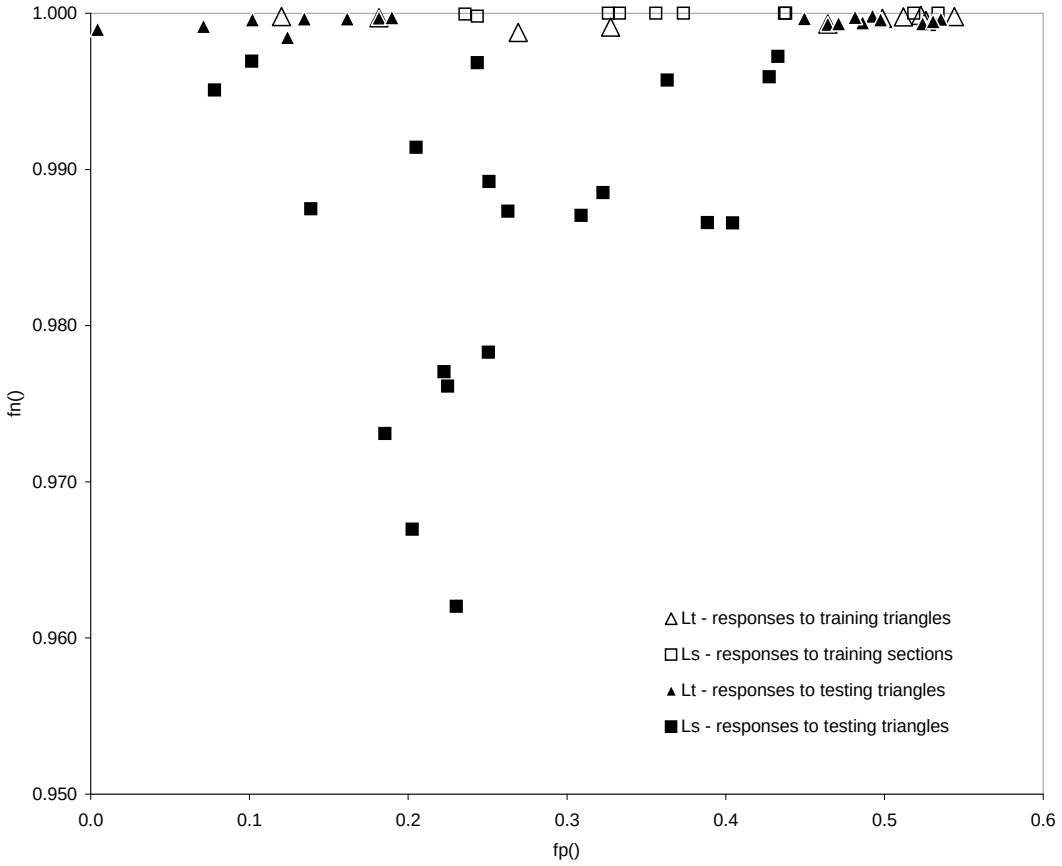

Figure 12. Test set evaluation: Responses of learners L_t and L_s to new examples of triangles.

The parts shown in Figs. 13...16 demonstrate how the learning process coped with problem decomposition and led to evolution of three sub-branches detecting particular vertices. The final part of the tree (Fig. 13) groups the primitives representing triangle vertices (the *SetUnion* operation right below the tree root) and carries out the DAs. Note also that, quite commonly for GP, a great part of individual's code is effectively 'dead', i.e., it does not contribute to the actual result of computing. Though this may seem superfluous, past research has shown that such redundancy makes individuals less prone to loss of fitness when mutated or crossed-over [27].

6 Conclusions

The proposed learning method successfully evolves image analysis procedures that are able to interpret compound geometrical patterns using very limited background knowledge. Generative aspect of the approach, implemented by means of drawing actions, enables in-depth evaluation of learner's understanding of the processed pattern. As demonstrated by exemplary solution presented

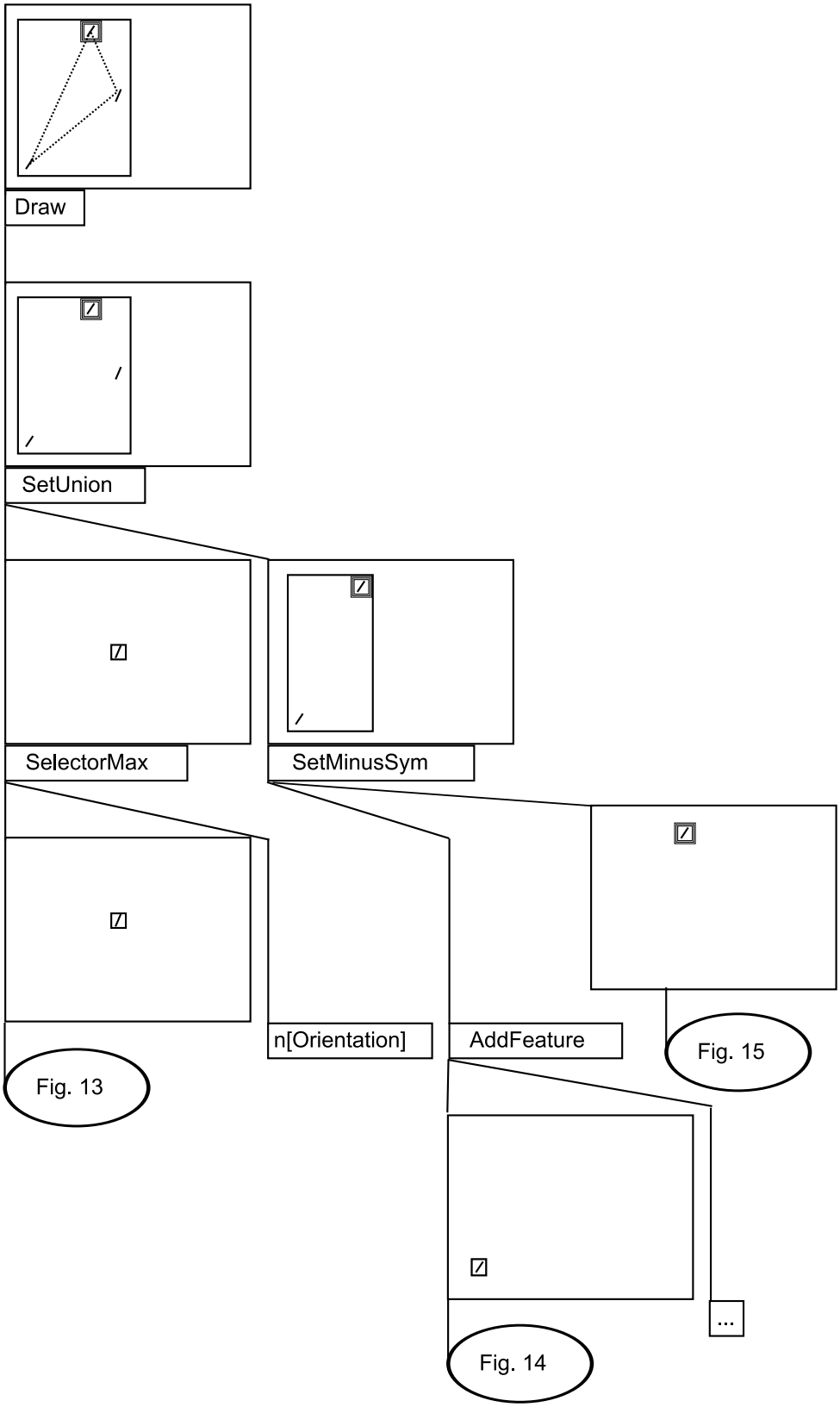

Figure 13. The root part of the learner L_t.

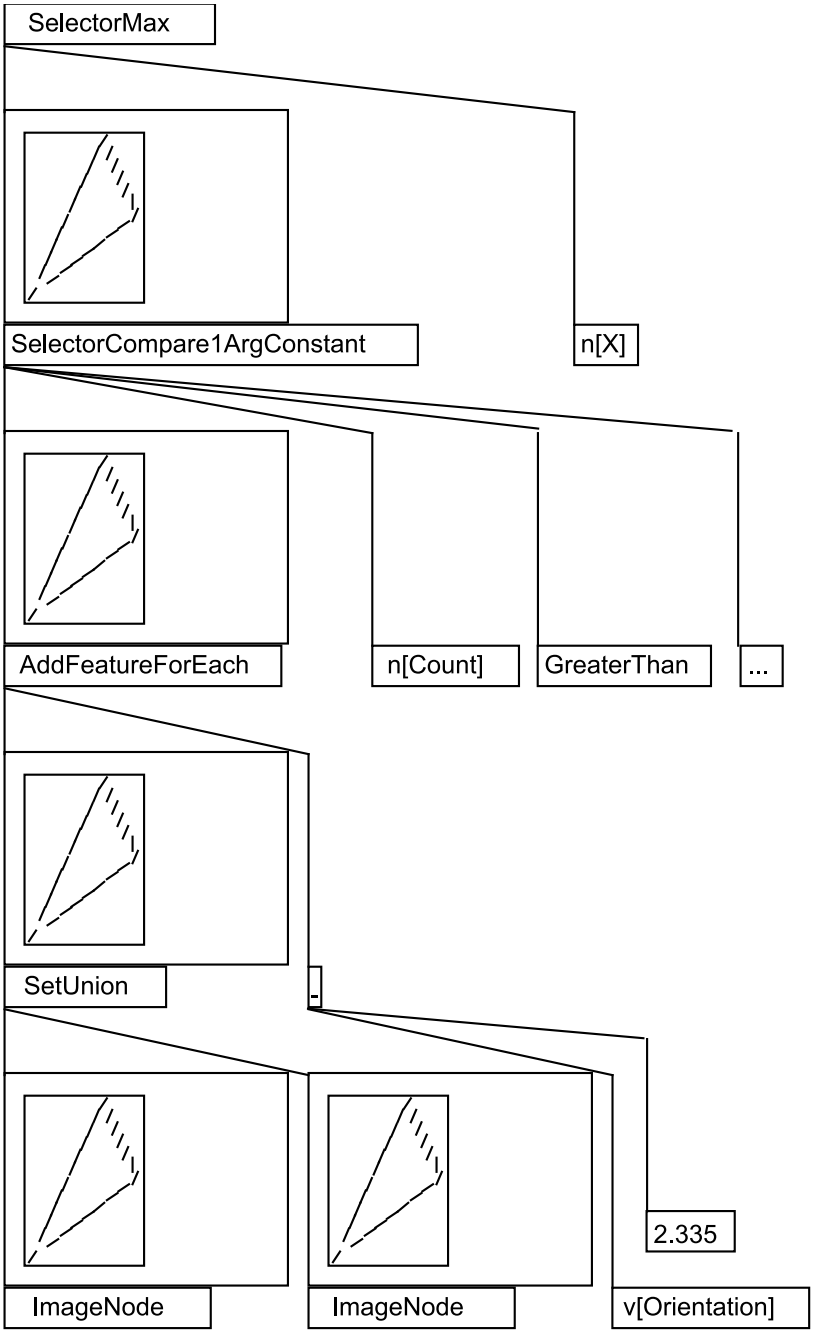

Figure 14. The left sub-branch of the learner L_t (continued from Fig. 13).

in Section 5.6, the method is able to autonomously decompose a complex recognition task into subtask. This feature potentially enables further re-use of acquired knowledge in other tasks; such possibility has been already confirmed in a preliminary study [28].

An important virtue of our method is low time complexity. This is partly due to using positive examples only, an approach known as *one-class learning* in machine learning community. This also facilitates *incremental* learning: adding

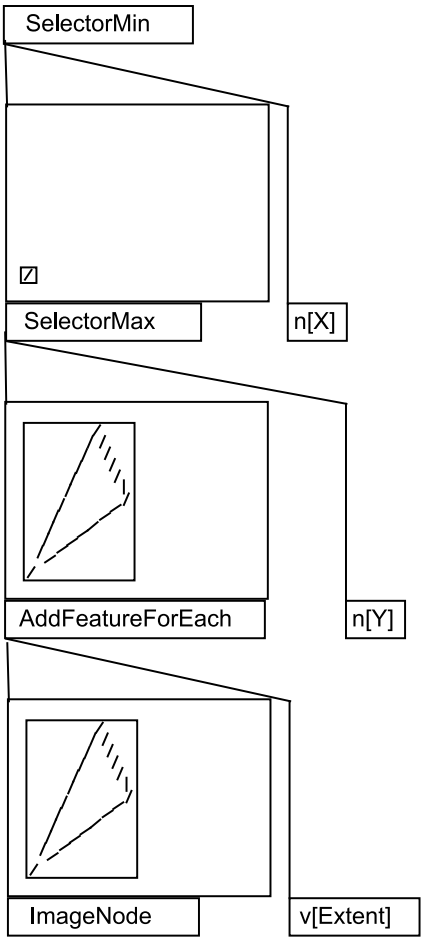

Figure 15. The middle sub-branch of the learner L_t (continued from Fig. 13).

a new a new shape class does not require re-training of the existing learners. Low time complexity results also from relatively small number of primitives processed, when compared to the amount of raster data. For instance, each training triangle is represented by only 30.3 primitives on the average. Another speedup comes from the way $f_p()$ and $f_n()$ are defined: their computation requires considering only pixels affected by drawing actions. Thanks to all these features, processing of an example takes on average only 1.2ms per individual, using our Java-based implementation running on 3.0 GHz Pentium processor. Importantly, this applies both to training (evolutionary run) and testing.

Future research on this topic could concern other aspects of visual information, like color or texture, and other input representations, like region adjacency graphs. It would be also interesting to investigate the possibility of some integration of different aspects of visual stimuli. We plan also to verify our approach on the real world task of interpreting hand-drawn sketches.

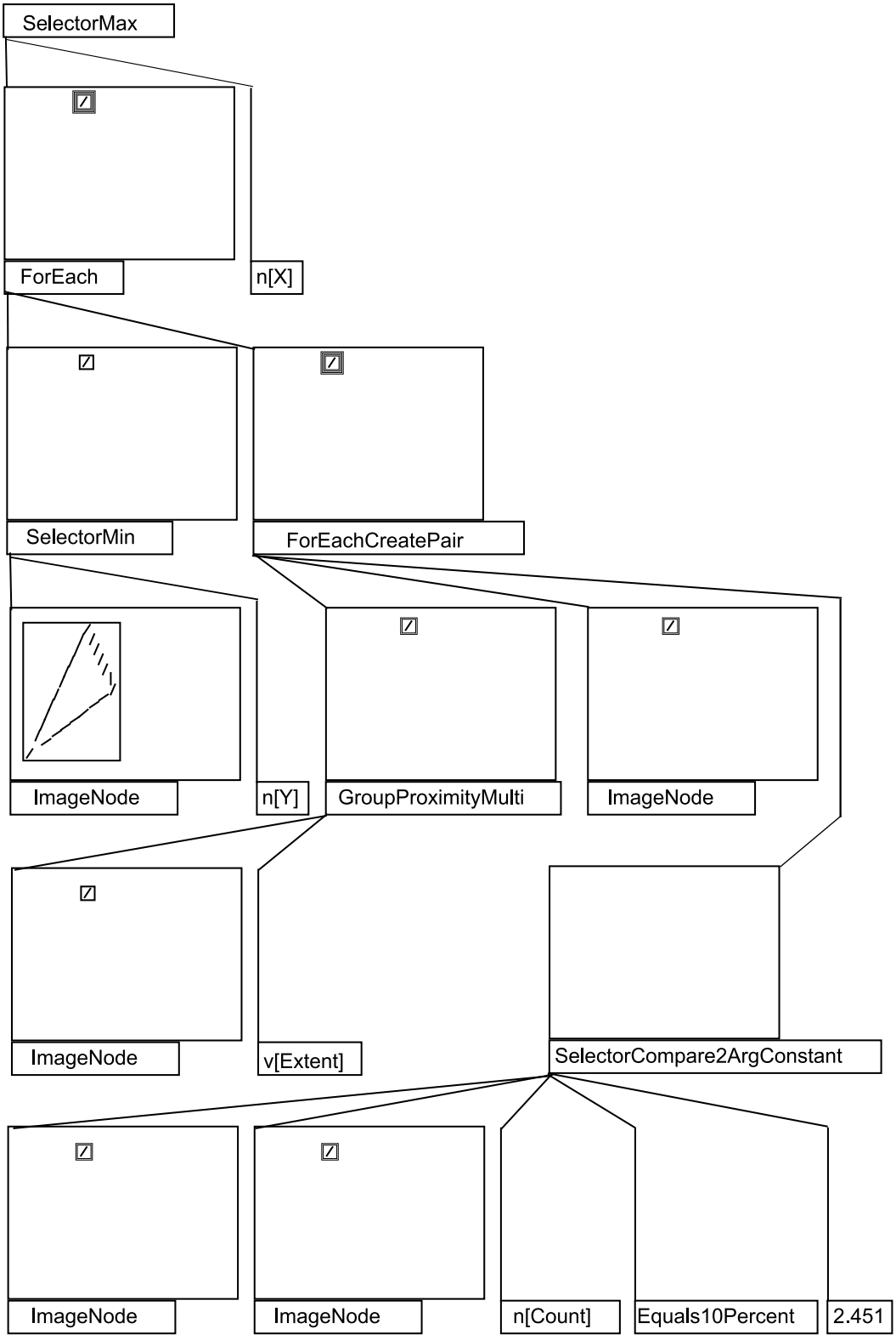

Figure 16. The right sub-branch of the learner L_t (continued from Fig. 13).

References

[1] J. Koza, Genetic programming – 2, MIT Press, Cambridge, MA, 1994.

[2] D. Goldberg, Genetic algorithms in search, optimization and machine learning, Addison-Wesley, Reading, 1989.

[3] Z. Michalewicz, Genetic algorithms + data structures = evolution programs, Springer-Verlag, Berlin, 1994.

[4] R. Michalski, G. Tecuci (Eds.), Machine learning: a multistrategy approach. Volume IV, Vol. 4, Morgan Kaufmann, Los Altos, CA, 1994.

[5] P. Langley, Elements of machine learning, Morgan Kaufmann, San Francisco, 1996.

[6] R. Kohavi, G. John, Wrappers for feature subset selection, Artificial Intelligence Journal 2 (1997) 273–324.

[7] K. Krawiec, Genetic programming-based construction of features for machine learning and knowledge discovery tasks, Genetic Programming and Evolvable Machines 4 (2002) 329–343.

[8] K. Krawiec, B. Bhanu, Visual learning by coevolutionary feature synthesis, IEEE Transactions on System, Man, and Cybernetics – Part B 35 (3) (2005) 409–425.

[9] B. Bhanu, Y. Lin, K. Krawiec, Evolutionary Synthesis of Pattern Recognition Systems, Springer-Verlag, New York, 2005.

[10] M. Revow, C. K. I. Williams, G. E. Hinton, Using generative models for handwritten digit recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 18 (6) (1996) 592–606.

[11] W. Banzhaf, P. Nordin, R. Keller, F. Francone, Genetic Programming: An Introduction. On the automatic Evolution of Computer Programs and its Application, Morgan Kaufmann, 1998.

[12] J. Koza, Human-competitive applications of genetic programming, in: A. Ghosh, S. Tsutsui (Eds.), Advances in Evolutionary Computing, Springer-Verlag, 2003, pp. 663–682.

[13] B. Draper, A. Hanson, E. Riseman, Learning blackboard-based scheduling algorithms for computer vision, Int. J. of Pattern Recognition and Artificial Intelligence 7 (1993) 309–328.

[14] M. Johnson, P. Maes, T. Darrell, Evolving visual routines, in: R. Brooks, P. Maes (Eds.), Artificial Life IV: Proc of the 4^{th} international workshop on the synthesis and simulation of living systems, MIT Press, Cambridge, MA, 1994, pp. 373–390.

[15] J. Segen, GEST: A learning computer vision system that recognizes hand gestures, in: R. Michalski, G. Tecuci (Eds.), Machine learning. A Multistrategy Approach. Volume IV, Morgan Kaufmann, San Francisco, CA, 1994, pp. 621–634.

[16] A. Teller, M. Veloso, PADO: A new learning architecture for object recognition, in: K. Ikeuchi, M. Veloso (Eds.), Symbolic Visual Learning, Oxford Press, New York, 1997, pp. 77–112.

[17] S. Luke, ECJ evolutionary computation system, (http://cs.gmu.edu/eclab/projects/ecj/) (2002).

[18] M. Rizki, M. Zmuda, L. Tamburino, Evolving pattern recognition systems, IEEE Transactions on Evolutionary Computation 6 (2002) 594–609.

[19] M. Maloof, P. Langley, T. Binford, R. Nevatia, S. Sage, Improved rooftop detection in aerial images with machine learning, Mach. Learn. 53 (2003) 157–191.

[20] A. Torralba, K. Murphy, W. Freeman, MIT-CSAIL computer vision annotated image library, Tech. rep., http://web.mit.edu/torralba/www/database.html (2004).

[21] D. Howard, S. C. Roberts, C. Ryan, Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance., Pattern Recognition Letters 27 (11) (2006) 1275–1288.

[22] J. Yu, B. Bhanu, Evolutionary feature synthesis for facial expression recognition, Pattern Recogn. Lett. 27 (11) (2006) 1289–1298.

[23] K. Krawiec, Learning high-level visual concepts using attributed primitives and genetic programming, in: F. R. (Ed.), EvoWorkshops 2006, LNCS 3907, Springer-Verlag, Berlin Heidelberg, 2006, pp. 515–519.

[24] B. Krishnapuram, C. M. Bishop, M. Szummer, Generative models and bayesian model comparison for shape recognition, in: IWFHR '04: Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition (IWFHR'04), IEEE Computer Society, Washington, DC, USA, 2004, pp. 20–25.

[25] J. Koza, Genetic Programming, MIT Press, Cambridge, MA, 1992.

[26] Java advanced imaging API specification, version 1.2, Tech. rep. (2001).

[27] H. Mayer, ptGAs – genetic algorithms evolving noncoding segments by means of promoter/terminator sequences, Evolutionary Computation 6 (4) (1998) 361–386.

[28] W. Jaśkowski, K. Krawiec, B. Wieloch, Genetic programming for cross-task knowledge sharing, in: D. T. et al. (Ed.), Genetic and Evolutionary Computation Conference GECCO, Association for Computing Machinery, 2007, pp. 1620–1627.