Evolutionary Learning with Cross-Class Knowledge Reuse for Handwritten Character Recognition

Wojciech Jaśkowski, Krzysztof Krawiec, and Bartosz Wieloch

Institute of Computing Science, Poznan University of Technology
Piotrowo 2, 60965 Poznań, Poland
{wjaskowski|kkrawiec|bwieloch}@cs.put.poznan.pl

Abstract. We propose a learning algorithm that reuses knowledge acquired in past learning sessions to improve its performance on a new learning task. The method concerns visual learning and uses genetic programming to represent hypotheses, each of them being a procedure that processes visual primitives derived from the training images. The process of recognition is generative, i.e., a procedure is supposed to restore the shape of the processed object by drawing its reproduction on a separate canvas. This basic method is extended with a knowledge reuse mechanism that allows learners to import genetic material from hypotheses that evolved for the other decision classes (object classes). We compare both methods on a task of handwritten character recognition, and conclude that knowledge reuse leads to significant improvement of classification accuracy and reduces the risk of overfitting.

1 Introduction

Most of contemporary machine learning (ML) algorithms are designed to process *isolated* learning tasks. Usually, a learning algorithm (*inducer*) produces a *classifier* based exclusively on the training data provided for particular learning task. In that process, the inducer relies on fixed inductive bias (priors) that does not change from task to task. There is no way of reusing the knowledge that the inducer could have acquired when inducing classifiers in the past.

This limitation is conspicuously inconsistent with the human way of learning, which is always based on individual's past experience. Priors in human learning come from one's history of dealing with similar tasks. More than that, acquiring new skills in isolation from past experience is impossible for humans. By reusing knowledge, humans can successfully learn in demanding conditions, e.g., when the number of training examples is small or in presence of data inconsistency.

The ability to reuse knowledge would be definitely a virtue for a machine learning system, speeding up the convergence of the learning process, reducing the risk of overfitting, and keeping down the number of training examples required to learn the concept. Some of these benefits have been already demonstrated in related studies on, e.g., multitask learning [1]. However, the last decade

did not bring breakthrough in this topic, and knowledge reuse is still listed among the most challenging issues in ML [14].

Incapability of ML systems to reuse knowledge is due to several reasons. Firstly, in the most popular paradigm of inductive learning from examples described by attribute-value pairs, it is difficult to identify universal, or even domain-specific knowledge. Attributes describing examples are highly task-specific, which reduces chances of finding their counterparts in another learning task. Secondly, many knowledge representations used in ML make it difficult to modularize or transfer knowledge. For instance, there is little chance for a fragment of a neural network to be useful in another network taught to solve a different learning task.

Challenged by these limitations, in this paper we exploit the paradigm of genetic programming (GP, [7]) as a vehicle for knowledge reuse. We demonstrate that GP is a convenient platform for this purpose, due to, among others, the symbolic knowledge representation and the ability of abstraction from a specific context. In particular, we use our method presented in [5] that implements knowledge reuse between evolving learners by allowing them to cross over parts of their genetic material. We apply the method to a large-scale task of visual learning (handwritten character recognition) and show that knowledge reuse improves the convergence of the learning process and prevents overfitting.

2 Related Work

Reported research on knowledge reuse concerns mostly knowledge reuse within a *single* learning task, with the exception of multitask learning [1] and meta-learning [19], which mostly concern learning from fixed-length attribute-value representation. In the context of GP, knowledge reuse is often connected with knowledge encapsulation [8,17,2,4], which is however not used in the approach presented here. Among reported approaches, the Case Injected Genetic Algorithm (CIGAR) by Louis *et al.* [11] resembles our contribution the most. In CIGAR, the experience of the system is stored in a form of solutions to problems solved earlier ('cases'). When confronted with a new problem, CIGAR evolves a new population of individuals and injects it periodically with such remembered cases. Experiments demonstrated CIGAR's superiority to standard GA in terms of search convergence. However, CIGAR injects *complete* solutions only, works in a strictly sequential way, and does not involve GP, making it significantly different from our approach.

In this paper, we investigate *visual* learning. As image interpretation is inherently complex, it is difficult to devise a learning method that solves such a task as a whole. Rather than that, most methods proposed so far introduce some learning or adaptation at a particular stage of image processing and analysis, which enables easy interfacing with the remaining components of the recognition system. For instance, training a machine learning classifier on some predefined image features is a typical example of such an approach. In this paper, we propose a learning method that spans the *entire* processing chain, from the input

image to the final decision making, and produces a complete recognition system. Former research on such systems is rather scant [18,16,13,10,3].

3 Generative Visual Learning

The approach, originally proposed in [9] and further developed in [6], may be shortly characterized as *generative visual learning*, as our evolving learners aim at *reproducing the input image* using some simpler means. Reproduction takes place on a virtual canvas spanned over the input image. On that canvas, the learner (genetic programming individual) is allowed to perform some elementary *drawing actions* (DAs for short) in response to the input image. In particular, we consider handwritten characters and, to enable learners to restore their shapes, our DAs line sections. Fitness function compares the contents of the canvas to the input image, and rewards individuals that provide high quality of reproduction. Thus, an individual is here awarded not for the final result of decision making only, but for its 'understanding' of the pattern being analyzed.

Such an evaluation method allows us to examine the processing performed by an individual-learner in a more thorough way than in non-generative approach, where individuals are expected to produce a scalar feature or a binary decision in response to the input image. Thanks to that, the risk of overfitting, so immense in learning from high-dimensional image data, becomes significantly smaller.

To reduce the amount of data that has to be processed, our approach abstracts from raster data and relies only on selected salient features found in the input image s. The features correspond to prominent local luminance gradients derived from s using a straightforward procedure described in [5]. For each detected feature, we build a *visual primitive* (VP), described by three scalars called hereafter *attributes*; these include two spatial coordinates of the edge fragment and the local gradient orientation. The complete set of VPs derived from s, denoted in the following by P, is usually several orders of magnitude more compact than the original image s, yet it well preserves the sketch of s.

On the top level, the proposed method uses evolutionary algorithm that maintains a population of visual learners (individuals, solutions), each of them implemented as GP expression. Each learner L is a procedure written in a form of a tree, with nodes representing *functions* that process sets of VPs. The terminal nodes (named *ImageNode*s) fetch the set of primitives P derived from the input image s, and the consecutive internal nodes process the primitives, all the way up to the root node. We use strongly-typed GP (cf. [7]), which implies that two nodes may be connected to each other only if their input/output types match. The following types are used: numerical scalars, sets of VPs, attribute labels, binary arithmetic relations, and aggregators.

The GP functions may be divided into scalar functions, selectors (select some VPs based on their attributes), iterators (process VPs one by one), and grouping operators (group VPs based on their attributes and features, e.g., spatial proximity). Given these operators, an individual-learner L applied to an input image s builds a hierarchy of VP sets derived from s. Each invoked tree node

creates a new set of VPs that includes other elements of the hierarchy. In the end, the root node returns a nested VP hierarchy built atop of P, which reflects the processing performed by L for s. A more detailed description of this process, including the full list of GP functions, may be found in [9].

Individual's fitness is based on DAs (drawing actions) that it performs in response to visual primitives P derived from training images $s \in S$. To reconstruct the essential features of the input image s, the learner is allowed to perform DAs that boil down to drawing sections on the output canvas c. To implement that within the GP framework, we introduce an extra GP function called *Draw*. It expects as an argument one VP set T and returns it unchanged, drawing on canvas c sections connecting each pair of VPs from T. Evaluation of L consists in comparing the contents of c to s and assumes that the difference between c and s is proportional to the minimal total cost of bijective assignment of lit pixels of c to lit pixels of s. The total cost is a sum of costs for each pixel assignment. The cost of assignment depends on the distance between pixels: when the distance is less than 5, the cost is 0; maximum cost equals 1 when the distance is greater than 15; between 5 and 15 the cost is a linear function of the distance. For pixels that cannot be assigned (e.g., because there are more lit pixels in c than in s), an additional penalty of value 1 is added to the total cost. In order to compute the minimal total cost of assignment, an effective greedy heuristic was applied.

The (minimized) fitness of L is defined as the total cost of the assignment normalized by the number of lit pixels in $s \in S$, averaged over the entire training set of images S. An ideal learner perfectly restores shapes in all training images and its fitness amounts to 0. The more the canvas c produced by L differs from s, the greater (worse) its fitness value. Thus, fitness function rewards individuals that exactly and completely reproduce as many images from S as possible, therefore promoting discovery of similarities between the training images.

In terms of ML, this generative visual learning (GVL) procedure performs *one-class learning* [15], as it uses training examples from the positive class only and tries to describe it, having no idea about the existence of other decision classes (object classes in case of visual learning). To handle a k-class classification problem, we run in parallel k independent evolutionary processes for n generations, each of them devoted to one object class. The k best individuals obtained from particular runs form the complete multi-class classifier (*recognition system*), ready to recognize new images using a straightforward voting procedure detailed in Section 5.

4 Knowledge Reuse

Given the similar visual nature of learning tasks related to particular decision classes in GVL, we expect them to require some common knowledge. Therefore, running them in isolation may be redundant, as many decision classes may need similar fragments of GP code to, e.g., detect the important features like stroke junctions. For instance, locating the lower end of the shape of letter Y presented in an image may require similar subtree of GP operators as locat-

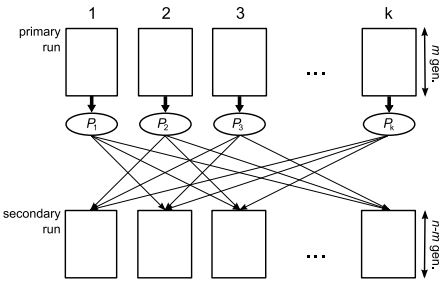

Fig. 1. The architecture of CCKR.

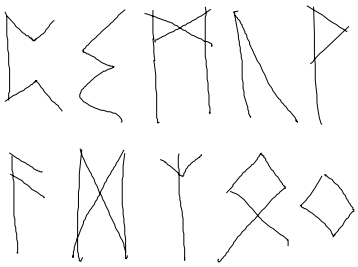

Fig. 2. Examples of handwritten runes.

ing the lower ends of letter X. To exploit such commonalities, we enable GVL to involve *cross-class knowledge reuse* (CCKR) between evolutionary processes devoted to particular classes. For the initial m generations ($m < n$), called hereafter *primary run*, evolution proceeds exactly as in GVL. As the run devoted to i^{th} decision class ($i = 1...k$) reaches the m^{th} generation, we store its population in a *pool* P_i, so that P_i constitutes a snapshot of i^{th} evolutionary run at m^{th} generation. Next, the population is re-initialized (in the same way as the initial population of the primary run), and the evolution continues for the remaining $n - m$ generations, referred to as *secondary run*.

The secondary run slightly differs from the primary one in that it activates an extra *crossbreeding operator* that is allowed to import genetic material from the pools P_i. Crossbreeding works similarly to GP crossover, however, it interbreeds an individual from the current population (a 'native') with an individual from one of the pools P_j, $j \neq i$ (an 'alien'). First, it selects a native parent from the current population using the same selection procedure as crossover. Then, it picks out an alien parent by first randomly choosing one of the pools P_j, $j \neq i$, and then randomly selecting an individual from P_j, disregarding its fitnesses. Finally, crossbreeding randomly selects two nodes N_n and N_a in the native and the alien parent, respectively, and replaces N_n by the subtree rooted in N_a. The modified native parent (offspring) is injected into the subsequent population (provided it meets the constraints optionally imposed on GP trees). Thus, crossbreeding may involve large portions of code as well as small code fragments.

Figure 1 outlines the CCKR approach. Each column composed of primary and secondary run relates to learning one decision class, and arrows depict the transfer of genetic material between them. As GVL requires k runs lasting n generations each, while CCKR involves k runs lasting m generations and k runs lasting $n - m$ generations, the total number of fitness function calls (the effort) is the same. Thus, if we ignore the cost of re-initialization of k populations and the cost of cross-breeding, which are in fact very low compared to overall computation, the time complexity of CCKR is the same as that of GVL on the average (though the actual evolution time may vary due to variability of fitness computation time). As the pools P_is, once created, remain unchanged, the runs do not have to work in parallel, but may be carried out sequentially.

5 The Experiment

In experimental part, we approach a real-world multiclass problem of handwritten character recognition. The task is to recognize letters from the Elder Futhark, the oldest form of the runic alphabet, which consists of the following characters:

ᚠ ᚢ ᚦ ᚨ ᚱ ᚲ ᚷ ᚹ ᚺ ᚾ ᛁ ᛃ ᛇ ᛈ ᛉ ᛊ ᛏ ᛒ ᛖ ᛗ ᛚ ᛜ ᛞ ᛟ

Elder Futhark letters are written with straight pen strokes only, which makes them a good testbed for our generative recognition approach that uses sections to reconstruct the recognized shapes. Using a TabletPC computer, we prepared a training set containing 240 images (examples, objects) of $k = 24$ runic alphabet characters, each character class represented by 10 examples written by 7 persons (three of them provided two character sets). Figure 2 shows examples of selected handwritten characters.

The purpose of the experiment is to compare CCKR, the method with knowledge reuse, to the basic approach (GVL) that provides us with control results. Technically, we use generational evolutionary algorithm maintaining a population of 10,000 GP individuals for $n = 600$ generations. The initial population is created using Koza's ramped half-and-half operator with ramp from 2 to 6 [7]. We apply tournament selection with tournament of size 5, using individuals' sizes for tie breaking and thus promoting smaller GP trees and alleviating the problem of code bloat. For GVL runs, offspring are created by crossing over selected parent solutions from previous generation (with probability 0.8), or mutating selected solutions (with probability 0.2). For CCKR, the primary run lasts for $m = 300$ generations with the same settings as GVL, while in the secondary run the mutation probability is lowered to 0.1 to yield 0.1 to the crossbreeding operator. The GP tree depth limit is set to 10; the mutation and crossover operations may be repeated up to 5 times if the resulting individuals do not meet this constraint; otherwise, the parent solutions are copied into the subsequent generation. Except for the fitness function implemented for efficiency in C++, the algorithm has been implemented in Java with help of the ECJ package [12]. For evolutionary parameters not mentioned here, ECJ's defaults have been used.

To intensify the search, we split the population into 10 islands and exchange individuals between them every 20^{th} generation starting from the 50^{th} generation. During exchange, each odd-numbered island donates 10% of its well-performing individuals (selected by tournament of size 5) to five even-numbered islands, where they replace the poorly-performing individuals selected using an inverse tournament of the same size. The even-numbered islands donate their representatives to the odd-numbered islands in the same way. The islands should *not* be confused with the boxes depicting evolutionary runs in Fig. 1 – the island model is implemented within *each* evolutionary process independently.

Using these settings, we evolve and compare CCKR recognition systems to GVL recognition systems, with the latter serving as control results. In both cases, this involves running $k = 24$ evolutionary processes, each of them using training examples from one character class for fitness computation. The ensemble

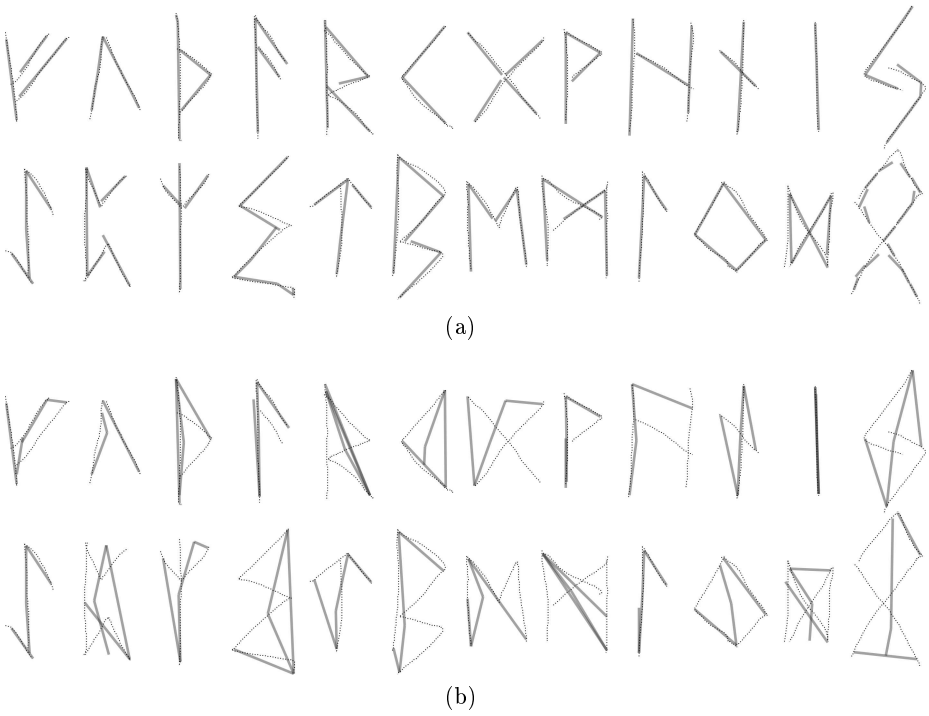

Fig. 3. Examples of letter reconstructions. The original images are drawn with a thin dotted line. In (a) each letter was reconstructed by a individual taught on a appropriate class, whereas in (b) an attempt was made to reconstruct all shapes using an individual taught on class ᛇ .

of all 24 best-of-run individuals constitute the complete *recognition system*. The recognition system undergoes evaluation on the testing set of characters, which is disjoint with the training set and contains 1440 images, that is 60 images for each character class. The system classifies an example t by computing fitnesses (responses) of all individuals for t and indicating the class associated with the individual that responded in the smallest (the best) value. Such procedure is motivated by an obvious observation, that a learner is taught to perform well on images from one class and its raw (minimized) fitness should be near 0 only for images of this class. For example, in Fig. 3a, each character was reconstructed using individual taught on its coresponding class, so all the reconstructions are good. On the other hand, in Fig. 3b, where each shape was reconstructed using the individual taught on class ᛇ , only the restoration of character ᛇ is correct[1].

Such a *simple recognition system* may be obtained at a relatively low computational expense of k evolutionary runs. Given more runs, recognition accuracy

[1] Though also restoration of ᛁ seems correct, closer examination reveals surplus overlying strokes that will be penalized by the fitness measure.

Table 1. Test-set classification accuracy for different voting methods.

Voting method	simple	vote-2	vote-3	vote-4	vote-5	vote-30
GVL	69.79±1.66	78.50±1.12	82.50±1.02	85.21±0.79	86.66±0.61	91.32
CCKR	81.94±0.89	87.88±0.49	91.19±0.41	92.58±0.31	93.18±0.27	95.56

Table 2. True positive (TP) and false positive (FP) ratios for vote-30 CCKR method.

Letter	ᚠ	ᚢ	ᚦ	ᚨ	ᚱ	ᚲ	ᚷ	ᚹ	ᚺ	ᚾ	ᛁ	ᛃ
TP	90.0%	80.0%	100%	95.0%	98.3%	100%	98.3%	81.7%	100%	93.3%	98.3%	98.3%
FP	0.0%	18.3%	21.7%	1.7%	0.0%	1.7%	3.3%	3.3%	0.0%	6.7%	8.3%	0.0%
Letter	ᛇ	ᛈ	ᛉ	ᛊ	ᛏ	ᛒ	ᛖ	ᛗ	ᛚ	ᛜ	ᛞ	ᛟ
TP	100%	98.3%	93.3%	100%	95.0%	100%	100%	96.7%	80.0%	100%	100%	96.7%
FP	3.3%	1.7%	0.0%	1.7%	3.3%	0.0%	3.3%	0.0%	21.7%	6.7%	0.0%	0.0%

may be further boosted by employing more *voters* per each decision class, as opposed to one voter per class in the above scheme. This is especially appealing in the context of evolutionary computation, as each evolutionary run usually produces a different best-of-run individual, so their fusion may result in synergy. Table 1 presents the test-set classification accuracy of GVL and CCKR for the simple recognition system and the voting method with a different number of voters. The table shows averages with .95 confidence intervals; for vote-30, confidence intervals cannot be provided as, with 30 independent runs for each character class, only one unique vote-30 recognition system can be built. Quite obviously, the more voters, the better the performance. But more importantly, no matter what the number of voters is, CCKR impressively outperforms the corresponding GVL approach. The gap between them narrows when the number of voters increases, but remains significant even for the vote-30 case.

Table 2 presents detailed test set results for the vote-30 CCKR experiment. Nine out of 24 characters were recognized perfectly. Overall, only three characters were recognized in less than 90% of cases: ᚢ , ᚹ , and ᚨ . Basing on the complete confusion matrix (not shown here due to the lack of space), we can conclude that ᚢ is sometimes mistaken for ᚨ , hence both yield high false positive errors. Similarly, the recognition system occasionally incorrectly classifies ᚹ as ᚦ . Since these pairs of letters are very similar to each other and even a human might find them troublesome, this result may be considered appealing.

6 Conclusions

Despite the large number of decision classes (24), low number of training examples per class (10), variability of handwriting styles (7 persons), and, last but not least, one-class learning (learners have no idea what the negative examples look like), the presented approach of GP-based generative visual learning per-

forms surprisingly well, attaining near-to-perfect classification accuracy on the large and diversified test set. This encouraging result should be attributed to the generative trait of the proposed approach, which does not allow the evolving learners to 'skim' the training data for *any* discriminating feature, but forces them to fully 'understand' the recognized pattern.

The additional mechanism of cross-class knowledge reuse (CCKR) further boosts the recognition accuracy, even when the base approach (GVL) already uses an extensive voting procedure like vote-30. Most of erroneous recognitions concern character classes that are hard to tell apart also for humans. With the difficult character classes excluded (ﻸ and ﻷ , ﻹ and ﻺ), CCKR attains 98.0% recognition accuracy on the test set.

CCKR is straightforward and may be implemented by a relatively simple extension of canonical genetic programming. The method does not increase the computational effort of the learning process, and provides a significant performance improvement at the same computational expense as GVL. Thanks to one-class learning, recognition system may be easily extended by a new decision class by separately evolving an extra learner; the existing components of the recognition system do not have to be modified.

At the current stage, it is difficult to conclude if the results obtained here generalize to other variants of GP-based learning. The conservative answer should be probably negative: many other GP-based learning methods would benefit less from this form of knowledge reuse. For instance, GP expressions operating in the space of attributes in conventional learning from examples, would probably be not beneficiary of CCKR, for the reasons stated earlier in Introduction (low probability of usefulness of GP subexpressions in other learning tasks). However, this tentative conclusion should not restrain us from further investigation of the topic. Quite on the contrary, it may be a useful hint for building GP representations that are susceptible to knowledge reuse. In other words, it would be interesting to pose an inverse problem: instead of trying to devise a knowledge reuse method for a particular knowledge representation, try to define a knowledge representation that makes the knowledge reuse possible.

Acknowledgments

This work was supported by grant DS 91-443. Computations were carried out in Poznań Supercomputing and Networking Center.

References

1. R. Caruana. Multitask learning. *Mach. Learn.*, 28(1):41–75, 1997.
2. E. Galvan Lopez, R. Poli, and C. A. Coello Coello. Reusing code in genetic programming. In M. K. *et al.*, editor, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 359–368. Springer-Verlag, 2004.

3. D. Howard, S. C. Roberts, and C. Ryan. Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. *Pattern Recognition Letters*, 27(11):1275–1288, 2006.
4. W. H. Hsu, S. J. Harmon, E. Rodriguez, and C. Zhong. Empirical comparison of incremental reuse strategies in genetic programming for keep-away soccer. In M. Keijzer, editor, *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, 26 July 2004.
5. W. Jaśkowski, K. Krawiec, and B. Wieloch. Knowledge reuse in genetic programming applied to visual learning. In D. Thierens, editor, *Genetic and Evolutionary Computation Conference GECCO*, pages 1790–1797. Association for Computing Machinery, 2007.
6. W. Jaśkowski, K. Krawiec, and B. Wieloch. Learning and recognition of hand-drawn shapes using generative genetic programming. In M. G. et al., editor, *EvoWorkshops 2007*, volume 4448 of *LNCS*, pages 281–290, Berlin Heidelberg, 2007. Springer-Verlag.
7. J. Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1992.
8. J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming. In T. H. *et al.*, editor, *Proceedings of International Conference on Evolvable Systems: From Biology to Hardware (ICES-96)*, volume 1259 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
9. K. Krawiec. Learning high-level visual concepts using attributed primitives and genetic programming. In F. R., editor, *EvoWorkshops 2006*, LNCS 3907, pages 515–519, Berlin Heidelberg, 2006. Springer-Verlag.
10. K. Krawiec and B. Bhanu. Visual learning by coevolutionary feature synthesis. *IEEE Transactions on System, Man, and Cybernetics – Part B*, 35(3):409–425, June 2005.
11. S. Louis and J. McDonnell. Learning with case-injected genetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 8(4):316–328, 2004.
12. S. Luke. ECJ evolutionary computation system, 2002. (http://cs.gmu.edu/eclab/projects/ecj/).
13. M. Maloof, P. Langley, T. Binford, R. Nevatia, and S. Sage. Improved rooftop detection in aerial images with machine learning. *Mach. Learn.*, 53:157–191, 2003.
14. T. M. Mitchell. The discipline of machine learning. Technical Report CMU-ML-06-108, Machine Learning Department, Carnegie Mellon University, July 2006.
15. K. M. W. Moya, M. R. and L. D. Hostetler. One-class classifier networks for target recognition applications. In *Proceedings world congress on neural networks*, pages 797–801, Portland, OR, 1993. International Neural Network Society.
16. M. Rizki, M. Zmuda, and L. Tamburino. Evolving pattern recognition systems. *IEEE Transactions on Evolutionary Computation*, 6:594–609, 2002.
17. S. C. Roberts, D. Howard, and J. R. Koza. Evolving modules in genetic programming by subtree encapsulation. In J. F. M. *et al.*, editor, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 160–175. Springer-Verlag, 2001.
18. A. Teller and M. Veloso. PADO: A new learning architecture for object recognition. In K. Ikeuchi and M. Veloso, editors, *Symbolic Visual Learning*, pages 77–112. Oxford Press, New York, 1997.
19. R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artif. Intell. Rev.*, 18(2):77–95, 2002.