Knowledge Reuse for an Ensemble of GP-Based Learners

Wojciech Jaśkowski, Krzysztof Krawiec, Bartosz Wieloch

Institute of Computing Science, Poznan University of Technology, Poznań, Poland

Abstract. We propose a method of knowledge reuse for an ensemble of genetic programming-based learners solving a visual learning task. First, we introduce a visual learning method that uses genetic programming individuals to represent hypotheses. Individuals-hypotheses process image representation composed of visual primitives derived from given training images that contain objects to be recognized. The process of recognition is generative, i.e., an individual is supposed to restore the shape of the processed object by drawing its reproduction on a separate canvas. This canonical method is in the following extended with a knowledge reuse mechanism that allows a learner to import genetic material from hypotheses that evolved for other decision classes (object classes). We compare the performance of the extended approach to the basic method on a real-world tasks of handwritten character recognition, and conclude that knowledge reuse leads to significant convergence speedup and reduces the risk of overfitting.

1 Introduction

Standard machine learning (ML) algorithms do not accumulate knowledge when faced with consecutive learning tasks, and work with fixed *learning biases*. The ability to detect and reuse universal background knowledge (a.k.a. common-sense knowledge), or more specific domain-related knowledge, would speed up the convergence of the learning process, reduce the risk of overfitting, and keep down the number of required training examples. Thus, the ability of knowledge reuse is essential for further progress in ML and is quoted among its most challenging and important issues [13].

There are several reasons for the inability of most of the standard ML systems to identify and reuse knowledge fragments. Firstly, in the most popular paradigm of inductive learning from attributed examples, it is difficult to identify universal knowledge (a.k.a. *inductive bias* in inductive learning). Secondly, knowledge is in general difficult to modularize or transfer. For instance, there is little chance for a fragment of a neural network to be useful at another location of the same network, or a different network delegated to solve another learning task. This is also due to the fact that, in most ML problems, attributes that describe examples are highly task-specific, which reduces chances of finding their analogs in another learning task.

Inspired by these limitations, in this paper we exploit the paradigm of genetic programming (GP, [7]) as a vehicle for knowledge reuse. GP offers an excellent platform

for knowledge transfer due to symbolic representation of solutions and the ability of abstraction from a specific context. Those properties, together with a built-in mechanisms of knowledge propagation by inheritance, enable advanced and effective knowledge manipulation. We demonstrate that a relatively simple mechanism of knowledge reuse introduced between related visual learning tasks improves dramatically the convergence of the learning process and, in consequence, prevents overfitting.

2 Related Work

Though knowledge reuse is definitely an important issue in computational intelligence, it has so far attracted relatively little attention. Reported research concerns mostly knowledge reuse within a *single* learning task, with the exception of limited work on multitask learning [1], which predominantly uses neural nets for knowledge representation. In the context of GP, knowledge reuse is often connected with knowledge encapsulation [6, 15, 2, 4], which is however not used in the approach presented here. Among the reported contributions, the work done by Louis *et al.* most resembles our contribution [11, 10]. In particular, in Case Injected Genetic Algorithms (CIGAR) described in [10], 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 and does not involve GP.

The work presented here is also partially related to visual learning. As image interpretation is an inherently complex task, 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 learning or adaptation only at a particular stage of image processing and analysis, which enables easy interfacing with the remaining components of the recognition system. For instance, a machine learning classifier learning from 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 [14, 9, 3].

3 Generative Visual Learning

The proposed method uses evolutionary algorithm that maintains a population of visual learners (solutions) implemented as GP individuals. Each learner performs *generative learning*, aiming at reproducing (re-generating) the input image, and is rewarded according to the quality of that reproduction. This process is technically implemented by allowing the learner to perform some elementary *drawing actions* (DAs for short) on a virtual canvas spanned over the input image. In this paper in particular, individuals learn to recognize shapes of characters, and the DAs boil down to drawing of sections.

The approach abstracts from raster data and relies only on selected salient features in the input image s. For each locally detected feature, we build an independent *visual primitive* (VP for short). The complete set of VPs derived from s is denoted in the following by P. As in this paper we focus on shape, we use VPs representing prominent local luminance gradients derived from s using a straightforward procedure. Each VP is described by three scalars called hereafter *attributes*; these include two spatial coordinates

of the edge fragment and the local gradient orientation. Details on VP extraction may be found in [5, 16].

Technically, each visual learner L is a procedure written in a form of a tree, with nodes representing *elementary operators* that process sets of VPs. The terminal nodes (named *Input*) 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. The non-terminal GP operators may be divided into the following categories: *scalar operators* (implement basic arithmetics as in symbolic regression; see [8]), *selectors* (filter VPs according to some objectives or condition), *iterators* (process VPs one by one), and *grouping operators* (group primitives into subsets according to some objectives or conditions). We use strongly-typed GP (cf. [8]), which implies that two operators may be connected to each other only if their input/output types match. The complete list of operators and specification of GP types may be found in [5, 16].

Given an input image s, an individual-learner L builds a hierarchy of VP sets derived from it. Each application of selector, iterator, or grouping operator creates and returns a new set of VPs that includes other elements of the hierarchy. In the end, individual's root node returns a nested VP hierarchy built atop of P, which reflects the processing performed by L for s. However, fitness calculation is based on a side-effect of that computation, i.e., on the similarity of s and the canvas c resulting from DAs performed during tree processing. Technically, we implement DA as an extra GP operator called *Draw*, which draws sections connecting each pair of VPs from the set of VPs given as its argument. As to fitness calculation, we assume 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, computed by a greedy heuristics. When the distance d between pixels is less than 5, the cost is 0; if $d > 15$, the cost is maximal and equals 1, and for $5 \leq d \leq 15$ the cost is a linear function of d. 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. 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.

4 Knowledge Reuse

When applying our approach to a k-class classification problem, we run in parallel k independent evolutionary processes for n generations. Each evolutionary process uses representation of individuals and fitness function described in Section 3 and is devoted to one decision class. The best representatives obtained from particular runs form a complete multi-class classifier (recognition system), which is then ready to classify new examples using a straightforward voting procedure detailed in Section 5. This basic approach will be in the following denoted shortly as 'GP'.

In the basic approach, each class may be learnt in isolation, making possible processing of multiple classes in parallel. Thus, adding a new class to the already learnt problem is straightforward and does not require re-training of the already evolved recognizers. On the other hand, given the similar nature of particular elementary learning tasks, we expect them to share some domain knowledge. Thus, it seems natural to hypothesize that enabling some knowledge reuse between the elementary learning processes would be

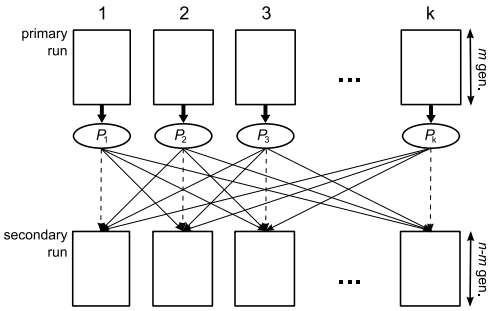

Figure 1. The architecture of GPKR and GPKR$^+$.

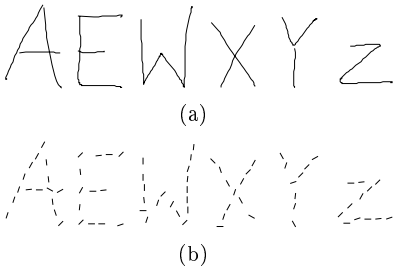

Figure 2. (a) selected training examples and (b) visual primitives derived from them.

profitable for convergence of evolutionary processes and/or for the performance of the resulting recognition system.

To verify this possibility, we come up with the following architecture for cross-class knowledge reuse, denoted by 'GPKR' in the following. For the initial m generations ($m < n$), evolutionary runs (called hereafter *primary runs*) proceed exactly in the same way as in GP. 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 the *secondary run*.

In the secondary run, we activate an extra *crossbreeding operator*, which is intended to import some genetic material from the pools P_i. This operator works similarly to the crossover operator, but it interbreeds individuals from the current population of the i^{th} secondary run ('natives') with the individuals from one of the pools P_j $j \neq i$ ('aliens'). First, it selects a native parent from the current population using the same selection procedure as crossover. Next, it selects an alien parent by randomly choosing one of the pools P_j, $j \neq i$, and then randomly selecting an individual from P_j, where this choice is *not* influenced by individual's fitness. Then, two nodes N_n and N_a are randomly selected in the native and alien parent, respectively, and the subtree rooted in N_n in the native parent is replaced by the subtree rooted in N_a. The modified native parent is treated as offspring and injected into the subsequent population.

We have also considered a slightly different version of GPKR, named GPKR$^+$. In this variant of our approach, we drop the constraint $j \neq i$, i.e., we allow for crossbreeding of natives with individuals from the primary run of the same decision class.

Figure 1 shows the architecture of the GPKR approach. The only difference between GPKR$^+$ and GPKR is reflected by the presence of an extra dashed arrow connecting the primary and secondary runs of the same class. Both GP and GPKR use the same parameter settings and require k evolutionary runs lasting n generations each. Thus, if we ignore the negligible time needed for population re-initialization, the time complexity of GPKR is the same as that of GP on the average. Note also, that, as the pools P_is are fixed, the runs devoted to particular classes do not have to work literally in parallel, but may be carried out sequentially.

5 The Experiment

The purpose of the experiment is to compare GPKR, the method with knowledge reuse, to the basic method (GP) on a real-world task of handwritten character recognition. Using a TabletPC computer we prepared a training set containing 72 images (examples, objects) of six ($k = 6$) upper-case characters: A, E, W, X, Y, and Z, each character class represented by 12 examples written by 3 persons. Figure 2 illustrates selected training examples and the primitives obtained from them. Each short segment depicts a single VP, with its spatial coordinates located in the middle of the segment and the orientation depicted by slant.

We use generational evolutionary algorithm maintaining a population of 25,000 GP individuals for $n = 400$ generations. The initial population is created using Koza's ramped half-and-half operator with ramp from 2 to 6 [8]. We apply tournament selection of size 5, using individuals' sizes (number of tree nodes) for tie breaking and thus promoting smaller GP trees and alleviating the problem of code bloat. For GP 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 GPKR runs, crossover probability stays the same, while mutation probability is lowered to 0.17 to yield 0.03 to the crossbreeding operator (see Section 4). The GP tree depth limit is set to 10. 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 for both GP and GPKR runs, we split the population into 10 islands and exchange some individuals between them every 20^{th} generation starting from the 50^{th} generation. In that exchange, each odd-numbered island donates 10% of its well-performing individuals (selected by tournament of size 5) to five even-numbered islands, where the donated individuals replace the 'worst' 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 described here is implemented within *each* evolutionary process independently.

Synthesis of each recognition system involves running $k = 6$ evolutionary processes, each of them using training examples from one character class for fitness computation, and producing one best-of-run individual. The ensemble of all 6 best-of-run individuals constitute the complete recognition system, which undergoes evaluation on the test set of characters, which is a separate collection of $6 \times 68 = 408$ characters and includes also characters written by *other* people than for the training set. A recognition system classifies an example t by computing fitnesses (responses) of all six individuals for t and indicating the class associated with the fittest individual. This entire procedure of evolutionary training and testing is repeated 33 times to obtain statistically conclusive results. In each of 33 trials, we evolve one GP recognition system (control experiment without knowledge reuse), and four independent GPKR-m recognition systems that vary in the generation number m where transition between primary and secondary run takes place $m \in \{50, 100, 200, 300\}$. We also evolved GPKR$^+$-$m$ recognition systems, but the results of GPKR turned out to be slightly better, thus we skip GPKR$^+$-m in the following.

In Fig. 3, we show the fitness graphs of the best-of-generation individuals averaged

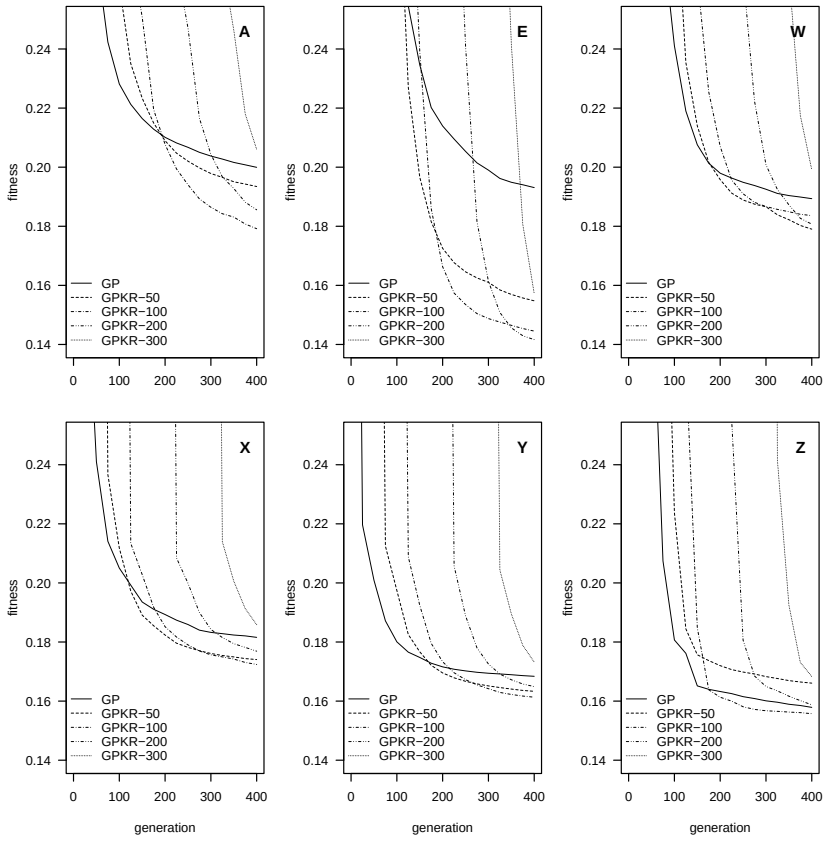

Figure 3. Mean fitness graphs.

over all 33 evolutionary runs. For GPKR, only secondary runs are shown, as the primary run of GPKR is equivalent to GP stopped after m generations. Despite the reduced number of available generations, the secondary runs of GPKR converge to solutions that are not worse than GP, quite independently of the primary run length m, though GPKR-100 and GPKR-200 seem to perform best. For characters A, E, and W, GPKR ends up with significantly better fitness values; for task E, the improvement amounts to approximately a quarter of the GP fitness. On the other hand, the lack of difference between GP and GPKR for character class Z, together with the relatively quick GP's convergence suggest that GPKR cannot help much if the task is easy enough to be solved without knowledge reuse.

Table 1 shows the *test-set* fitness values of the evolved best-of-run individuals averaged over 33 evolutionary runs. Values printed in bold indicate GPKR's superiority to GP with respect to the paired two-sided t-Student test at 0.05 significance level. Again, statistically, GPKR never performs worse than GP; quite on the contrary, it is superior to GP in 9 out of 24 cases, which clearly suggests that GPKR prevents overfitting.

Table 2 presents the *average* true positive (TP) and false positive (FP) of complete recognition systems on the test set. The results confirm earlier observations: for instance,

Table 1. Test set fitness of the best-of-run individuals averaged over 33 evolutionary runs.

Class	GP	GPKR-50	GPKR-100	GPKR-200	GPKR-300
A	0.415	0.450	0.394	**0.360**	0.389
E	0.462	**0.369**	**0.328**	**0.325**	**0.318**
W	0.322	0.297	0.318	0.299	0.304
X	0.444	**0.360**	**0.354**	**0.329**	**0.328**
Y	0.381	0.390	0.386	0.392	0.375
Z	0.283	0.302	0.271	0.281	0.285

Table 2. Average TP rates and FP rates of the recognition systems on the test set.

Class	TP rate					FP rate				
	GP	GPKR				GP	GPKR			
		50	100	200	300		50	100	200	300
A	0.952	0.943	0.934	0.957	0.936	0.003	0.003	0.003	0.002	0.001
E	0.910	0.931	0.959	0.965	0.968	0.017	0.019	0.017	0.011	0.011
W	0.994	0.999	0.990	0.997	0.999	0.011	0.012	0.014	0.012	0.015
X	0.815	0.918	0.915	0.934	0.926	0.015	0.022	0.025	0.027	0.027
Y	0.894	0.853	0.847	0.830	0.848	0.032	0.012	0.015	0.013	0.014
Z	0.953	0.914	0.939	0.947	0.939	0.018	0.020	0.010	0.010	0.010

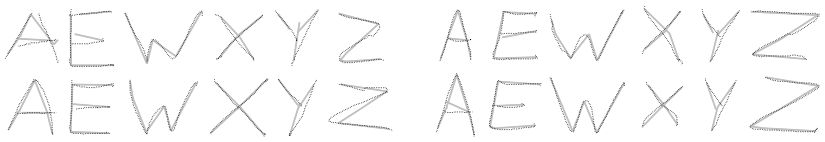

Figure 4. Visualization of the process of shape restoration.

in terms of TP, GPKR-200 outperforms GP on decision classes A, E, W, and X. In terms of FP, the comparison outcome varies depending on decision class, but in general *all* GPKR systems perform better than GP, attaining accuracy of classification from 92.65% (GPKR-50) to 93.82% (GPKR-200), versus 91.96% for GP.

In Fig. 4, we illustrate the process of generative shape restoration performed by the well-performing individuals for selected character classes. Thin dotted lines mark the shapes drawn by a human, whereas thick continuous lines depict drawing actions performed by the individual (sections). In most cases, the evolved individuals successfully reproduce the overall shape of the recognized object, despite various forms of imperfectness of the hand-drawn characters.

6 Conclusions

We demonstrated the possibility of obtaining substantial performance increments by introducing knowledge reuse in a variant of genetic programing designed to process visual information and recognize objects. The proposed mechanism of knowledge reuse is straightforward and may be implemented by a relatively simple extension of the canonical scheme of evolutionary algorithm and introduction of an off-shelf GP crossover operator for crossbreeding. The method does not increase the computational effort of the learning process, and provides statistically significant performance improvement at the same computational expense as the basic method.

Bibliography

[1] Rich Caruana. Multitask learning. *Mach. Learn.*, 28(1):41–75, 1997.

[2] Edgar Galvan Lopez, Riccardo Poli, and Carlos A. Coello Coello. Reusing code in genetic programming. In Maarten Keijzer *et al.*, editor, *Genetic Programming 7th European Conference, Proceedings*, volume 3003 of *LNCS*, pages 359–368, 2004.

[3] Daniel Howard, Simon C. Roberts, and Conor Ryan. Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. *Pattern Recognition Letters*, 27(11):1275–1288, 2006.

[4] William H. Hsu, Scott J. Harmon, Edwin Rodriguez, and Christopher Zhong. Empirical comparison of incremental reuse strategies in genetic programming for keep-away soccer. In Maarten Keijzer, editor, *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, 26 July 2004.

[5] Wojciech Jaśkowski. Genetic programming with cross-task knowledge sharing for learning of visual concepts. Master's thesis, Poznan University of Technology, Poznań, Poland, 2006.

[6] John R. Koza, Forrest H Bennett III, David Andre, and Martin A Keane. Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming. In Tetsuya Higuchi *et al*, editor, *Proceedings of International Conference on Evolvable Systems: From Biology to Hardware (ICES-96)*, volume 1259 of *LNCS*, Tsukuba, Japan, 1996.

[7] J.R. Koza. *Genetic Programming*. MIT Press, Cambridge, MA, 1992.

[8] J.R. Koza. *Genetic programming – 2*. MIT Press, Cambridge, MA, 1994.

[9] Krzysztof Krawiec and Bir Bhanu. Visual learning by coevolutionary feature synthesis. *IEEE Trans. on System, Man, and Cybernetics – Part B*, 35(3):409–425, 2005.

[10] SJ Louis and J. McDonnell. Learning with case-injected genetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 8(4):316–328, 2004.

[11] Sushil J. Louis. Genetic learning from experience. In *Proceedings of the International Congress on Evolutionary Computation*, Canberra, Australia, 2003. IEEE Press.

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

[13] T. M. Mitchell. The discipline of machine learning. Technical Report CMU-ML-06-108, Machine Learning Department, Carnegie Mellon University, July 2006.

[14] M. Rizki, M. Zmuda, and L. Tamburino. Evolving pattern recognition systems. *IEEE Transactions on Evolutionary Computation*, 6:594–609, 2002.

[15] Simon C. Roberts, Daniel Howard, and J.R. Koza. Evolving modules in genetic programming by subtree encapsulation. In Julian F. Miller *et al.*, editor, *Genetic Programming, Proceedings of EuroGP'01*, volume 2038 of *LNCS*, pages 160–175, 2001.

[16] Bartosz Wieloch. Genetic programming with knowledge modularization for learning of visual concepts. Master's thesis, Poznan University of Technology, Poznań, Poland, 2006.