Knowledge Reuse in Genetic Programming applied to Visual Learning

Wojciech Jaśkowski
wjaskowski@cs.put.poznan.pl

Krzysztof Krawiec
kkrawiec@cs.put.poznan.pl

Bartosz Wieloch
bwieloch@cs.put.poznan.pl

Institute of Computing Science, Poznan University of Technology, Piotrowo 2, 60965 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 the 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 following extended with a knowledge reuse mechanism that allows a learner to import genetic material from hypotheses that evolved for the 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, more importantly, significantly reduces the risk of overfitting.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Genetic Programming, Machine Learning, Pattern Recognition, Knowledge Reuse

1. INTRODUCTION

Despite several decades of intense research, machine learning (ML) is still a limited imitation of basic human skills. One of the reasons for that deficiency is the lack of ability to identify and reuse knowledge fragments across different learning tasks. Standard ML algorithms do not accumulate knowledge when faced with consecutive learning tasks, and work with fixed learning biases. Though such approaches perform well in many scenarios, as demonstrated by successful ML applications in many areas, an ability to detect and reuse the universal background knowledge (a.k.a. commonsense knowledge), or more specific domain-related knowledge, or even more specific task-related knowledge, would be definitely a virtue for a learning system. In particular, such an ability would, among others, improve the convergence speed of the learning process, reduce the risk of overfitting, and keep down the number of training examples required to learn the concept. Some of these profits resulting from knowledge reuse have been already demonstrated in related studies on, e.g., multitask learning [1]. However, apart from scarce studies, the last decade did not see a breakthrough in this topic, and knowledge reuse has been recently listed by Tom Mitchell [2] among the most important and challenging issues in ML.

There are several reasons for the inability of most of 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, or even domain-specific knowledge (often identified with *inductive bias* in inductive learning). Secondly, knowledge encoded in representations that are predominantly used in machine learning is 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 a similar/analogous attribute in another learning task.

Inspired by these limitations, in this paper we exploit the paradigm of genetic programming (GP, [3]) 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. This property, together with a built-in mechanisms of propagation of evolutionary material through crossover, enable advanced and effective knowledge manipulation. Thus, the major contribution of this paper is a novel method of knowledge reuse in GP that operates between a group of learners that work in parallel. In particular, 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.

The remaining part of this paper starts with a short review of related work (Section 2). Next, in Section 3 we detail the proposed approach of generative object recognition based on GP individuals that process visual primitives. The following

Section 4 is the core part of this paper and presents the proposed method of knowledge reuse within the GP paradigm. Section 5 describes an extensive computational experiment concerning handwritten character recognition that demonstrates the positive impact of knowledge reuse on the recognition accuracy. Final Section 6 draws general conclusions from this study.

2. RELATED WORK

Though knowledge reuse is definitely an important issue in computational intelligence, it attracted so far relatively modest 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 context of GP, knowledge reuse is often connected with knowledge encapsulation [4, 5, 6, 7], which is however not used in the approach presented here. Among reported research, the work done by Louis *et al.* resembles our contribution the most [8, 9, 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, works in a strictly sequential way, and does not involve GP. These (and also some other) features make it different from our approach.

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 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, using a machine learning classifier to learn and reason from some predefined and fixed image features computed from the input image 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 [11, 12, 13, 14, 15]. In [16, 14] we proposed a methodology that evolved feature extraction procedures encoded either as genetic programming or linear genetic programming individuals. The idea of GP-based processing of attributed visual primitives was explored for the first time in [17], and was further developed in [18, 19, 20].

The approach presented in this paper may be considered as a variant of generative pattern recognition. In a typical paper on that topic [21], Revow *et al.* used a predefined set of deformable models encoded as B-splines and an elastic matching algorithm based on expectation maximization for the task of handwritten character recognition. In [22], an analogous approach has been proved useful for recognition of hand-drawn shapes. However, the approach presented here goes significantly further, as it does not require *a priori* database of object models. And, last but not least, the recognition (restoration) algorithm has to restore the input image using *multiple* drawing actions, which implies the ability to decompose the analyzed shape into elementary components.

3. GENERATIVE VISUAL LEARNING

The proposed approach may be shortly characterized as *generative visual learning*, as our evolving learners aim at reproducing the input image and are rewarded according to the quality of that reproduction. The reproduction is partial, i.e., the learner restores only a particular *aspect* of the image contents. In this paper, the aspect of interest is shape, whereas other factors, like color, texture, shading, are ignored.

The reproduction takes place on a virtual canvas spanned over the input image. On that canvas, the learner (GP individual) is allowed to perform some elementary *drawing actions* (DAs for short). To enable successful reproduction, DAs should be compatible with the image aspect that is to be reconstructed. In this paper, we consider hand-drawn polygons and, to enable the learner to restore their shape, we make our DAs draw sections.

As an example, let us consider reconstruction of an empty triangular shape. It requires from the learner performing the following steps: (i) detection of conspicuous features — triangle corners, (ii) pairing of the detected triangle corners, and (iii) performing DAs that connect the paired corners. However, within the proposed approach, the learner is not given *a priori* information about the concept of the corner nor about the expected number of them. We expect the learner to discover these on its own.

To reduce the amount of data that has to be processed and to bias the learning towards the image aspect of interest, our 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 following by P.

The learning algorithm itself does not make any assumptions about the particular salient feature used for VP creation. Reasonable instances of VPs include, but are not limited to, edge fragments, regions, texems, or blobs. However, the type of detected feature determines the image aspect that is reconstructed. 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.

In the preprocessing phase that transforms an input image s into its VP representation P, *candidate VPs* are extracted from s based on local image magnitude (brightness). Then, the obtained candidate locations are sorted with respect to decreasing magnitude, and at most 80% of most prominent of them are included into the primitive representation. Also, to filter out the less prominent candidates and reduce the final number of VPs, a lower limit d_{min} on the mutual proximity of VPs is imposed. The VP candidates are processed sequentially with respect to decreasing magnitude, and a new primitive p may be added to P only if there is no other primitive already in P closer than d_{min}, i.e., there is no $p' \in P$ such that $\|p, p'\| < d_{min}$.

The resulting image VP representation P is usually several orders of magnitude more compact than the original image s. On the other hand, the essential sketch of the input

image s is well preserved. Figure 4 shows the VP representations derived from objects from Fig. 3. Each short segment depicts a single VP, with its (x,y) coordinates located in the middle of the segment and the orientation depicted by slant.

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 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 *ImageNodes*) 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. A particular tree node may (i) group primitives, (ii) perform selection of primitives using constraints imposed on VP attributes or their other properties, or (iii) add new attributes to primitives.

We use strongly-typed GP (cf. [23]), which implies that two operators may be connected to each other only if their input/output types match. The following types are used: numerical scalars ($\Re$ for short), sets of VPs (Ω, potentially nested), attribute labels (A), binary arithmetic relations (R), and aggregators (G).

The full list of GP operators may be found in [19, 20]. The non-terminal GP operators may be divided into following categories:

1) Scalar operators (as in standard GP applied to symbolic regression; see [23]). Scalar operators accept arguments of type $\Re$ and return result of type $\Re$.

2) Selectors. The role of a selector is to filter out some of the VPs it receives from its child node(s) according to some objectives or condition. Selectors accept at least one argument of type Ω and return 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, operator *LessThan* applied to child nodes (P, p_o, 0.3) filters out all VPs from P for which the value of the attribute p_o (orientation) is less than 0.3.

3) Iterators. The role of an iterator is to process, one by one, the VPs it receives from one of its children. For instance, operator *ForEach* iterates over all the 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 VP and returned.

4) Grouping operators. The role of those operators is to group primitives into a certain number of sets according to some objectives or conditions. For instance, *GroupHierarchyCount* uses agglomerative hierarchical clustering, where euclidean distance of primitives serves as the distance metric.

5) Attribute constructors. An attribute constructor defines and assigns a new attribute to the VP it processes. The definition of a new attribute, which must be based on the values of existing VP attributes, is given by the GP code contained in the right child subtree. To compute the value of a new attribute, attribute constructor passes the VP (operator *AddAttribute*) or the sub-primitives of the VP (operator *AddAtributeToEach*) through that subtree. Attribute constructors accept one argument of type Ω and one of type $\Re$, and return a result of type Ω.

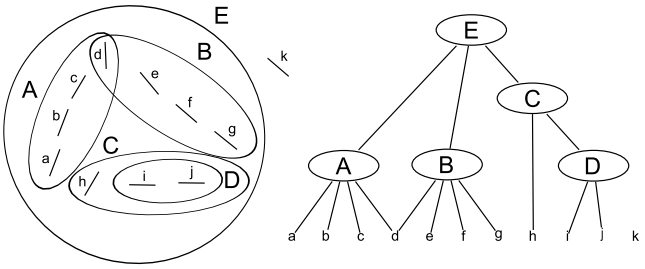

Figure 1: The primitive hierarchy built by the learner from VPs, imposed on the image (left) and shown in an abstract form (right); VP attributes not shown for clarity.

Given the elementary operators, an individual-learner L applied to an input image s builds gradually a hierarchy of VP sets derived from s. Each application of selector, iterator, or grouping operator 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. Some of the elements of the hierarchy may be tagged by new attributes created by attribute constructors.

Figure 1 illustrates an example of VP hierarchy built by the learner in response to input image/stimulus s. In the left part of the figure, the short edge fragments labeled by single lower-case letters represent the original VPs derived from the input image s, which together build up P. In the right part, the VP hierarchy is shown in an abstract way, without referring to the actual placement of particular visual primitives 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.

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 operator 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.

The drawing created on the canvas c by the learner L for an input image s is then evaluated to provide feedback for L and enable its potential improvement. This evaluation consists in comparing the contents of c to s. For this purpose, a simple and efficient approach was designed. In general, this approach 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 in the following way. 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, a 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. The ideal learner perfectly restores shapes in all training images and its fitness amounts to 0. The more the canvas c produced by a learner differs from s, the greater its fitness value.

4. INTRODUCING KNOWLEDGE REUSE

In terms of ML, the procedure described in Section 3 performs *one-class learning* [24], 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). When applying our approach to a k-class classification problem, we run in parallel k independent evolutionary processes for m 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 following denoted shortly as 'GP'.

Such multiple application of one-class learning has many advantages. Firstly, each class may be learnt in isolation, making possible processing of multiple classes in parallel. Secondly, extension to incremental learning becomes trivial, as 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. Running the elementary tasks in isolation may be partially redundant, as some basic functionalities may be common for many (or all) decision classes. For instance, locating the lower end of the shape of letter Y presented in an image may require similar subtree of GP operators as locating the lower ends of letter X. Thus, it seems natural to hypothesize that enabling some knowledge sharing between the elementary learning processes would be 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 that extends our approach by cross-class knowledge reuse, denoted by 'GPKR' in following. For the initial n generations ($n < m$), evolutionary runs (called hereafter *primary runs*) proceed exactly in the same way as in the basic version of the approach. As the run devoted to i^{th} decision class ($i = 1...k$) reaches the n^{th} generation, we store its population in a *pool* P_i, so that P_i constitutes a snapshot of i^{th} evolutionary run at n^{th} generation. Next, the current population of the run is re-initialized (in the same way as the initial population of the primary run), and the evolution continues for the remaining $m - n$ generations, referred to as *secondary run*.

The secondary run proceeds with almost the same evolutionary settings as the primary one, with the only exception of activating an extra *crossbreeding operator*, which is intended to import some genetic material from the pools created in the primary run. Technically, for a certain task i, this operator works similarly to the crossover operator used in both primary and secondary runs, but it interbreeds individuals from the current population ('natives') with the individuals from one of the pools P_j $j \neq i$ ('aliens'). First,

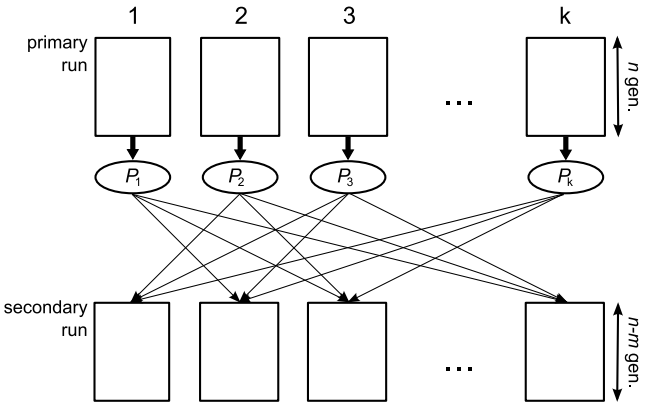

Figure 2: The architecture of GPKR.

it selects a native parent from the current population using the same selection procedure as crossover (see experimental part for details). Then, 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 tree rooted in N_a. The modified native parent is treated as offspring and injected into the subsequent population (provided it meets the constraints optionally imposed on GP trees).

Figure 2 shows the architecture of the GPKR approach. Boxes represent primary and secondary evolutionary runs, and arrows depict the transfer of genetic material between them. Each i^{th} column containing sequence of primary and secondary run is responsible for learning the i^{th} decision class.

It should be emphasized that GPKR does not involve knowledge encapsulation. No building blocks are explicitly defined in the genetic material gathered in pools. Virtually any code fragment evolved in a primary run may be injected into an individual evolving in one of the secondary runs. The crossbreeding operation may involve large portions of code as well as small code fragments, single terminal nodes in extreme case.

Both GP and GPKR use the same parameter settings (except for crossbreeding) and require k evolutionary runs lasting m generations each. The number of fitness function calls (the effort) is therefore the same. Thus, if we ignore the negligible needed for population re-initialization, the time complexity of GPKR is the same as that of GP on the average (though the actual evolution time may vary due to variability of fitness computation time, which in turn depends on tree size). This eases the comparison of both methods in the following experimental part. 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

5.1 The Setup and Parameters

The purpose of the experiment is to compare GPKR, the method with knowledge reuse, to the basic method (GP)

Figure 3: Selected training examples.

Figure 4: Visual primitives derived from examples in Fig. 3.

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 ($k = 6$) uppercase characters: A, E, W, X, Y, and Z, each character class represented by 12 examples. The letters were written by three persons, and placed at random locations on a raster image of 640×480 pixels. Figure 3 illustrates selected training examples, and Fig. 4 shows the primitives obtained from them. Each segment depicts a single VP, with its spatial coordinates located in the middle of the segment and the orientation depicted by slant.

The runs of the GP method serve as control experiment. Technically, we use generative evolutionary algorithm maintaining a population of 25,000 GP individuals for $m = 400$ generations. The initial population is created using Koza's ramped half-and-half operator with ramp from 2 to 6 [23]. 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 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; 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 [25]. For evolutionary parameters not mentioned here explicitly, 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 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. 2 – the island model described here is implemented within *each* evolutionary process independently.

The experiment was conducted according to the following procedure. Using the above settings, we evolved the entire recognition system thrice: once without knowledge reuse (GP, the control experiment), and twice for the approach with knowledge reuse (GPKR), as depicted in Fig. 2, for two different lengths n of the primary run, $n = 100$ (GPKR-100) and $n = 200$ (GPKR-200). In all three scenarios, this involves running $k = 6$ evolutionary processes, each of them using training examples from one character class as fitness cases for fitness computation. Each run produces one best-of-run individual. The ensemble of all 6 best-of-run individuals for particular character classes constitute the ultimate result of the learning process, i.e., the complete recognition system. Finally, the complete recognition system undergoes evaluation on the test set of characters, which is disjoint with the training set. This entire procedure of evolutionary training and testing is repeated 33 times to obtain statistically conclusive results.

Table 1: Fitness of the best-of-run individuals averaged over 33 evolutionary runs.

	GP	GPKR-100		GPKR-200	
Class	avg	avg	p value	avg	p value
A	0.200	**0.1792**	0.000	**0.1856**	0.002
E	0.193	**0.1445**	0.001	**0.1416**	0.000
W	0.189	0.1836	0.439	0.1807	0.081
X	0.182	**0.1724**	0.003	0.1769	0.118
Y	0.168	**0.1613**	0.001	0.1649	0.140
Z	0.158	0.1557	0.726	0.1587	0.913

5.2 The Results

In Fig. 5, we show the fitness graphs of the best-of-generation individuals averaged over all 33 evolutionary runs, plotted with 0.95 confidence intervals. Continuous lines plot results for GP, while dashed and dotted ones for GPKR-100 and GPKR-200, respectively. For GPKR, only secondary runs are shown, as the plots for the primary runs overlap with the results for GP (primary run of GPKR is equivalent to GP stopped after n generations).

The first general observation is that, despite the reduced number of available generations (300 for GPKR-100 and 200 for GPKR-200), the secondary runs of GPKR converge to solutions that are not worse than GP. Moreover, for character tasks A, E, and W, GPKR ends up with significantly better fitness values. This difference is especially prominent for the task E, where it amounts to approximately a quarter of the fitness value attained by GP. On the other hand, for character class Z, the virtual lack of difference in performance between GP and GPKR, together with the relatively quick GP's convergence to good solutions suggest that GPKR cannot help much if the task at hand may be successfully solved without knowledge reuse.

Table 1 shows the final results, i.e., fitness values of the evolved best-of-run individuals averaged over 33 evolutionary runs, for GP and for the two separate GPKR experiments that use different lengths of the primary run: GPKR-100 ($n = 100$) and GPKR-200 ($n = 200$). Table 2 shows analogous information for the testing set, which is a separate collection of 6×68 = 408 hand-drawn characters created analogously to the testing set. In should be emphasized that the testing set contains also characters written by different people than the training set. Values printed in bold indicate

Figure 5: Mean fitness graphs with 0.95 confidence intervals (whiskers shifted to improve legibility).

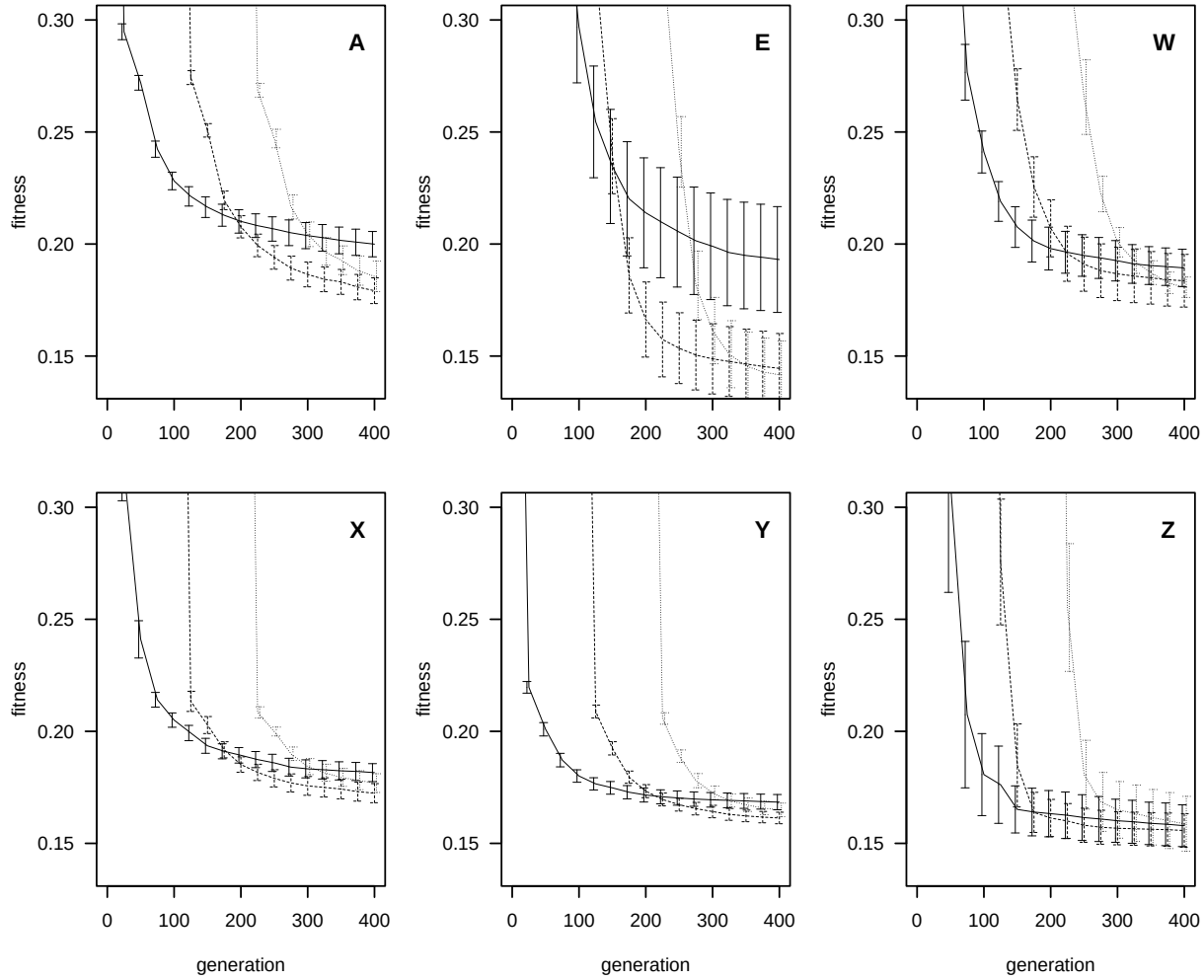

Table 2: Test set fitness of the best-of-run individuals averaged over 33 evolutionary runs.

	GP	GPKR-100		GPKR-200	
Class	avg	avg	p value	avg	p value
A	0.415	0.3939	0.302	**0.3599**	0.000
E	0.462	**0.3279**	0.000	**0.3250**	0.000
W	0.322	0.3183	0.848	0.2993	0.086
X	0.444	**0.3542**	0.000	**0.3291**	0.000
Y	0.381	0.3862	0.670	0.3915	0.324
Z	0.283	0.2710	0.616	0.2808	0.930

GPKR's superiority to GP with respect to the paired two-sided t-Student test at 0.05 significance level. Each fitness column is accompanied by an extra column containing the p-value of the corresponding t-Student test. Again, GPKR never performs worse than GP; quite on the contrary, it is superior to GP on the training set in six out of twelve cases, and in five out of twelve cases on the testing set. This last observation is especially appealing, as it clearly suggests that GPKR successfully prevents overfitting. Note also that, apart from these conclusive cases, there are a few other characters for which the p-values are reasonably small (e.g., character W for GPKR-200), suggesting possible positive impact of knowledge reuse.

The results shown so far refer to fitness values only; in following, we describe the performance in terms of machine learning. To this aim, for each set of GP or GPKR results, we combine the six best-of-run individuals into one recognition system. The system performs recognition of an example t by computing fitnesses (responses) of all six individuals for t and indicating the class associated with the fittest individual. Such procedure is motivated by an obvious observation, that a learner is taught to perform well on images from one class and its fitness should be near 0 only for images of this class. For example, it is unlikely that an individual that successfully learned to restore the shape of (recognize) character E could equally well restore the shape of letter W; for such negative examples, it will thus produce a high fitness value.

Table 3 presents the *average* confusion matrix of the complete, six-class recognition systems built from the best-of-run individuals evolved using the GP approach according to the procedure described at the end of section 5.1. To demonstrate the average expected behavior of an evolved

Table 3: Normalized test-set confusion matrix for the complete GP recognition system (rows: actual object classes, columns: system's decisions).

class	A	E	W	X	Y	Z
A	**95.23**	1.83	2.32	0.31	0.04	0.27
E	0.98	**91.00**	0.98	0.18	0.13	6.73
W	0.13	0.27	**99.38**	0.13	0.04	0.04
X	0.27	1.29	0.13	**81.51**	15.55	1.25
Y	0.04	1.34	2.18	6.51	**89.35**	0.58
Z	0.13	3.74	0.09	0.40	0.31	**95.32**

Table 4: Normalized test-set confusion matrix for the complete GPKR-100 recognition system (rows: actual object classes, columns: system's decisions).

class	A	E	W	X	Y	Z
A	**93.40**	1.69	4.28	0.18	0.04	0.40
E	0.98	**95.90**	0.22	0	0.13	2.76
W	0.04	0.22	**98.98**	0.22	0.53	0
X	0.31	0.71	0.13	**91.53**	6.55	0.76
Y	0.18	0.89	2.05	11.23	**84.71**	0.94
Z	0.18	4.81	0.09	0.80	0.22	**93.89**

recognition system, this matrix has been computed by averaging the 33 confusion matrices obtained from 33 independent recognition systems applied to the test set. Tables 4 and 5 show analogous results for GPKR-100 and GPKR-200, respectively. In all these tables, rows correspond to actual class assignments of an example, while columns correspond to the decisions of the recognition system. To improve readability, the matrices do not show absolute numbers of examples but are normalized row-wise; for instance, the value 1.83 at the intersection of row A and column E in Table 3 means that, on the average, 1.83% of examples of character A have been mistakenly classified as E by the GP recognition system.

It may be easily observed that all the evolved recognition systems perform well, exceeding 90% accuracy of classification on the average. For all three recognition systems, most of misclassifications occur between character classes X and Y, which may be due to the visual similarity of the shapes of these letters. However, both GPKR-100 and GPKR-200 perform better than GP, attaining classification accuracy of 93.07% and 93.82%, respectively, versus 91.96% for GP.

The presented confusion matrices reflect the expected performance of a recognition system that has been obtained at a relatively low computational expense. Given more evolutionary runs, these results may be further boosted by employing more voters per each decision class, as opposed to one voter per class in the above scheme. Such an approach is especially appealing in the context of evolutionary computation, as each evolutionary run usually produces a unique best-of-run individual, so their fusion may result in synergy. For instance, a compound recognition system composed of five voters per class (i.e., $5 \times 6 = 30$ voters in total) yields 98.77% accuracy of classification for GPKR-100, 98.28% for GPKR-200, whereas only 93.87% for GP.

In Fig. 6, 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

Table 5: Normalized test-set confusion matrix for the complete GPKR-200 recognition system (rows: actual object classes, columns: system's decisions).

class	A	E	W	X	Y	Z
A	**95.68**	0.18	3.57	0.04	0.04	0.49
E	0.45	**96.52**	0.18	0.04	0.22	2.58
W	0.04	0.04	**99.69**	0	0.13	0.09
X	0.31	0.27	0	**93.36**	5.44	0.62
Y	0.09	0.67	2.41	12.79	**83.02**	1.02
Z	0.04	4.37	0	0.49	0.45	**94.65**

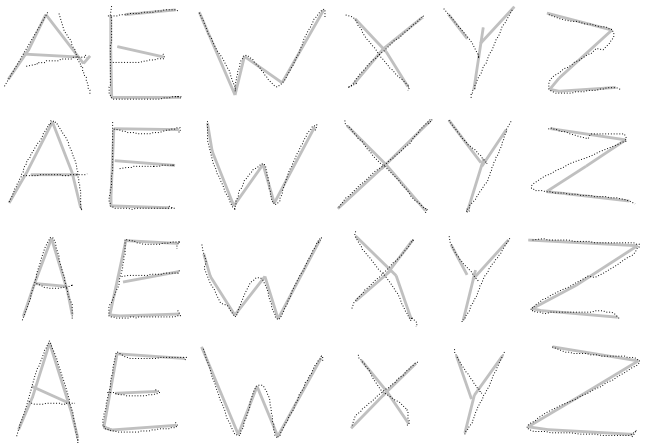

Figure 6: Visualization of the process of shape restoration.

drawing actions performed by the individual (sections). It may be easily observed that, in most cases, the evolved individuals successfully reproduce the overall shape of the recognized object. Reproduction seems to be robust 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 (GPKR) 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 (GP).

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 GPKR, for the reasons stated earlier in Introduction (relatively low probability of simultaneous 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.

The positive results obtained in this study suggest a successful match between the abstraction level of knowledge representation and the level at which GPKR operates. Supposedly, our operators that group and select visual primitives are well suited for identifying common learning subtasks (GP code fragments) and their reuse. In particular, we hypothesize that, among the reused code fragments, the most useful are those ones that help the evolving individuals to detect specific parts of recognized objects; these are most probably image features that are important for reconstruction and repeat across object classes: junctions and ends of pen strokes. However, this supposition needs more in-depth analysis of results and will be subject of another study.

7. REFERENCES

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

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

[3] Koza, J.: Genetic Programming. MIT Press, Cambridge, MA (1992)

[4] Koza, J.R., Bennett III, F.H., Andre, D., Keane, M.A.: Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming. In Higuchi, T., Masaya, I., Liu, W., eds.: Proceedings of International Conference on Evolvable Systems: From Biology to Hardware (ICES-96). Volume 1259 of Lecture Notes in Computer Science., Tsukuba, Japan, Springer-Verlag (7-8 October 1996)

[5] Roberts, S.C., Howard, D., Koza, J.R.: Evolving modules in genetic programming by subtree encapsulation. In Miller, J.F., Tomassini, M., Lanzi, P.L., Ryan, C., Tettamanzi, A.G.B., Langdon, W.B., eds.: Genetic Programming, Proceedings of EuroGP'2001. Volume 2038 of LNCS., Lake Como, Italy, Springer-Verlag (18-20 April 2001) 160–175

[6] Galvan Lopez, E., Poli, R., Coello Coello, C.A.: Reusing code in genetic programming. In Keijzer, M., O'Reilly, U.M., Lucas, S.M., Costa, E., Soule, T., eds.: Genetic Programming 7th European Conference, EuroGP 2004, Proceedings. Volume 3003 of LNCS., Coimbra, Portugal, Springer-Verlag (5-7 April 2004) 359–368

[7] Hsu, W.H., Harmon, S.J., Rodriguez, E., Zhong, C.: Empirical comparison of incremental reuse strategies in genetic programming for keep-away soccer. In Keijzer, M., ed.: Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference, Seattle, Washington, USA (26 July 2004)

[8] Louis, S.J.: Genetic learning from experience. In: Proceedings of the International Congress on Evolutionary Computation, Canberra, Australia, IEEE Press (2003)

[9] Louis, S.: Case injected genetic algorithms for learning across problems. Engineering Optimization **36** (apr 2004) 237–247(11)

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

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

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

[13] Maloof, M., Langley, P., Binford, T., Nevatia, R., Sage, S.: Improved rooftop detection in aerial images with machine learning. Machine Learning **53** (2003) 157–191

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

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

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

[17] Krawiec, K.: Learning high-level visual concepts using attributed primitives and genetic programming. In Rothlauf, F., ed.: EvoWorkshops 2006. LNCS 3907, Berlin Heidelberg, Springer-Verlag (2006) 515–519

[18] Krawiec, K.: Evolutionary learning of primitive-based visual concepts. In: Proc. IEEE Congress on Evolutionary Computation, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada July 16-21. (2006) 4451–4458

[19] Jaskowski, W.: Genetic programming with cross-task knowledge sharing for learning of visual concepts. Master's thesis, Poznan University of Technology, Poznań, Poland (2006)

[20] Wieloch, B.: Genetic programming with knowledge modularization for learning of visual concepts. Master's thesis, Poznan University of Technology, Poznań, Poland (2006)

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

[22] Krishnapuram, B., Bishop, C.M., Szummer, M.: 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), Washington, DC, USA, IEEE Computer Society (2004) 20–25

[23] Koza, J.: Genetic programming – 2. MIT Press, Cambridge, MA (1994)

[24] Moya, M. R., K.M.W., Hostetler, L.D.: One-class classifier networks for target recognition applications. In: Proceedings world congress on neural networks,

Portland, OR, International Neural Network Society (1993) 797–801
[25] Luke, S.: ECJ evolutionary computation system (2002) (http://cs.gmu.edu/ eclab/projects/ecj/).