Genetic Programming for Cross-Task Knowledge Sharing

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 consider multitask learning of visual concepts within genetic programming (GP) framework. The proposed method evolves a population of GP individuals, with each of them composed of several GP trees that process visual primitives derived from input images. The two main trees are delegated to solving two different visual tasks and are allowed to share knowledge with each other by calling the remaining GP trees (subfunctions) included in the same individual. The method is applied to the visual learning task of recognizing simple shapes, using generative approach based on visual primitives, introduced in [17]. We compare this approach to a reference method devoid of knowledge sharing, and conclude that in the worst case cross-task learning performs equally well, and in many cases it leads to significant performance improvements in one or both solved tasks.

Categories and Subject Descriptors: I.2.6 [Artificial Intelligence]: Learning—Concept learning

General Terms: Algorithms

Keywords: Genetic Programming, Representations, Knowledge Sharing, Multitask Learning

1. INTRODUCTION

In [22], Mitchell lists the "transfer of what is learned for one task to improve learning in other related tasks" among the most important current research issues in machine learning (ML). Multitask learning (MTL) may be considered as a special form of such transfer of knowledge. MTL is usually defined as an extension of standard single-task learning (STL), where the learner solves more than one learning task at a time. For instance, in case of artificial neural networks (ANN), MTL usually consists in using a layered network with multiple outputs, which are expected to serve different, however related, classification or regression tasks [2]. MTL is motivated mostly by expected improvement in generalization, reduced training time, intelligibility of the acquired knowledge [2], accelerated convergence of the learning process, and potential reduction of amount of training data needed to learn the concept(s) [4]. Several studies [24, 23, 2, 4] have found MTL effective with respect to some of these criteria when compared to STL.

To our knowledge, almost all MTL approaches based on learning-from-examples paradigm assume that the tasks to be learned share the same structure (data schema). In this study, we introduce a novel approach based on genetic programming (GP, [15]), which is free from this limitation and learns while sharing knowledge between two loosely related tasks specified by disjoint training sets. This category of multitask learning is in the following referred to as *cross-task learning* (XTL). By not requiring the tasks to share the training data, XTL has obviously wider applicability than MTL. Moreover, thanks to symbolic knowledge representation used in our approach (see Section 3), the details of knowledge sharing (e.g., level of abstraction, extent, etc.) are left to the decision of the learner and, thus, do not have to be specified *a priori*. This is advantageous in comparison to, e.g., MTL using ANNs, where knowledge sharing has to be, to some extent, pre-specified by the network architecture (the way the particular neurons are shared between the tasks).

Most of traditional ML tasks given in the attribute-value form cannot benefit from XTL, unless they share some attributes that have similar or identical interpretation. However, this looks different in computer vision (CV) that is subject of this paper, as most of visual learning tasks require some kind of common visual bias. For instance, off-line recognition of handwritten Latin characters requires similar knowledge to an analogous task for Kanji characters, as in both cases the recognition process focuses on shape analysis of black-and-white images composed of pen strokes.

In this paper we demonstrate an effective XTL method for visual learning that is build upon our former research on genetic programming applied to generative visual learning [17]. After detailing motivations and reviewing related work in Section 2, and presenting the base generative learning approach in Section 3, we describe our XTL architecture in Section 4. Then, in Section 5, we provide experimental evidence of XTL efficiency on a group of five loosely related visual learning tasks and analyze the obtained solutions from the viewpoint of knowledge sharing, also ruling out the alternate explanations for the observed phenomena. Section 6 summarizes the results and groups conclusions.

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

2. RELATED RESEARCH

Following [2] and [4], we may name several potential advantages of multitask learning: improved generalization, reduced training time, intelligibility of the acquired knowledge, accelerated convergence of the learning process, and reduction of number of examples required to learn the concept(s). The ability of MTL to fulfill some of these expectations has been demonstrated, mostly experimentally, in different ML scenarios, most of which used ANNs as the underlying learning paradigm [27, 24, 23].

In this paper, we use tree-like GP expressions for representing the knowledge acquired by learners, including the shared knowledge, implemented by additional GP trees. This makes our approach related to GP research on modularization through subtree encapsulation; to some extent, it may be considered as GP learning with Automatically Defined Functions (ADFs) shared between two co-learning tasks. In our approach, however, the shared subtrees are never fixed, but evolve though all the evolution process. Apart from the canonical ADFs, defined by Koza [15, section 6.5.4], more research on encapsulation [6, 5, 26] and code reuse [16, 3] has been done within the GP community. Proposed approaches include sharing of function-defining branches (partial results) between GP individuals in population [28], reuse of assemblies of parts within the same individual [7], identifying and re-using code fragments based on the frequency of occurrences in population [8], or explicit expert-driven task decomposition using layered learning [1, 10] for Robosoccer tasks. Nevertheless, no research on *parallel* cross-task learning is known to us, especially with knowledge sharing taking place internally within each individual.

The approach presented in this paper learns a computer vision task, which makes it related also to the domain of visual learning. Learning in computer vision, traditionally dominated by neural approaches, is currently receiving more and more attention from machine learning and from different paradigms of bio-inspired computing, including evolutionary computation [25, 21, 19, 9]. It should be however emphasized, that in most approaches reported in literature, visual learning is limited to parameter optimization that usually concerns only a particular image processing step, such as image segmentation or feature extraction. Methods that are able to produce a more or less complete recognition system, as does the approach presented here, are rather scarce.

The approach to evolve visual routines was first proposed by Johnson *et al.* [14]. In [19], we proposed a methodology that evolved feature extraction procedures encoded either as genetic programming or linear genetic programming individuals. The idea of generative GP-based processing of attributed visual primitives was originally proposed in [18] and further developed in [17, 11, 29, 13]. In particular, in [12] we examine the possibility of sequential knowledge sharing, as opposed to this contribution, where knowledge sharing takes place between learning processes that proceed in parallel.

3. GENERATIVE VISUAL LEARNING USING GENETIC PROGRAMMING

3.1 The Idea of Generative Visual Learning

The proposed approach may be shortly characterized as *generative visual learning*, as our evolving learners reproduce the input image and are rewarded according to the quality of that reproduction. That reproduction is partial, i.e., concerns 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 discarded.

The reproduction takes place on a virtual canvas spanned over the input image. On that canvas, the learner is allowed to perform some elementary *drawing actions* (DAs for short). To enable successful reproduction, we use DAs that are compatible with the image aspect that is to be reconstructed. As in this paper we recognize polygons, we implement DA as an insertion of a single section into the canvas.

As an example, let us consider the reconstruction of a triangle. It requires the learner performing at least 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 corner nor about the expected number of them – it is supposed to discover these on its own.

DAs result from processing carried out by the learner (GP tree) for the visual input it has been provided with. Technically, the coordinates of sections inserted by DAs into the canvas are derived from the salient features detected in the input image. To reduce the amount of data that has to be processed by the learner and to bias the learning towards the image aspect of interest, our approach abstracts from raster representation 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, denoted in the following by P, enables the learner to perform specific DAs.

The learning algorithm itself does not make any assumptions about the type of feature used for VP creation. Reasonable types 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 the subject of analysis. As in this paper we focus on shape, we use VPs representing prominent local brightness 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 orientation of local brightness gradient.

3.2 Embedding Generative Learning in GP Framework

The proposed method uses an evolutionary algorithm to maintain a population of generative visual learners outlined in Section 3.1. Technically, each visual learner L (individual, solution) is implemented as a genetic programming (GP) expression that is allowed to perform, among others, an arbitrary number of DAs. Each such procedure has the form of a tree, with nodes representing *elementary operators* that process sets of VPs. The terminal *Input* nodes fetch the set of VP primitives P derived from the input image, and the consecutive nodes process those sets of VPs, all the way up to the root node. A particular tree node may group primitives, perform selection of primitives using constraints imposed on VP attributes or other properties, add new attributes to primitives, or perform a DA. Individual's fitness depends on DAs it performs in response to visual primitives P derived from training images $s \in S$.

Table 1: The GP operators.

Type	Operator
$\Re$	ERC – Ephemeral Random Constant
Ω	*Input*() – the VP representation P of the input image s
A	p_x, p_y, p_o, and custom attributes added by *AddAttribute*
R	*Equals*, *Equals5Percent*, *Equals10Percent*, *Equals20Percent*, *LessThan*, *GreaterThan*
G	*Sum*, *Mean*, *Product*, *Median*, *Min*, *Max*, *Range*
$\Re$	+($\Re$,$\Re$), –($\Re$,$\Re$), *($\Re$,$\Re$), /($\Re$,$\Re$), *sin*($\Re$), *cos*($\Re$), *abs*($\Re$), *sqrt*($\Re$), *sgn*($\Re$), *ln*($\Re$), *AttributeValue*(Ω,$\Re$)
Ω	*SetIntersection*(Ω,Ω), *SetUnion*(Ω,Ω), *SetMinus*(Ω,Ω), *SetMinusSym*(Ω,Ω), *SelectorMax*(Ω,A), *SelectorMin*(Ω,A), *SelectorCompare*(Ω,A,R,$\Re$), *SelectorCompareAggreg*(Ω,A,R,G), *CreatePair*(Ω,Ω), *CreatePairD*(Ω,Ω), *ForEach*(Ω,Ω), *ForEachCreatePair*(Ω,Ω,Ω), *ForEachCreatePairD*(Ω,Ω,Ω), *AddAttribute*(Ω,$\Re$), *AddAttributeForEach*(Ω,$\Re$), *GroupHierarchyCount*(Ω,$\Re$), *GroupHierarchyDistance*(Ω, $\Re$), *GroupProximity*(Ω, $\Re$), *GroupOrientationMulti*(Ω, $\Re$), *Ungroup*(Ω), *Draw*(Ω)

Table 1 presents the complete list of GP operators. We use strongly-typed GP (cf. [15]), 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). Though the detailed explanation of particular GP operators is beyond the scope of this paper, in the following we try to sketch their overall characteristic, dividing them into the following categories:

1) Scalar operators (as in standard GP applied to symbolic regression). Scalar operators accept arguments of type $\Re$ and return a 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 a condition. Selectors accept at least one argument of type Ω and return a result of type Ω. *Non-parametric selectors* expect two child nodes of type Ω and produce an output of type Ω. Operators that implement basic set algebra, like set union, intersection, or difference, belong to this category. *Parametric selectors* expect three child nodes of types Ω, A, and $\Re$, respectively, and produce output of type Ω. For instance, operator *SelectorCompare* applied to child nodes (P, p_o, *LessThan*, 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 VPs from its left child and processes each of them using the GP code specified by its right child. Technically, the way a set of VPs P is processed by the right child is that its terminal *Input* nodes return P instead of fetching VPs from the input image as it happens normally. The VPs resulting from all iterations are grouped into one VP and returned.

4) Grouping operators. The role of these operators is to group primitives into a certain number of sets according to some objectives or a condition. 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 is given by the GP code contained in the right child subtree and is based on the values of existing VP attributes. To compute the value of a new attribute, attribute constructor passes a VP (operator *AddAttribute*) or the sub-primitives of a 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. The detailed definitions of all operators may be found in [11, 29].

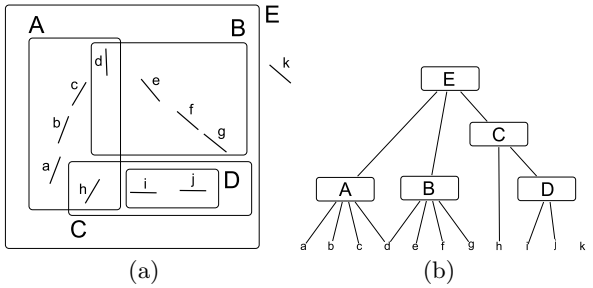


Figure 1: The primitive hierarchy built by the learner from VPs, imposed on the image (a) and shown in an abstract form (b); VP attributes were not shown for clarity.

Given these elementary operators, a learner L, when applied to an input image s, builds gradually a hierarchy of VPs 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 hierarchy of sets of VPs, built atop of P, which reflects the processing performed by learner 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 a 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 of Fig. 1, the VP hierarchy is shown in an abstract way, without referring to the actual placement of particular VPs in the input image. Note that the hierarchy does not have to contain all VPs from P, and that a particular VP from P may occur in multiple branches of the hierarchy.

To reconstruct the essential features of the input image s, the learner is allowed to perform drawing actions (DAs) that boil down to drawing sections on the output canvas. To implement that within the GP framework, an extra GP operator called *Draw* is included in the set of operators presented in Table 1. It expects as an argument one VP set T and

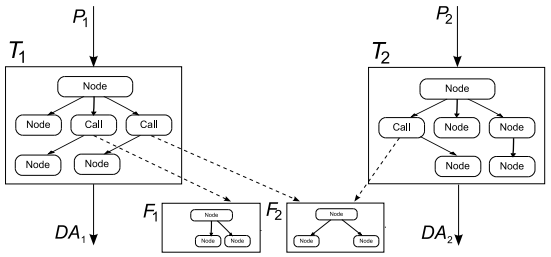


Figure 2: An exemplary XTL-individual that is able to process examples of two different tasks. See description in the text for details.

returns it unchanged, drawing on the canvas sections connecting each pair of VPs from T.

The obtained canvas is then evaluated with respect to its similarity to the input image s that gave rise to it, and the quantified similarity becomes individual's fitness. Detailed description of this evaluation process is presented in experimental part.

4. KNOWLEDGE SHARING ARCHITECTURE

The learner described in Section 3.2 consists of one tree and processes a single learning task. To extend our approach to cross-task learning (XTL), we define an XTL-*learner* or *XTL-individual* as an individual that consists of *several* GP-trees and is able to process examples from two different tasks. Although the processing flows are different in both trees, they are allowed to partially overlap and enabling so the knowledge sharing.

Our XTL architecture relies heavily on the GP tree representation and follows the paradigm of structural programming. A GP-tree can be treated as a function, which, when called for a certain set of VPs (passed to it by means of terminal *Input* nodes), returns a hierarchy of VPs at the root node. Each XTL-individual consists of two *main trees* (T_1 and T_2) and some number of *subtrees* (F_i) serving as *subfunctions* that can be called for certain arguments by main trees. For this purpose, a special node *Call*(Ω, $\mathbb{N}$) was introduced. *Call*(P, i) returns the result of subfunction F_i for the argument P.

When the control flow reaches a *Call* node, an appropriate subfunction tree is executed with arguments passed by the *Call* node. Technically, the subfunctions' *Input* nodes return P instead of fetching VPs from the input image as it happens for the main trees. Thus, a subfunction can be called many times by the main trees with different arguments. In particular, it can be executed in a loop (e.g., by a *ForEach* node).

An exemplary XTL-individual is shown in Fig. 2. It consists of four trees: T_1 and T_2 are the main trees; F_1 and F_2 are the shared trees – subfunctions. Solid arrows represent argument dependency whereas dashed ones show which subfunction is called by a certain *Call* node. P_1 and P_2 are the sets of VPs to process. DA_1 and DA_2 are the resulting drawing actions. Note that some argument dependencies were removed from the diagram for the clarity; *Node* represents any GP operator.

It is important to point out that *Call* nodes *may,* but not necessarily *have to*, be used by an individual in the process of evolution. Knowledge sharing occurs only if a subfunction is called by *both* main trees. The process of evolution may optionally take advantage of it.

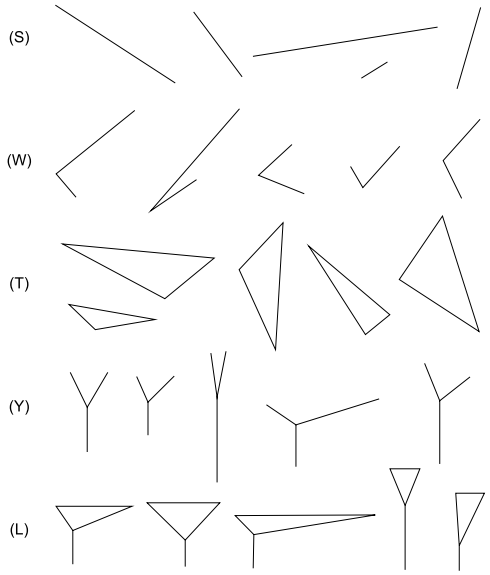


Figure 3: The examples of shapes of data sets from five different tasks: (S) sections; (W) wedges; (T) triangles; (Y) letters Y; (L) leaned triangles.

Obviously, the XTL architecture described here can be generalized to $n > 2$ tasks and to a more sophisticated call hierarchy. In this paper, we limit our discussion to two tasks, two subfunctions, and a single-level call hierarchy, i.e., subfunctions are not allowed to call subfunctions.

5. THE EXPERIMENT

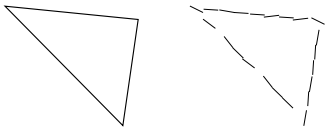
The objective of the experiment is to demonstrate the performance of XTL on a set of different visual learning tasks and to verify the hypothesized superiority of evolved XTL-solutions to solutions evolved in a control experiment without knowledge sharing.

5.1 The Task, Data, and Settings

The goal of the five considered learning tasks was to acquire the concepts of the following shapes: sections (S), wedges (W), triangles (T), Y-letters (Y), and leaned triangles (L). The data set of each task was composed of 10 gray-scale raster (640x480) images consisting of shapes of different dimensions, positions, and orientations.

The task to acquire the concept of a certain basic shape seems simple for humans, but, in fact, it is not so straightforward. The learners have only bare VPs at their disposal and no *a priori* information about, e.g., their collinear alignment or spacial proximity. The most obvious way to reconstruct the original images is to select VPs corresponding to polygon vertices, however it is not so easy, because the vertices are not marked in any way. There is also no obvious method to determine the junctions in such shapes as letter Y or leaned triangle.

Selected examples of each data set are shown in Fig. 3. It is important to note that although all the shapes in Fig. 3 are presented together on one canvas, each input image (training example) presented to the learner contains one shape only.



(a) (b)

Figure 4: A simple geometric figure (a) and a corresponding VP representation (b).

In the preprocessing phase that transforms an input image s into its VP representation P, *candidate VPs* are extracted from s based on local brightness gradients. Then, the obtained candidate locations are sorted with respect to decreasing brightness gradient, and at most 80% of the 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 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 4b shows the VP representation P derived from the object from Fig. 4a. 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.

Let s' be the canvas (see end of Section 3.2) an individual L produces in response to the visual primitives P derived from the training image s. The fitness of L is based on the difference between s and s' and takes into account two factors: true positives (amount of correctly restored edges) and false positives (amount of excessively drawn segments). However, these two objectives are clearly conflicting and it is hard to weight them explicitly. Therefore, we decided to drive the search of solution space using two-objective fitness based on the difference between an input image s and response (canvas) s' produced by individual L:

$$f(s, s') = \left(f^+(s, s'), f^-(s, s')\right). \tag{1}$$

The measures of true positives f^+ and false positives f^- are defined as follows:

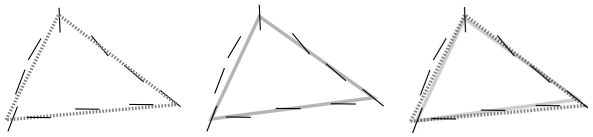
$$f^+(s, s') = \frac{\sum_{s(x,y)>0} \min\left(s(x, y), s'(x, y)\right)}{\sum_{x,y} s(x, y)}, \tag{2}$$

$$f^-(s, s') = 1 - \frac{\sum_{x,y} \max(0, s'(x, y) - s(x, y))}{M_b M_p - \sum_{x,y} s(x, y)}, \tag{3}$$

where $s(x, y)$ is the value of pixel (x, y) in image s; M_b is the maximum possible brightness; and M_p is the number of pixels in the image. In our experiment $M_b = 255$ and $M_p = 640 \times 480$.

It is easy to notice that the theoretically worst possible individual (the anti-ideal, producing a negative of the original image) has fitness (0,0) and the ideal, reconstructing the original image perfectly, has fitness (1,1).

The fitness of an individual L for a *set* of images S averages $f(s, s')$ over all $s \in S$:



(a) (b) (c)

Figure 5: (a) The original image s (the gray dashed triangle) and its VP representation (black sections). (b) The optimal output image s^* drawn manually. (c) The superposition of the ideal output image s' (equivalent to the original image s) and s^*.

$$f(S) = \left(f^+(S), f^-(S)\right) = \left(\frac{1}{|S|}\sum_{s\in S} f^+(s, s'), \frac{1}{|S|}\sum_{s\in S} f^-(s, s')\right).$$

For individuals implementing the XTL architecture where two data sets S_1 and S_2 are considered at the same time, the overall fitness of an individual L is defined as

$$f(S_1, S_2) = \left(\frac{1}{2}\left(f^+(S_1) + f^+(S_2)\right), \frac{1}{2}\left(f^-(S_1) + f^-(S_2)\right)\right).$$

Thanks to the two-objective fitness, there is no need for introducing any arbitrarily chosen weight in order to decide what is worse: misplaced or missing sections in the individual's output. The second advantage of the two-objective approach is that it helps to diversify the population, as non-dominated solutions include both the individuals with low amount of drawing (having usually high f^-) and the individuals that draw too much (having usually high f^+).

On the other hand, to perform verification on the test set, we need a *single* best individual of an evolutionary run (best-of-run individual). For this purpose, we come up with a single-objective method for selecting the best individual from a set of non-dominated solutions obtained at the end of evolution. Such selection could be done by comparing output images s' of the non-dominated individuals with ideal's output, i.e., with s, as the ideal individual by definition restores the input image s perfectly. This would make sense if the input set of VPs contained all information required for perfect restoration of s. In practice, this assumption is not satisfied due to unavoidable loss of information during VP extraction from the input image s. Therefore, instead of referring to the ideal drawing s produced by a hypothetical ideal individual that may not exist, we rely here on the *optimal output image* s^*. s^* is prepared manually in a way that makes its perfect reproduction possible (given the loss of information that takes place in the preprocessing). Technically, s^* is the simplest accurate restoration of s that can be drawn using sections that bridge exclusively the VPs extracted from s – see Fig. 5 for illustration. Given s^*, we define an *error* $e(s)$ of an individual L on s by measuring the pixel-wise distance of its output image s' from s^*:

$$e(s) = \sum_{x,y} \left|s'(x, y) - s^*(x, y)\right| / \sum_{x,y} s^*(x, y). \tag{4}$$

The error of an individual L for the data set S averages $e(s)$ over $s \in S$: $e(S) = e(s)/|S|$. Obviously, the greater $e(S)$, the worse individual L. In particular, $e(S) = 0$ implies

Table 2: Comparison of results of XTL and control experiments.

	S_1-S_2	L-W	L-Y	L-S	L-T	W-Y	W-S	W-T	S-Y	S-T	T-Y
XTL	S_1	**.830**±.16	**.757**±.21	**.736**±.21	**.684**±.13	**.719**±.20	**.653**±.16	**.758**±.26	**.141**±.12	.180±.14	**.662**±.36
	S_2	**.680**±.22	**.810**±.15	.194±.20	**.541**±.25	**.811**±.21	.178±.16	.788±.23	**.787**±.14	**.677**±.21	.905±.23
control	S_1	.834±.14	.834±.14	.834±.14	.834±.14	.807±.16	.807±.16	.807±.16	.173±.10	.173±.10	.723±.26
	S_2	.807±.16	.824±.18	.173±.10	.723±.26	.824±.18	.173±.10	.723±.26	.824±.18	.723±.26	.824±.18
t-test	S_1	=	=	=	<	=	<	=	=	=	=
(α=0.1)	S_2	<	=	=	<	=	=	=	=	=	=
t-test	S_1	=	<	<	<	<	<	=	=	=	=
(α=0.2)	S_2	<	=	=	<	=	=	=	=	=	=

that L produces optimal output images for all $s \in S$, i.e., $s' = s^*$.

The *best individual* for a certain data set S is the individual from the last generation of an evolutionary run that has the lowest error $e(S)$. In the XTL experiments, where each XTL-individual is evaluated on two data sets S_1 and S_2 corresponding to two different tasks, the final error of an individual L is averaged over both tasks:

$$e(S_1, S_2) = (e(S_1) + e(S_2))/2 \quad (5)$$

For each run, we use a generational evolutionary algorithm with 5000 individuals evolving for 500 generations. Since the fitness of individuals is two-objective, we perform selection based on Pareto-ranking: after the evaluation phase, the individuals are ranked using the dominance relation from the best (rank $r = 1$) to the worst. The selection operator randomly selects an individual from rank r with the probability of $\lambda/2^{r-1}$, where λ is set to 0.5. The subsequent generations are created by crossing-over, mutating or copying the selected individuals; the probability of crossing-over is 0.9; the probability of mutation – 0.1. The maximum depth of GP tree is 6. The cross-over and mutation operations can be repeated up to 5 times if the resulting individuals do not meet this constraint; after 5 such failures, the parent individuals are passed as the result of selection. Except for the fitness function implemented for efficiency in C++, the algorithm is written in Java with the help of ECJ package [20]. For evolutionary parameters not mentioned here explicitly, ECJ's defaults have been used.

The results of some preliminary experiments have shown that sometimes good (first-rank) solutions are forgotten in the process of evolution, i.e., they disappear in successive generations. Due to this fact, elitism was introduced. In each generation, from each set of first-rank individuals that have equal fitness,[1] one individual is drawn at random and it is unconditionally copied to the subsequent generation.

For the considered five shape classes, all 10 possible combinations of two tasks (*task pairs*) were examined: L-W, L-Y, L-S, L-T, W-Y, W-S, W-T, S-Y, S-T, and T-Y. In order to verify the impact of our XTL implementation, five appropriate control experiments were designed and carried out (S, W, T, Y, L). The control experiments were designed fairly to give them the same chance as the XTL experiments. The control individuals have analogous structure and may use at least the same number of nodes as the XTL individuals. More precisely, each individual in a control experiment consists of one main tree and 2 subfunctions. Thus, there is no knowledge sharing in control experiments.

As one XTL experiment corresponds to a pair of control experiments, $10+5 = 15$ experiments were performed. Each of them was repeated 9 times for different random number generator seeds. In total, 270 evolutionary runs lasted ca. 56 hours on five PCs, each with Pentium 4 3.0 GHz.

5.2 The Results

For each experiment, the best-of-run individual was tested on a testing set containing 100 images. Table 2 presents the average errors $e(S_1)$ and $e(S_2)$ for appropriate test sets S_1 and S_2 for the best-of-run individuals in each experiment. For example, .178±.16 in column W-S and row XTL S_2 is the average error for the test set S_2 (sections in this case) for best-of-run XTL-individuals evolved in the experiment with wedges-sections task (W-S). In this and all following tables, the numbers following the '±' sign are standard deviations.

Bold font in Table 2 marks the experiments where XTL was superior to control experiments in absolute numbers, without considering statistical significance. The two bottom rows of the table show the results of two-sample pooled Student's t-tests for averages. '=' denotes inconclusive result; '<' means that the hypothesis stating that XTL-individual's error is lower than the error made by the control-individual is true at the significance level α. XTL-individuals outperformed control-individuals in 15 out of 20 tasks (the bold ones). Statistically, at the significance level 0.1 this statement is true for 3 out of 10 task pairs. This trend is even more visible when the significance level α is weakened to 0.2. Then, in 6 out of 10 task pairs the XTL-individuals turn out to be superior to control individuals.

It is important to point out that in neither case control-individuals are significantly better than XTL-individuals. Also, although only in one case (L-T) the improvement is significant for *both* tasks (L and T), generally, an improvement for one task is enough to consider XTL useful. These results in favor of XTL have been obtained despite the fact that the control experiment had more chance of evolving more sophisticated and potentially more powerful GP individuals. With two evolutionary runs using separate subfunctions, it had a chance to use 50% more code ($3 + 3 = 6$ trees vs. 4 trees in XTL-individuals).

5.3 Detailed Analysis of Knowledge Sharing

In order to perform in-depth investigation of these encouraging results, in this section we focus on W-S (wedges-sections) task pair and XTL architecture. For this task pair,

[1]It is very common that in a generation numerous (e.g., 50) individuals have exactly the same fitness, because the same output image may be produced by different individuals.

Table 3: Results for the extended W-S experiment.

	XTL	control	t-test (α=0.05)
wedges	**.722**±.18	.823±.21	<
sections	**.192**±.17	.210±.17	=

XTL was found better on one of two tasks. Similar results were obtained for 4 other task pairs (L-W, L-Y, L-S, W-Y), thus the W-S task pair may be claimed a typical one.

For statistical analysis, the experiment for W-S with the same settings as specified in Section 5.1 was carried out for 45 different random number generator seeds. The results, presented in Table 3, confirm the observations from Table 2. Superiority of XTL for wedges is very strong as the t-test probability is 0.0073, which allows us to conclude that this task is the beneficiary when sharing knowledge with the section task. From the statistical viewpoint, the result for the section task is the same as in the control experiment, thus we can conclude that there is no need to trade off between the two tasks.

As calling subfunctions by main trees is optional, the question arises whether the code of subfunctions is really utilized. Two kinds of analysis were carried out in order to answer this question. Firstly, the trees of best individuals were analyzed syntactically in order to find out whether the subfunctions are called at all. It turned out that, when using XTL, 66.7% and 68.9% of individuals that evolved for the wedges and sections task respectively, had at least one *Call* node in the main tree. For the control experiment, these figures were 57.8% and 73.3%, respectively. Thus, in majority of cases, the subfunctions were called by the main trees in both the XTL and control experiments. Since the main trees are not *forced* to use subfunctions in our approach, this result suggests that their use is generally profitable. Let us also notice that the average usages of subfunctions are comparable for XTL and control experiments. Thus, the alternative explanation of XTL superiority, claiming that XTL-individuals perform better as they use more code, may be rejected.

Secondly, we performed also a semantic analysis in order to check if subfunctions are semantically relevant, i.e., if subfunction code influences the processing of the individual. To examine this, two experiments were carried out. In the first one, each subfunction in the best individual was substituted with a single *Empty* node that returns an empty set of VPs, no matter what its argument is. In the second experiment, subfunctions were substituted with a single *Identity* node that simply returns its argument (does no processing). Then, the errors on the testing set of the original individuals and the modified ones were compared in terms of the error function $e()$. Any difference in the value of $e()$ was interpreted as the indicator of influence of the subfunction code on the processing taking place in the main trees. Otherwise, the subfunction code was considered trivial or dead.

The experiment with *Empty* nodes showed that the subfunctions were semantically relevant in *all* cases. For the experiment with *Identity* nodes, the subfunctions were semantically relevant in 98% of cases.

In our approach, the main trees are not *forced* to use subfunctions and the actual knowledge sharing occurs only if both main trees of an individual call the same subfunction at least once. Thus, our XTL-individuals are not forced to implement any knowledge sharing. It seems, however, that in more than half of the evolutionary runs the learning process found sharing profitable. Precisely, in 53% of cases XTL-individuals actually implement knowledge sharing (i.e., both main trees call at least one and the same subfunction). If knowledge sharing was not profitable, the process of evolution would get rid of it.

Table 4: Comparison of XTL-individuals that use and do not use knowledge sharing (KS).

	%	Task W	Task S
use KS	53%	**0.693**±0.18	**0.146**±0.12
don't use KS	47%	0.755±0.18	0.245±0.20
t-test ($\alpha = 0.05$)		=	<

We also compared how the performance of XTL-individuals that use knowledge sharing (53%) differs from the performance of XTL-individuals that do not use it (47%). The results are presented in Table 4. Statistically, individuals that use knowledge sharing are better on the sections task and equally good on the wedges task.

A very interesting result can be obtained when considering only XTL-individuals whose main trees share *both* subfunctions. An average error for such individuals is 0.563±0.14 and 0.088±0.03 for wedges and sections, respectively. It is much better than for the XTL-individuals that use main trees that share *at least one* subfunction (see Table 4). Unfortunately, only 4 out of 45 computed best-of-run individuals belong to this category and statistical analysis would be doubtful. It seems that such individuals evolve rarely, but as soon as they appear, they outperform any others.

XTL is profitable also in terms of computational effort, as it takes less computing time than the control approach (13.8 min for each XTL evolutionary run vs. 17.4 min for two corresponding control evolutionary runs).

6. CONCLUSIONS

We presented a method for evolving individuals which learn simultaneously from two loosely related visual tasks and are allowed to share knowledge needed for that purpose. For many pairs of learning tasks, the XTL-enabled learners are superior to XTL-disabled learners in terms of test-set performance. On the other hand, for pairs of tasks for which knowledge sharing does not seem to be useful, the possibility of using XTL does not seem to affect negatively the learners' performance. As the detailed analysis ruled out the alternative explanations of XTL superiority, the discovery and usage of common background knowledge must be the factor that accounts for this result.

7. REFERENCES

[1] A. Bajurnow and V. Ciesielski. Layered learning for evolving goal scoring behavior in soccer players. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1828–1835, Portland, Oregon, 20-23 June 2004. IEEE Press.

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

[3] 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.

[4] J. Ghosn and Y. Bengio. Bias learning, knowledge sharing. *ijcnn*, 01:1009, 2000.

[5] T. D. Haynes. *Collective Adaptation: The Sharing of Building Blocks*. PhD thesis, Department of Mathematical and Computer Sciences, University of Tulsa, Tulsa, OK, USA, Apr. 1998.

[6] N. Hondo, H. Iba, and Y. Kakazu. Sharing and refinement for reusable subroutines of genetic programming. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, volume 1, pages 565–570, Nagoya, Japan, 20-22 May 1996.

[7] G. S. Hornby and J. B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artif. Life*, 8(3):223–246, 2002.

[8] D. Howard. Modularization by multi-run frequency driven subtree encapsulation. In R. L. Riolo and B. Worzel, editors, *Genetic Programming Theory and Practise*, chapter 10, pages 155–172. Kluwer, 2003.

[9] 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.

[10] 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.

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

[12] W. Jaśkowski, K. Krawiec, and B. Wieloch. Knowledge reuse in genetic programming applied to visual learning. In *Genetic and Evolutionary Computation Conference GECCO*, 2007.

[13] 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.

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

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

[16] 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.

[17] K. Krawiec. 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*, pages 4451–4458, 2006.

[18] 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.

[19] 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.

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

[21] 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.

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

[23] J. O'Sullivan and S. Thrun. A robot that improves its ability to learn, 1995.

[24] L. Y. Pratt, J. Mostow, and C. A. Kamm. Direct Transfer of Learned Information among Neural Networks. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 584–589. AAAI, July 1991.

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

[26] 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.

[27] T. Sejnowski and C. Rosenberg. Nettalk: A parallel network that learns to read aloud. Technical Report JHU/EECS-86/01, The John Hopkins University, Baltimore, MD, 1986.

[28] E. E. Vallejo and F. Ramos. Result-sharing: A framework for cooperation in genetic programming. In W. B. *et al.*, editor, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, page 1238. Morgan Kaufmann, 1999.

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