Active Learning with Near Misses

Nela Gurevich and **Shaul Markovitch** and **Ehud Rivlin**
Computer Science Department, Technion - Israel Institute of Technology, 32000, Haifa, Israel
{nelka,shaulm,ehudr}@cs.technion.ac.il

Abstract

Assume that we are trying to build a visual recognizer for a particular class of objects—chairs, for example—using existing induction methods. Assume the assistance of a human teacher who can label an image of an object as a positive or a negative example. As positive examples, we can obviously use images of real chairs. It is not clear, however, what types of objects we should use as negative examples. This is an example of a common problem where the concept we are trying to learn represents a small fraction of a large universe of instances. In this work we suggest learning with the help of *near misses*—negative examples that differ from the learned concept in only a small number of significant points, and we propose a framework for automatic generation of such examples. We show that generating near misses in the feature space is problematic in some domains, and propose a methodology for generating examples directly in the instance space using *modification operators*—functions over the instance space that produce new instances by slightly modifying existing ones. The generated instances are evaluated by mapping them into the feature space and measuring their utility using known active learning techniques. We apply the proposed framework to the task of learning visual concepts from range images.

Introduction

Assume that we are trying to build a visual recognizer for a particular class of objects—chairs, for example—using existing induction methods. Assume the assistance of a human teacher who can label an image of an object as a positive or a negative example. As positive examples, we can obviously use images of real chairs. It is not clear, however, what type of objects we should use as negative examples. Are images of a monkey, a galaxy and a pen good candidates for this purpose?

This is an example of a common problem where the concept that we are trying to learn represents a small fraction of a large universe of instances. Arbitrary negative examples will differ considerably from positive examples, which might allow the learner unwanted flexibility in determining the classification boundary—flexibility that could yield a poor classifier.

Winston (1975), in his seminal work, recognized this difficulty and proposed a method for constructing efficient training sequences by using *near misses*—negative examples that differ from the learned concept in only a small number of significant points. Such examples do not necessarily belong to known concepts. Winston constructed specialized objects, such as an arch with no gap between the poles, to obtain negative examples that are near the classification boundary. The applicability of Winston's approach is restricted in that the near misses are constructed manually. Manual construction is a costly process and requires a deep understanding of the specific concept to be learned. It does not, therefore, provide a generic solution for the problem described above.

In this work, we build on the ideas of Winston, and propose a framework for automatic generation of near misses. We observe that a learning process involves two spaces—the instance space and the feature space. The teacher labels examples in the instance space; the labeling is independent of any particular set of features. The input of the induction algorithm are labeled members of the feature space. Since the classification boundaries are defined with respect to the feature space, it would be natural to generate near misses as feature vectors. Such an approach, however, would require mapping the generated vectors back to the instance space to be labeled by the teacher. This would be impossible in many domains where the feature functions are not one-to-one. For example, consider the space of 2D grayscale images. The possible features of an image may be related to corner information, gray-level histogram, etc. Given an image, it is easy to evaluate its features. However, there may be vectors in the feature space that represent more than one image, or represent no image at all.

Our methodology solves this problem by generating examples in the instance space but evaluating their merit in the feature space. We assume the availability of an initial set of positive examples in the instance space and a set of *modification operators*—functions over the instance space that produce new instances by slightly modifying existing ones. We apply short sequences of *modification operators* to the initial examples in order to generate new instances. The idea is that a slight modification of a positive instance will produce a new instance belonging to the same class, or will produce a near miss. We evaluate the generated instances by mapping them into the feature space and measuring their utility using known active learning techniques. The best instances are then labeled by the teacher, converted to labeled feature vectors, and given to the induction algorithm.

We apply the proposed framework to the task of learning visual concepts from range images. We show that defining meaningful modification operators over the instance space of range images is problematic, because such images contain only low-level information. We solve the problem by using an intermediate instance space—the *functional representation space*, where objects are represented in a higher level language. The efficiency of the proposed framework for object recognition is demonstrated by testing it on real-world recognition tasks.

The contributions of this work are threefold. First, we analyze the problem of example generation and identify the difficulty arising from the need to operate in two spaces—the instance space and the feature space. Second, we present a general framework for automatic generation of near misses. Third, we develop a methodology for example generation for visual recognition by introducing an intermediate (functional) space.

Learning with Near Misses

Let X be a set of instances. Let $t : X \rightarrow \{0,1\}$ be a *teacher* who labels instances in X. Let $\{f_1,\ldots,f_n\}$ be a set of feature functions over instances in X, where $f_i : X \rightarrow d_{f_i}$, and d_{f_i} is the domain of feature f_i. The *feature space* is defined as: $X_f = d_{f_1}\times,\ldots,\times d_{f_n}$. Let $v_i = \langle f_1(x_i),\ldots,f_n(x_i)\rangle$ represent the feature vector of an instance $x_i \in X$. A *learning algorithm* L takes as input a training set $\{\langle v_i, t(x_i)\rangle \mid x_i \in X, v_i \in X_f, i = 1,\ldots,m\}$ and returns a classifier $h : X_f \rightarrow \{0,1\}$.

Many works in machine learning make no distinction between the instance space and the feature space. This distinction, however, is important for our framework. In particular, the above definitions mean that the label of an instance is independent of the particular features chosen to represent it. Nevertheless, we assume that the induction algorithm processes feature vectors.

We consider two modes of learning—active and passive. Given a pool of unlabeled instances, *pool-based active learning* (Cohn, Atlas, & Ladner 1994) can be described as an iterative procedure. At each iteration, an unlabeled instance is chosen from the pool to be labeled by the teacher. The feature vector of the labeled instance is then added to the training set, and the learner induces a new hypothesis. The informativeness of an example can be evaluated by means of several methods, developed for this purpose, which take the already labeled examples into account. Note that all existing methods evaluate examples in the feature space. In the passive setup, the selection of examples to be labeled is uninformed, and can therefore be assumed to be random.

Automatic Example Generation

Our framework includes an algorithm that uses modification operators to generate near misses. A modification operator is a function $m : X \rightarrow X$, where X is the instance space. Thus, the only requirement for modification operators is to produce valid instances. Recall, however, that the motivation behind these operators is to generate near misses by slightly modifying existing examples. Therefore, when defining a new set of such operators, one should attempt to include operators that are able to cross the classification boundary.

Procedure ANMG (D_{init}, M)
  While $|X_{gen}| < n$
    $x_{new} \leftarrow$ Random object from D_{init}
    Repeat
      $m \leftarrow$ Random operator from M
      $x_{new} \leftarrow m(x_{new})$
      $stop = \begin{cases} true & \text{with probability } p_{stop} \\ false & \text{with probability 1 - } p_{stop} \end{cases}$
    Until $stop = true$
    $X_{gen} \leftarrow X_{gen} \cup \{x_{new}\}$
  Return X_{gen}

Figure 1: The ANMG algorithm

The ANMG (Automatic Near Misses Generation) algorithm implements the core of the proposed framework. It receives D_{init}, a set of positive instances in X, and $M = \{m_1,\ldots,m_n\}, m_i : X \rightarrow X$, a set of modification operators, and produces a pool of unlabeled examples X_{gen}.

To get a variety of new instances, our algorithm applies random sequences of operators to random initial positive instances. To control the length of the sequences, our algorithm uses a parameter p_{stop}. The algorithm selects a random initial instance and iteratively modifies it. At each iteration, it stops the modification with probability p_{stop} or continues with probability $1 - p_{stop}$. Therefore, the length of the sequence of modification operators applied to an initial example is a random variable with geometric distribution, defined by the parameter $1 - p_{stop}$. Figure 1 presents the pseudocode of the ANMG algorithm.

Active Learning with Near Misses

The ANMG algorithm generates a pool that is likely to include useful examples. In the passive learning setup, the pool is sampled randomly. In the active learning setup, we attempt to select potentially useful examples first.

To evaluate examples, we use an extension of the utility-based evaluation method presented by Lindenbaum, Markovitch, & Rusakov (2004). The method performs a lookahead to measure the examples' expected effect on the classifier induced by the learner. This effect is estimated by considering all possible example labels and their probabilities. The strength of the method is that it considers both the confidence of the current classifier in each possible label of an example, and the effect of adding the example with each of its possible labels to the training set of the learner (which will result in a new classifier). Specifically, in this work we use the *one-step lookahead* evaluation function, adjusted to the specific task of learning with near misses.

The one-step lookahead evaluation function is defined as:

$$U(x) = \sum_{l=0,1} P(t(x) = l|D) \cdot U_L(D \cup \{\langle v_x, l\rangle\}, D), \quad (1)$$

where $x \in X$, $v_x \in X_f$ is the feature vector of x, $P(t(x) = l|D)$ denotes the probability of an example x to be labeled l given the D, and $U_L(D \cup \{\langle v_x, l\rangle\}, D)$ is the utility, with respect to learner L, of adding $\langle v_x, l\rangle$ to training set D. We denote $D' = D \cup \{\langle v_x, l\rangle\}$.

The function $U_L(D', D)$ computes the difference between the hypotheses generated by learner L, when its training set is changed from D to D'. This difference is measured over the unlabeled examples in X_{gen}:

$$U_L(D', D) = \frac{|\{\, x \mid x \in X_{gen}, L(D)(v_x) \neq L(D')(v_x)\}|}{|X_{gen}|}, \quad (2)$$

where $L(D)$ is the hypothesis produced by the learner given a training set D.

The other component of Equation 1 that needs to be estimated is $P(t(x) = l|D)$. These probabilities are estimated using a *random field* based method. Let x_1 be the labeled example closest to x, x_2 the labeled example second closest to x, and $dist(v_{x_i}, v_{x_j})$ the normalized Euclidean distance function. Let $d_{01} = dist(v_x, v_{x_1}), d_{02} = dist(v_x, v_{x_2})$ and $d_{12} = dist(v_{x_1}, v_{x_2})$. The probabilities for the possible example labels are calculated as follows:

$$\delta = \begin{cases} \frac{\gamma(d_{01})+\gamma(d_{02})}{\frac{1}{2}+2\gamma(d_{12})} & t(x_1) = t(x_2) \\ \frac{\gamma(d_{01})-\gamma(d_{02})}{\frac{1}{2}-2\gamma(d_{12})} & \text{otherwise} \end{cases}, \quad (3)$$

$$P(t(x) = 1|t(x_1), t(x_2)) = \begin{cases} \frac{1}{2} + \delta & t(x_1) = 1 \\ \frac{1}{2} - \delta & \text{otherwise} \end{cases} \quad (4)$$

where $\gamma(d)$ is an approximation of a covariance function:

$$\gamma(d) = \frac{1}{4} e^{\frac{-d}{\sigma}} \quad (5)$$

and σ is calculated on the basis of the average distance between the examples in X_{gen}, scaled by parameter $\mathfrak{D}$:

$$\sigma = \frac{1}{\mathfrak{D}} \cdot \frac{1}{|X_{gen}|^2} \sum_{x_i \in X_{gen}} \sum_{x_j \in X_{gen}} dist(v_{x_i}, v_{x_j}). \quad (6)$$

As can be seen in Equation 4, the probabilities are biased towards 0.5, and the deviation from 0.5 is determined by the distance of the examined example from its neighbors. The scale parameter $\mathfrak{D}$ determines to which extent these distances affect the probability calculations.

In our framework, we would like to minimize the effect of the labeled examples on the probability estimations at the beginning of the learning process, when the training set of the learner is small and contains mostly positive examples. Later, when more examples are learned, we would like to increase the effect of the labeled examples on the estimations. We therefore amend the estimation process by dynamically updating $\mathfrak{D}$ using a decay formula:

$$\mathfrak{D} = 4 + \mathfrak{D}_0 e^{-\lambda t}, \quad (7)$$

where 4 is the value found by Lindenbaum et. al. to be best for $\mathfrak{D}$, and t is the index of the learning iteration. Choosing suitable values for $\mathfrak{D}_0$ and λ will allow us to manipulate the influence of the labeled examples on the estimated probabilities when necessary.

Object Recognition with Near Misses

The research described in this paper has been motivated by the problem of object recognition from range images. Assume that we are trying to learn a visual recognizer using a set of range images and the help of a teacher as described in the introduction. It looks as if the framework described above suits this task very well. The concepts are only a small fraction of the instance space. Therefore, using near misses of a concept as negative examples seems a logical solution. In addition, the mapping between the image space and the feature space is not one-to-one. Thus, the creation of examples directly in the instance space, as we suggest, is important.

Working with object images raises, however, a serious difficulty: the common image-based modification operators, such as a pixel change or overall scaling, usually change aspects of the image that are not relevant to the recognition task. To be able to define more meaningful modification operators in this domain, we need to represent instances in higher level language. We propose using an intermediate space—the *functional representation space*, where an object is described in terms of the functionality that it implements.

The main idea of the functional approach is that, in some cases, it is hard to describe all possible shapes of objects in a particular object class, but it is easy to describe a set of functional properties that they all share (Winston *et al.* 1983). For example, a chair has infinitely many possible shapes, but in functional terms the chair should only provide "sittability." The main assumption of the functional approach is that the object's shape is sufficient to deduce the existence of its functional properties. In this work, we adopt the ideas of Rivlin et al. (Rivlin, Dickinson, & Rosenfeld 1995; Froimovich, Rivlin, & Shimshoni 2002), who assume that the primary function of an object can be decomposed into lower-level functions. For example, the "sittability" of a chair is composed of a surface suitable for sitting (seat), a stable ground support, and possibly a back support. Thus, each object is composed of *functional parts*, each of which maps to one low-level function. A functional part is realized by a set of *primitive parts*—shape primitives of 3 basic types: sticks, plates and blobs. A stick is a part where one dimension in 3D is considerably larger than the other two, a plate is a part where two dimensions are considerably larger than the third, and a blob is a part where no dimension is considerably different than the other. Note that the mapping between functional parts and sets of primitive parts is not necessarily one-to-one: a single functional part may be realized by several different shape configurations. Nevertheless, all realizations conform to the same functional properties defined for the functional part.

In order to acquire the functional representation of a real object, it is scanned by a laser scanner, the resulting range image is segmented into regions, and the regions are classified into shape primitives. The shape primitives are then partitioned into functional parts. The process of mapping a range image to its functional representation can be automated as was shown by Froimovich, Rivlin, & Shimshoni (2002).

Since objects in the functional representation space are described by functional parts that consist of primitive parts, we can generate new examples by modifying the primitive or the functional parts of an object. An example of a possible modification operator in the functional representation space can be a function that, given two functional parts and a value d, changes the relative orientation of the two functional parts by d degrees. Other examples are primitive part scaling and primitive part moving.

Since we need to map objects in the functional representation space to feature vectors, we must define a set of features over such objects. We use a set of feature functions, called *geometric attributes* (Pechuk, Soldea, & Rivlin 2005), which analyze the geometric properties of an object's functional parts. A geometric feature can be of four types: (1) A feature of a single functional part, e.g., its volume, (2) A feature of a functional part relative to the whole object, e.g., the relation of the volume of the functional part to the volume of the object, (3) A feature of a relation between two different functional parts, e.g., the distance between the centers of their masses, and (4) A feature of the whole object, e.g., its stability.

To summarize this section, we outline how the information flows in the learning system. First, we create an initial set of positive examples by scanning real objects, segmenting the resulting range images into regions, and partitioning the regions into functional parts. Next, the ANMG algorithm generates a pool of new objects in the functional space using the modification operators and the initial examples. The new objects are mapped to the feature space, and evaluated in that space using the lookahead evaluation function. The object that is evaluated as the most informative is then mapped into the image space using a graphic viewer and presented to the expert for labeling[1]. The selected example features are retrieved and the example is then added as a labeled feature vector to the training set of the learner. Figure 2 illustrates this process.

Empirical Evaluation

Three classes of objects were used for the experiments: *stool, chair* and *fork.* 200 positive examples of each class were created by laser scanning of real objects, then segmenting and partitioning the results. In each experiment, a set of 10 positive examples was used as the initial labeled set. During the learning process, the training set was augmented with the generated examples that were presented to the expert for labeling.

For each class, two types of test sets were used. One type contained positive examples of the class that was being learned, and examples of the other known classes, labeled as negative. We will refer to such a test set as a *real-world test set.* Another test set type contained positive examples of the class being learned, as well as negative examples, which were near misses of that class. The negative examples were generated by applying random sequences of modification operators to random positive examples, and selecting the modified objects that represented negative examples. For each class, such a test set contained 300 examples—150 positive and 150 negative. The number of operators in each modification sequence—10 at most—was randomly chosen. Such test sets were designed to check that, while learning a concept with the help of near misses, the positive and near miss examples can be easily distinguished. Note that the near misses generated by the experimenter for the test sets were different from the ones generated by the learner for training. We will refer to such a test set as a *near miss test set.*

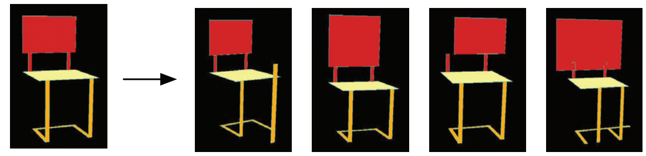

Figure 3: Chair modifications: scaling, scaling in one dimension, moving and a combination of all three.

The positive examples that were chosen as the initial labeled set in a certain experiment were excluded from the test set used in that experiment. In addition, the positive examples that were chosen as the initial labeled set were not used to produce the near miss examples in the near miss test set. This assured that there was no overlap between the training set and the test set used in an experiment. We implemented three of the modification operators: primitive part scaling, primitive part scaling in one dimension, and primitive part moving. The Nearest Neighbor algorithm was used for learning. In all experiments, unless stated otherwise, the value for the parameter n was set to 300, the value of the parameter p_{stop} was set to 0.2, λ was set to 0.4, and $\mathfrak{D}_0$ to 100. The values of λ and $\mathfrak{D}_0$ were chosen so that the value of the scaling parameter $\mathfrak{D}$ is very high at first, but decreases quickly. Results presented for each experiment are the average of 30 runs of that experiment. We refer to the active learning version of our algorithm as ANMG-LA and the passive learning as ANMG-R.

Figure 4 presents the learning curves of the ANMG-LA and the ANMG-R algorithms for each object class and on the two types of test sets as described above. The graphs show typical behavior for learning curves. The performance improves and then stabilizes after processing 15–30 examples, with ANMG-LA performing better than ANMG-R. The percentage of negative examples in the generated pools was around 85%. Among the examples that ANMG-LA chose to present to the expert during the first 50 learning iterations, 90% were labeled as negative. This shows that ANMG-LA does find near misses to be informative. We compared the performance of ANMG-LA to the perfor-

[1] One potential problem with this approach is that near misses may be more difficult to label for the expert than clearly negative or clearly positive examples.

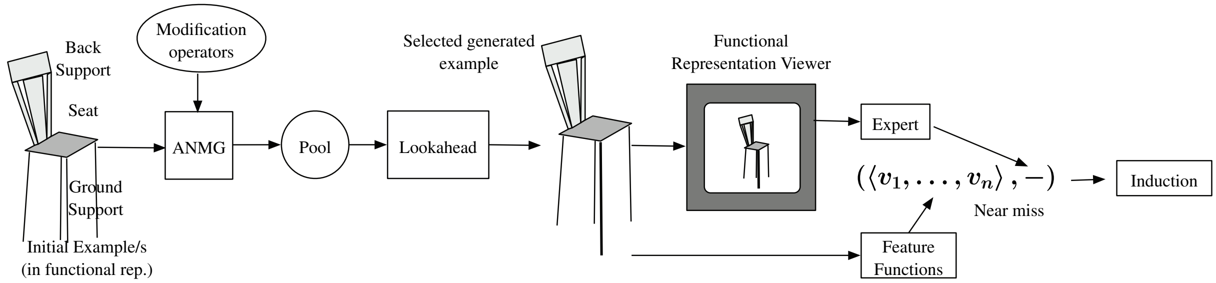

Figure 2: The flow of information during learning in the object recognition domain

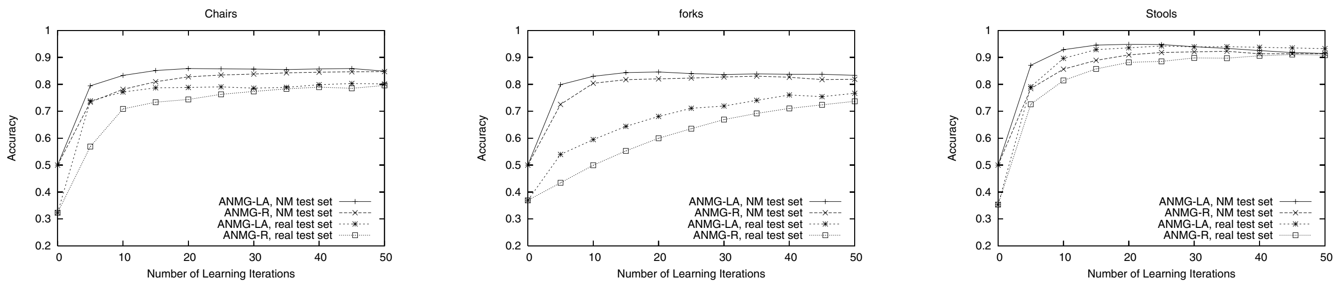

Figure 4: The learning curves of ANMG-LA and ANMG-R

mance of the ANMG-R algorithm using a variation of the paired t-test. An experiment was run on 30 problem setups. In each setup, the initial set of examples was identical for both algorithms. In order to reduce variation sources even further, (in a single problem setup) both algorithms used the same pool of generated unlabeled examples X_{gen} in order to select the next example for labeling. As mentioned above, the pools of examples were generated using the default parameter values: $n = 300, p_{stop} = 0.2$.

For each setup, we measured the accuracy of the classifiers produced by the ANMG-LA and the ANMG-R algorithms after performing 10 learning iterations. We denote the difference in algorithm performance for setup i as $\delta_i = acc_{la_i} - acc_{r_i}$, where acc_{la_i} and acc_{r_i} are the measured accuracies of the ANMG-LA and ANMG-R algorithms respectively. To reduce the noise associated with random selection, we calculated acc_{r_i} by running the ANMG-R algorithm 50 times with the fixed parameters and averaging the results. The difference in the performance of the two algorithms, denoted δ, was averaged over the experimental setups: $\delta = \frac{1}{30}\sum_i \delta_i$. Table 1 summarizes the differences in performance for the two algorithms, showing that at the initial stages of the learning the active approach is significantly better than the passive one.

Test Set	Concept	ANMG-R	ANMG-LA	δ	Sig
Real-World	stool	82.67%	89.65%	**6.98%**	√
	chair	69.34%	77.59%	**8.25%**	√
	fork	49.95%	59.58%	**9.53%**	√
Near Miss	stool	84.83%	92.87%	**8.04%**	√
	chair	77.49%	83.29%	**5.80%**	√
	fork	78.23%	82.98%	**4.75%**	√

Table 1: Performance after 10 iterations

Related Work

Rendell & Cho (1990) examine the effect of training data characteristics on learning and show that increasing the concentration of the training data near the classification boundary improves the resulting classification accuracy. Maloof & Michalski (2004) show that keeping extreme examples (which include near misses) in partial memory is efficient. In the work of Yip & Sussman (1997), near misses are sometimes created as a side effect of the generalization process. Porter & Kibler (1986) get an initial set of positive examples, which are then perturbated by applying modification operators. However, contrary to our approach, near misses are discarded and only positive near examples are used. Scholkopf, Burges, & Vapnik (1996) generate new examples without a teacher. To overcome the lack of a labeler, they use a set of modification operators that are known to be invariant with respect to the example labels. This is opposite to our approach. Whereas their method use invariant operators, we attempt, rather, to cross the classification boundary. Lopes & Camarinha-Matos (1995) also assume lack of a teacher and combine two neighboring examples of the same class to generate a new one, assuming that the new example belongs to the same class. Pomerleau (1991) takes yet another approach to obtain labels for the generated examples by defining modification operators that determine the example label.

Several works consider automatic generation of new examples for active learning. Hwang *et al.* (1991) generate boundary examples for training a neural network. Baum (1991) suggests generating examples that will help to find separating hyperplanes for the hidden units of a neural network. Melville & Mooney (2004) create diverse committees for active learning. Good diversity among the committee members is achieved by generating artificial examples and labeling them against the committee's prediction.

The above works are problematic in that they do not distinguish between the instance space and the feature space. The new examples are generated in the feature space, which causes difficulties because the mapping between the instance space and the feature space is not always one-to-one. A major drawback of this approach was shown by Lang & Baum (1992): when trying to apply an example generation method to the domain of images of handwritten characters, many of the images constructed by the algorithm did not contain any recognizable characters. Our work solves this problem by generating directly in the instance space.

Discussion

This paper proposes a new methodology for automatic generation of examples for inductive learning. The work builds on the early work of Winston, and extends it by proposing a general method for *automatic* generation of near misses. While near misses have been recognized as useful by many works in cognitive science and computer science, they are mostly assumed to be constructed by the teacher.

A possible critique of this work might be its lack of experimental comparison to other works. We seriously considered this point, but after performing a vast literature survey we concluded that such a comparison is impossible in our case. Most works that consider near misses assume their initial availability rather than generate them, rendering comparison impossible. Comparison with works that do consider automatic example generation is likewise impossible. Some works generate instances in the feature space—an approach that is inapplicable in the world of visual concepts. The few works that generate examples in the instance space generate new examples while keeping the example labels invariant.

Another possible critique might be that the functional approach is not suitable for all possible object recognition tasks. This is true in some cases—not all visual concepts can be described in functional terms. However, if we consider man-made objects, their physical structure usually reflects the task that they were created for (Stark & Bowyer 1991). Therefore, man-made objects, which surround us in the modern world, can be discussed in functional terms. Moreover, recognition of objects using the functional approach (and using an intermediate instance space) is only one implementation of the framework that we propose. The proposed framework is applicable to any problem domain where modification operators can be defined. In order to apply the proposed framework to a new problem domain, one needs to acquire positive examples of the concept to be learned as well as define modification operators over the instance space of the problem. The modification operators should be defined while taking into account how the instances are represented.

Acknowledgments

This work was partially supported by funding from the EC-sponsored MUSCLE Network of Excellence (FP6-507752).

References

Baum, E. B. 1991. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Transactions on Neural Networks* 2(1):5–19.

Cohn, D.; Atlas, L.; and Ladner, R. 1994. Improving generalization with active learning. *Machine Learning* 15(2):201–221.

Froimovich, G.; Rivlin, E.; and Shimshoni, I. 2002. Object classification by functional parts. In *Proceedings of the First International Symposium on 3D Data Processing Visualization and Transmission*, 648–655.

Hwang, J.-N.; Choi, J.; Oh, S.; and Marks, R.J., I. 1991. Query-based learning applied to partially trained multilayer perceptrons. *IEEE Transactions on Neural Networks* 2(1):131–136.

Lang, K. J., and Baum, E. B. 1992. Query learning can work poorly when a human oracle is used. In *IJCNN'92*.

Lindenbaum, M.; Markovitch, S.; and Rusakov, D. 2004. Selective sampling for nearest neighbor classifiers. *Machine Learning* 54(2):125–152.

Lopes, L., and Camarinha-Matos, L. 1995. Example generation and processing for inductive learning in the assembly domain. In *Proceedings of the 4th European Workshop on Learning Robots*.

Maloof, M. A., and Michalski, R. S. 2004. Incremental learning with partial instance memory. *Artificial Intelligence* 154:95–126.

Melville, P., and Mooney, R. 2004. Diverse ensembles for active learning. In *ICML'04*, 584–591.

Pechuk, M.; Soldea, O.; and Rivlin, E. 2005. Function-based classification from 3D data via generic and symbolic models. In *AAAI'05*.

Pomerleau, D. 1991. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation* 3(1):88–97.

Porter, B. W., and Kibler, D. F. 1986. Experimental goal regression: A method for learning problem-solving heuristics. *Machine Learning* 1(3):249–285.

Rendell, L., and Cho, H. 1990. Empirical learning as a function of concept character. *Machine Learning* 5(3):267–298.

Rivlin, E.; Dickinson, S. J.; and Rosenfeld, A. 1995. Recognition by functional parts. *Computer Vision and Image Understanding* 62(2):164–176.

Scholkopf, B.; Burges, C.; and Vapnik, V. 1996. Incorporating invariances in support vector learning machines. In *ICANN'96*, 47–52.

Stark, L., and Bowyer, K. W. 1991. Achieving generalized object recognition through reasoning about association of function to structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13(10):1097–1104.

Winston, P. H.; Binford, T.; Katz, B.; and Lowry, M. 1983. Learning physical descriptions from functional definitions, examples, and precedents. In *AAAI'03*, 433–439.

Winston, P. H. 1975. *The Psychology of Computer Vision*. New York, NY: McGraw-Hill. chapter Learning Structural Descriptions From Examples, 157–209.

Yip, K., and Sussman, G. J. 1997. Sparse representations for fast, one-shot learning. In *AAAI'97*, 521–527.