Learning To Recognize 3D Objects With SNoW

Ming-Hsuan Yang Dan Roth Narendra Ahuja

Department of Computer Science and Beckman Institute
University of Illinois at Urbana-Champaign, Urbana, IL 61801
mhyang@vison.ai.uiuc.edu danr@cs.uiuc.edu ahuja@vision.ai.uiuc.edu

Abstract. This paper describes a novel view-based learning algorithm for 3D object recognition from 2D images using a network of linear units. The SNoW learning architecture is a sparse network of linear functions over a pre-defined or incrementally learned feature space and is specifically tailored for learning in the presence of a very large number of features. We use pixel-based and edge-based representations in large scale object recognition experiments in which the performance of SNoW is compared with that of Support Vector Machines (SVMs) and nearest neighbor using the 100 objects in the Columbia Image Object Database (COIL-100). Experimental results show that the SNoW-based method outperforms the SVM-based system in terms of recognition rate and the computational cost involved in learning. Most importantly, SNoW's performance degrades more gracefully when the training data contains fewer views. The empirical results also provide insight into practical and theoretical considerations on view-based methods for 3D object recognition.

1 Introduction

View-based object recognition has attracted much attention in recent years. In contrast to methods that rely on pre-defined geometric (shape) models for recognition, view-based methods learn a model of the object's appearance in a two-dimensional image under different poses and illumination conditions. At evaluation time, given a two-dimensional image, the learned model is used to determine if the target object is present in the image or not.

Among the view-based object recognition methods, parametric eigenspace [11] [12] and support vector machine approaches [14] have demonstrated excellent recognition results on the COIL-20 and COIL-100 databases. Although these systems can recognize objects in almost real-time, the computational cost involved in learning is extremely high. Consequently these methods are typically demonstrated using only small, often different and unspecified subsets of objects from the whole database, which makes fair comparison of results difficult. More significantly, the training sets used in previous experimental studies consist of images taken in nearby poses (usually 10° apart). This particular experimental setup, as we will show, makes the learning problem less challenging. In order to study algorithms in a somewhat more realistic situation, it is of great interest to compare the performance of these methods when only a limited number of views of the objects are presented during training.

In this work, we propose a method that applies the SNoW (Sparse Network of Winnows) learning algorithm [16] [3] (available at `http://L2R.cs.uiuc.edu/~cogcomp.html`) to 3D object recognition and compare its performance with SVM and nearest neighbor methods. SNoW is a sparse network of linear functions that utilizes the Winnow update rule [8]. It is specifically tailored to learning in domains in which the potential number of features taking part in decisions is very large (and may be unknown a priori), although only a small number of them is typically relevant to a decision. Some of the characteristics of this learning architecture are its sparsely connected units, the allocation of features and links in a data driven way, the decision mechanism and the utilization of a feature-efficient update rule. An additional property of the SNoW architecture that makes it attractive for learning in vision is that it learns a representation for each object rather than a discrimination rule for each pair, as do other methods. This allows for more appealing evaluation schemes and for the incorporation of external information sources into the process of learning a representation and recognizing an object. SNoW has been used successfully on a variety of large scale learning tasks in natural language processing [16] [5] and recently, on face detection [21].

This paper is organized as follows. Previous work on view-based methods that learn to recognize 3D objects is described in Section 2 which also provides details on the use of SVMs for this problem. The SNoW learning architecture and its use for object recognition are presented in Section 3. Section 4 presents the experimental setup and an experimental comparison of the proposed method with SVM and nearest neighbor. The experimental comparison focuses on varying the number of view points and, for SNoW, also on different image representations. We conclude with some comments on these learning methods and future work in Section 5.

2 View-Based Methods

The appearance of an object is the combined effects of its shape, reflectance properties, pose, and the illumination in the scene. While shape and reflectance are intrinsic properties that do not change for a rigid object, pose and illumination vary from one scene to another. View-based recognition methods attempt to use data observed under different poses and illumination conditions to learn a compact model of the object's appearance; this, in turn, is used to resolve the recognition problem from view points that were not observed previously.

A number of view-based schemes have been developed to recognize 3D objects. Poggio and Edelman [13] show that 3D objects can be recognized from the raw intensity values in 2D images (we call this representation here a *pixel-based representation*) using a network of generalized radial basis functions. They argue and demonstrate that full 3D structure of an object can be estimated if enough 2D views of the object are provided. Turk and Pentland [18] demonstrate that human faces can be represented and recognized by "eigenfaces." Representing a face image as a vector of pixel values, the eigenfaces are the eigenvectors associated with the largest eigenvalues which are computed from a covariance matrix of the sample vectors. An attractive feature of this method is that the eigenfaces can be learned from the sample images in pixel representation without any feature selection. The eigenspace approach has since been used in different vision tasks from face recognition to object tracking. Murase and Nayar [11] [12] develop a parametric eigenspace method to recognize 3D objects directly from their appearance. For each object of interest, a set of images in which the object appears in different poses is obtained as training examples. Next, the eigenvectors are computed from the covariance matrix of the training set. The set of images is projected to a low dimensional subspace spanned by a subset of eigenvectors, in which the object is represented as a manifold. A compact parametric model is constructed by interpolating the points in the subspace. In recognition, the image of a test object is projected to the subspace and the object is recognized based on the manifold it lies on. Using a subset of the Columbia Object Image Library (COIL-100), they show that 3D objects can be recognized accurately from their appearances in real-time.

In contrast to these algebraic methods, general purpose learning methods such as support vector machines (SVMs) have also been used for this problem. Schölkopf [17] was the first to apply SVMs to recognize 3D objects from 2D images and has demonstrated the potential of this approach in visual learning. Pontil and Verri [14] also used SVMs for 3D object recognition and experimented with a subset of the COIL-100 dataset. Their training set consisted of 36 images (one for every 10°) for each of the 32 objects they chose, and the test sets consist of the remaining 36 images for each object. For 20 random selections of 32 objects from the COIL-100, the system achieves perfect recognition rate (but see comments on that in Sec. 4). More recently, a subset of the COIL-100 has been used by also Roobaert and Van Hulle [15] to compare the performance of SVMs with different pixel-based input representations.

Given the success of this approach, which we use to compare with the approach presented here, we present below the SVM method in some more details.

2.1 Support Vector Machines

The Support Vector Machine (SVM) [20] [4] is a general purpose learning method for pattern recognition and regression problems that is based on the theory of structural risk minimization. According to the structural risk minimization inductive principle, a function that describes the training data well and belongs to a set of functions with low VC dimension[1] will generalize well (that is, will guarantee a small expected recognition error for the unseen data points) regardless of the dimensionality of the input space [20]. Based on this principle, the SVM is a systematic approach to find a linear function (a hyperplane) that belongs to a set of functions of this forms with the lowest VC dimension. The reason for using a linear function is that for a set of linearly separable points, it is possible to explicitly quantify the VC dimension in terms of the minimal distance between positive and negative points. SVMs provide non-linear function approximations

[1] The VC dimension of a class of functions is a combinatorial parameter that measures the richness of the function class. See [20] [6] for details.

by mapping the input vectors into a high dimensional feature space where a linear hyperplane that separates the data exists. It can also be extended to cases where the best hyperplane in the resulting high dimension space does not quite separate all the data points.

Given a set of samples $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$, $\ldots$, $(\mathbf{x}_l, y_l)$ where $\mathbf{x}_i \in R^N$ is the input vector and $y_i \in \{-1, 1\}$ its label, an SVM aims to find an optimal hyperplane that leaves the largest possible fraction of data points of the same class on the same side while maximizes the distance of either class from the hyperplane (margin distance). Vapnik [20] shows that maximizing the margin distance is equivalent to minimizing the VC dimension and therefore contributes to better generalization. The problem of finding the optimal hyperplane is thus posed as a constrained optimization problem and solved using quadratic programming techniques. The optimal hyperplane, which determines the class label of a data point $\mathbf{x} \in R^N$, is of the form

$$f(\mathbf{x}) = sgn(\sum_{i=1}^{l} y_i \alpha_i \cdot k(\mathbf{x}, \mathbf{x}_i) + b)$$

where $k(\cdot, \cdot)$ is a kernel function and sgn is the function that outputs $+1$ on positive inputs and -1 otherwise. Constructing an optimal hyperplane is equivalent to determining the nonzero α_is. Sample vectors $\mathbf{x}_i$ that corresponds to a nonzero α_i are called the *support vector*s (SVs) of the optimal hyperplane. The hope, when using this method, is for a small number of support vectors, thereby producing a compact classifier.

The use of kernel functions allows, using Mercer theorem, to avoid the need to blow up the dimensionality in order to reach a state in which the sample is linearly separable. If the kernel is of the form $k(\mathbf{x}, \mathbf{x}_i) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i)$ for some nonlinear function $\Phi : R^N \to F^M, \;\; M \gg N$, the computation can be done in the original, lower dimension space rather than working in the M dimensional space, although the hypoerplane is constructed in R^M. For a linear SVM, the kernel function is simply the dot product of vectors in the input space. Several kernel functions, such as polynomial functions and radial basis functions, have been shown to satisfy Mercer theorem and used in nonlinear SVM, allowing the construction of a variety of learning machines, some of which coincide with classical architectures. However, this also results in a drawback since one needs to find the "right" kernel function when using SVMs. It is interesting to observe, though, that although the use of kernel functions seems to be one of the advantages of SVMs from a theoretical point of view, most experimental studies have used linear SVMs which were found to perform better. One potential reason is that SVMs are prone to outliers and various kinds of noise in the data, and this gets worse when non-linear kernels are used.

3 A SNoW-Based Approach to Object Recognition

The SNoW learning architecture, the focus of this work, also finds its origin in computational learning theory [19] and relies on VC theory to relate its behavior on the training data to that on unseen test data, just like SVMs. And, as in SVMs, the learning architecture uses linear functions. One main difference is the way the linear functions are learned from the data. However, the focus in SNoW is significantly different in several respects. First, it attmepts to learn representations for the target concepts, rather than discriminators. This allows SNoW units to serve as input when learning other, more involved representations [10]. Second, the empahsis is on an arhictecture and algorithms that can deal efficiently with very high dimensional spaces, both in terms of the numebr of examples required to learn and in terms of the computational complexity of learning and evaluation. Therefore, generating expressive features (i.e., blowing up the dimensionality of the space) in order to guarantee linear separability, if needed, is not a problem. This algorithmic aspect makes the architecture especially advantagueous when the function space is sparse (that is, the target definition depends on a few relevant attributes relative to the overall dimensionality of the space) and when not all the attributes that are used to describe instances are known ahead of time. This latter property is important for scalability, but also for future considerations when, for example, additional information sources may become available to a recognition module only in later stages of the process. In this section, we first present the SNoW learning architecture and algorithm, and then describe how we apply SNoW algorithm to 3D object recognition.

3.1 The SNoW Architecture

The SNoW (Sparse Network of Winnows[2]) learning architecture is a sparse network of linear units over a common pre-defined or incrementally learned feature space. Nodes in the input layer of the network represent simple relations over the input instance and are being used as the input features. Each linear unit is called a *target node* and represents relations or concepts which are of interest over the input; in the current application, target nodes represent a definition of an object in terms of the relations (features) extracted from the 2D image input. An input instance is mapped into a set of features which are active in it; this representation is presented to the input layer of SNoW and propagates to the target nodes. Target nodes are linked via weighted edges to (some of) the input features.

Let $\mathcal{A}_t = \{i_1, \ldots, i_m\}$ be the set of features that are active in an example and are linked to the target node t. Then the linear unit corresponding to t is *active* iff

$$\sum_{i \in \mathcal{A}_t} w_i^t > \theta_t,$$

where w_i^t is a positive weight on the edge connecting the ith feature to the target node t, and θ_t is the threshold for the target node t.

Each SNoW *unit* may include a collection of subnetworks, one for each of the target relations but all using the same feature space. In the current case, we may have one unit with target subnetworks for all the target objects or we may define different units, each with two competing target objects. A given example

[2] To winnow: to separate chaff from grain.

is treated autonomously by each target subnetwork; an example labeled t may be treated as a positive example by the subnetwork for t and as a negative example by the rest of the target nodes.

The learning policy is on-line and mistake-driven; several update rules can be used within SNoW. The most successful and the only one used in this work, is a variant of Littlestone's Winnow update rule [8], a multiplicative update rule that we tailored to the situation in which the set of input features is not known a priori, as in the infinite attribute model [1]. This mechanism is implemented via the sparse architecture of SNoW. That is, (1) input features are allocated in a data driven way – an input node for the feature i is allocated only if the feature i was active in any input sentence and (2) a link (i.e., a non-zero weight) exists between a target node t and a feature i if and only if i was active in an example labeled t.

One of the important properties of the sparse architecture is that the complexity of processing an example depends only on the number of features active in it, n_a, and is independent of the total number of features, n_t, observed over the life time of the system. This is important in domains in which the total number of features is very large, but only a small number of them is active in each example.

The Winnow update rule has, in addition to the threshold θ_t at the target t, two update parameters: a *promotion* parameter $\alpha > 1$ and a *demotion* parameter $0 < \beta < 1$. These are being used to update the current representation of the target t (the set of weights w_i^t) only when a mistake in prediction is made. Let $\mathcal{A}_t = \{i_1, \ldots, i_m\}$ be the set of active features that are linked to the target node t. If the algorithm predicts 0 (that is, $\sum_{i \in \mathcal{A}_t} w_i^t \leq \theta_t$) and the received label is 1, the active weights in the current example are *promoted* in a multiplicative fashion:

$$\forall i \in \mathcal{A}_t, w_i^t \leftarrow \alpha \cdot w_i^t.$$

If the algorithm predicts 1 ($\sum_{i \in \mathcal{A}_t} w_i^t > \theta_t$) and the received label is 0, the active weights in the current example are *demoted*:

$$\forall i \in \mathcal{A}_t, w_i^t \leftarrow \beta \cdot w_i^t.$$

All other weights are unchanged.

The key feature of the Winnow update rule is that the number of examples required to learn a linear function grows linearly with the number n_r of *relevant* features and only logarithmically with the total number of features. This property seems crucial in domains in which the number of potential features is vast, but a relatively small number of them is relevant. Moreover, in the sparse model, the number of examples required before converging to a linear separator that separates the data (provided it exists) scales with $O(n_r \log n_a)$. Winnow is known to learn efficiently any linear threshold function and to be robust in the presence of various kinds of noise and in cases where no linear-threshold function can make perfect classifications, while still maintaining its abovementioned dependence on the number of total and relevant attributes [9] [7].

Once target subnetworks have been learned and the network is being evaluated, a decision support mechanism is employed, which selects the dominant active target node in the SNoW unit via a winner-take-all mechanism to produce a final prediction. In other applications the decision support mechanism may also cache the output and process them along with the output of other SNoW units to produce a coherent output.

Figures 1, 2 and 3 provide more details on the SNoW learning architecture.

SNoW: *Objects and Notation*

$F = \mathcal{Z}^+ = \{0, 1, \dots\}$	/* Set of potential features */
$T = \{t_1, \dots t_k\} \subset F$	/* Set of targets */
$F_t \subseteq F$	/* Set of features linked to target t */
$t_{NET} = \{[(i, w_i^t) : i \in F_t], \theta_t\}$	/* The representation of the target t. */
$activation : T \rightarrow \Re$	/* activation level of a target t. */
$SNoW = \{t_{NET} : t \in T\}$	/* The SNoW Network */
$e = \{i_1, \dots, i_m\} \subset F^m$	/* An example, represented as a list of active features */

Fig. 1. SNoW: *Objects and Notation*

SNoW: *Training and Evaluation*

Training Phase: SNoW-Train (SNoW, e)

Initially: $F_t = \phi$, for all $t \in T$.
For each $t \in T$
1. UpdateArchitecture (t, e)
2. Evaluate (t, e)
3. UpdateWeights (t, e)

Evaluation Phase: SNoW-Evaluation(SNoW, e)

For each $t \in T$
    Evaluate (t, e)
MakeDecision (SNoW, e)

Fig. 2. SNoW: *Training and Evaluation*. Training is the learning phase in which the network is constructed and weights are adjusted. Evaluation is the phase in which the network is evaluated, given an observation. This is a conceptual distinction; in principle, one can run in on line mode, in which training is done continuously, even when the network is used for evaluating examples.

SNoW: *Building Blocks*

Procedure Evaluate(t, e)

activation $= \sum_{i \in e} w_i^t$

Procedure UpdateWeights(t, e)

If $(activation(t) > \theta_t)$ & $(t \notin e)$ /* predicted positive on negative example */
for each $i \in e$: $w_t^i \leftarrow w_i^t \cdot \beta$
If $(activation(t) \leq \theta_t)$ & $(t \in e)$ /* predicted negative on a positive example */
for each $i \in e$: $w_t^i \leftarrow w_i^t \cdot \alpha$

Procedure UpdateArchitecture(t, e)

If $t \in e$
– For each $i \in e \setminus F_t$, set $w_i^t = w$ /* Link feature to target; set initial weight */
Otherwise: do nothing

Procedure MakeDecision(SNoW, e)

Predict winner $= argmax_{t \in T} activation(t)$ /* Winner-take-all Prediction */

Fig. 3. SNoW: *Main Procedures*

3.2 Learning 3D Objects with SNoW

Applying SNoW to 3D object recognition requires specifying the architecture used and the representation chosen for the input images. As described above, to perform object recognition we associate a target subnetwork with each target object. This target learns a definition of the object in terms of the input features extracted from the image. We could either define a single SNoW unit which contains target subnetworks for all the 100 different target objects, or we may define different units, each with several (e.g., two) competing target objects. Selecting a specific architecture makes a difference both in training time, where learning a definition for object a makes use of negative examples of other objects that are part of the same unit but, more importantly, it makes a difference in testing; rather that two competing objects for a decision, there may be a hundred. The chances for a spurious mistake caused by an incidental view point are clearly much higher. On the other hand, it has significant advantages in terms of space complexity and the appeal of the evaluation mode. This point will be discussed later.

An SVM is a two-class classifier which, for an n-class pattern recognition problem, trains $\frac{n(n-1)}{2}$ binary classifiers. Since we compare the performance of the proposed SNoW-based method with SVMs, in order to maintain a fair comparison we have to perform it in the *one-against-one* scheme. That is, we use SNoW units of size two. To classify a test instance, tournament-like pair-wise competition between all the machines is performed and the winner determines the label of the test instance. The recognition rates of the SVM and SNoW based methods shown in Table 2 were performed using the one-against-one scheme. (That is, we trained $\binom{100}{2} = 4,950$ classifiers for each method and evaluated $98(= 50 + 25 + 12 + 6 + 3 + 1 + 1)$ classifiers on each test instance.

Pixel-Based Representation The first feature representation used in this work is a set of Boolean features that encode the positions and intensity values of pixels. Let the (x, y) pixel of an image with width w and height h have intensity value $I(x, y)$ $(0 \leq I(x, y) \leq 255)$. This information is encoded as a feature whose index is $256 \times (y \times w + x) + I(x, y)$. This representation ensures that different points in the $\{\texttt{position} \times \texttt{intensity}\}$ space are mapped to different features. (That is, the feature indexed $256 \times (y \times w + x) + I(x, y)$ is *active* if and only if the intensity in position (x, y) is $I(x, y)$.) In our experiments images are normalizes so that $w = h = 32$. Note that although the number of potential features in our representation is $262,144$ $(32 \times 32 \times 256)$, only 1024 of those are active (present) in each example, and it is plausible that many features will never be active. Indeed, in one of the experiments, it turned out that only 13,805 of these features were ever active. Since the algorithm's complexity depends on the number of active features in an example, rather than the total number of features, the sparseness also contributes to efficiency. Also notice that while this representation seems to be too simplistic, the performance levels reached with it are surprisingly good.

Edge-Based Representation Edge information contains significant visual cues for human perception and has the potential to provide more information than the previous representation and guarantee robustness. Edge-based representations can be used, for example, to obtain a hierarchical description of an object. While perceptual grouping has been applied successfully to many vision problems including object and face recognition, the grouping procedure is usually somewhat arbitrary. This word can this be viewed as a systematic method to learn representation of objects based on conjunctions of edges.

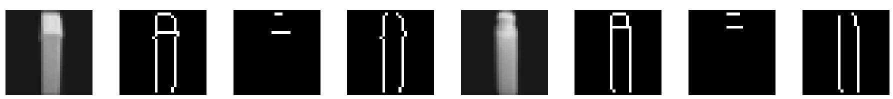
For each image, a Canny edge detector [2] is first applied to extract edges. Let $I(x, y)$ represent the (x, y) pixel in an image I. Let $E(x, y)$ be the Canny edge map in which $E(x, y) = 1$ indicates the existence of an edge at $I(x, y)$. To prune extraneous small edge fragments and reduce the computation complexity we keep only edges with length above some threshold (e.g., 3 pixels). $\hat{E}$ is the resulting edge map after pruning. That is, the pixel $I(x, y)$ is considered to contain significant perceptual information to describe the object, when $\hat{E}(x, y) = 1$; otherwise, $\hat{E}(x, y) = 0$. For consistency we index an edge using its top left pixel. For each pixel we maintain up to two possible edges in the resulting $\hat{E}$ map, a vertical one and a horizontal one, denoted by

$$e =< (x, y), d >, d \in \{v, h\}.$$

Features are generated to represent conjunctions of size two of these edges. That is, features are elements of the cross product

$$\hat{E} \times \hat{E} = \{(e, e') | e \neq e'\}.$$

This representation thus constitutes a hierarchical representation of an object which encodes the spatial relationships of the edges in an object. Figure 4 illustrates the benefit of this encoding in object recognition. It shows two objects with very similar appearance for which the edge maps (where the minimum length is 3) are different. Note that some of edges are blurred or missing because of the the aggressive downsampling (from 128×128 to 32×32 pixels). Nevertheless, this difference grows when conjunctions of edges are used. Finally, we note that the number of potential features when using this representation is very large, but very few of them are active.



(a) object 65 (b) edge map (c) hor. edge (d) vert. edge (e) object 13 (f) edge map (g) hor. edge (h) vert. edge

Fig. 4. Two objects with similar appearance (in terms of shape and intensity values) but their edge maps are very different. Note that some of the edges are blurred or missing because of aggressive downsampling (from 128×128 to 32×32 pixels).

4 Experiments

We use the COIL-100 dataset [11] to test our method and compare its performance with other view-based methods in the literature. In this section, we describe the characteristics of the COIL-100 dataset, present some experiments in which the performance of several methods is compared and discuss the empirical results.

4.1 Dataset and Experimental Setups

We use the Columbia Object Image Library (COIL-100) database in all the experiments below. COIL is available at http://www.cs.columbia.edu/CAVE. The COIL-100 dataset consists of color images of 100 objects where the images of the objects that were taken at pose intervals of 5°, i.e., 72 poses per object. The images were also normalized such that the larger of the two object dimensions (height and width) fits the image size of 128×128 pixels. Figure 5 shows the images of the 100 objects taken in frontal view, i.e., zero pose angle. The 32 highlighted objects in Figure 5 are considered more difficult to recognize in [14]; we use all 100 objects including these in our experiments. Each color image is converted to a gray-scale image of 32×32 pixels for our experiments. In this paper, given the use of the COIL-100 dataset, it is assumed that the illumination conditions remain constant and hence object pose is the only variable of interest.

Fig. 5. Columbia Object Image Library (COIL-100) consists of 100 objects of varying poses (5° apart). The objects are shown in row order where the highlighted ones are considered more difficult to recognize in [14].

4.2 Ground Truth of the COIL-100 Dataset

At first glance, it seems difficult to recognize the objects in the COIL dataset because it consists of a large number of objects with varying pose, texture, shape and size. Since each object has 72 images of different poses (5° apart), many view-based recognition methods use 36 (10° apart) of them for training and the remaining images for testing. However, it turns out that under these dense sampling conditions the recognition problem is not difficult (even when only grey-level images are used). Namely, in this case, instances that belong to the same object are very close to each other in the image space (where each data point represents an image of an object in a certain pose). We verified this by experimenting with a simple nearest neighbor classifier (using the Euclidean distance), resulting in an average recognition rate of 98.50% (54 errors out of 3,600 tests). Figure 4.2 shows some of the objects misclassified by nearest neighbor method.

In principle, one may want to avoid using the nearest neighbor method since it requires a lot of memory for storing templates and its recognition time complexity is high. The goal here was simply to show that this simple method is comparable to the complex SVM approaches [14] [15] for the case of dense sampling. Therefore, the abovementioned recognition problem is not appropriate for comparison among different methods.

(a) (8:80,23:85) (b) (31:80,79:85) (c) (65:270,13:265) (d) (69:80,91:75) (e) (96:260,69:85)

Fig. 6. Mismatched objects using the nearest neighbor method. $(x : a, y : b)$ means that object x with view angle a is recognized as object y with view angle b. It shows some of the 54 errors (out of 3,600 test samples) made by the nearest neighbor classifier when there are 36 views per object in the training set.

It is interesting to see that the pairs of the objects on which the nearest neighbor method misclassified have similar geometric configurations and similar

Table 1. Recognition rates of nearest neighbor classifier

Results	30 objects randomly selected from COIL	32 objects shown in Figure 5 selected by [14]	The whole 100 objects in COIL
Errors/Tests	14/1080	46/1152	54/3600
Recognition rate	98.70%	96.00%	98.50%

poses. A close inspection shows that most of the recognition errors are made between the three packs of chewing gums, bottles and cars. Other dense sampling cases are easier for this method. Consequently, the set of selected objects in an experiment has direct effects on the recognition rate. This needs to be taken into account when evaluating results that use only a subset of the 100 objects (typically 20 to 30) from the COIL dataset for experiments. Table 1 shows the recognition rates of nearest neighbor classifiers in several experiments in which 36 poses of each object are used for templates and the remaining 36 poses are used for tests.

Given this baseline experiment we have decided to perform our experimental comparisons in cases in which the number of views of objects available in training is limited.

4.3 Empirical Results Using Pixel-Based Representation

Table 2 shows the recognition rates of the SNoW-based method, the SVM-based method (using linear dot product for the kernel function), and the nearest neighbor classifier using the COIL-100 dataset. The important parameter here is that we vary the number of views of an object (n) during training and use the rest of the views ($72 - n$) of an object for testing.

Table 2. Experimental results of three classifiers using the 100 objects in the COIL-100 dataset

Methods	# of views/object			
	36	18	8	4
	3600 tests	5400 tests	6400 tests	6800 tests
SNoW	95.81%	92.31%	85.13%	81.46%
Linear SVM	96.03%	91.30%	84.80%	78.50%
Nearest Neighbor	98.50%	87.54%	79.52%	74.63%

The experimental results show that the SNoW-based method performs as well as the SVM-based method when many views of the objects are present during training and outperforms SVM-based method when the numbers of views is limited. Although it is not surprising to see that the recognition rate decreases as the number of views available during training decreases, it is worth noticing that both SNoW and SVM are capable of recognizing 3D objects in the COIL-100 dataset with satisfactory performance if enough views (e.g., > 18) are provided. Also they seems to be fairly robust even if only a limited number of views (e.g., 8 and 4) are used for training; the performance of both methods degrades gracefully.

To provide some more insight into these methods, we note that in the SVM-based methods, only 27.78% (20 out of 72) of the input vectors serves as support vectors. For SNoW, out of 262,144 potential features in the pixel-based representation, only 13,805 were active in the dense case (i.e., 36 views). This shows the advantage gained from using the sparse architecture. However, only a small number of those may be relevant to the representation of each target, as a more careful look as the SNoW output hypothesis reveals.

An additional potential advantage of the SNoW architecture is that it does not learn discriminators, but rather can learn a representation for each object, which can then be used for prediction in the one-against-all scheme or to build hierarchical representations. However, as is shown in Table 3, this implies a significant degradation is the performance. Finding a way to make better predictions in the one-against-all scheme is one of the important issues for future investigation, to better exploit the advantages of this approach.

Table 3. Recognition rates of SNoW using two learning paradigms

SNoW	# of views/object			
	36	18	8	4
one-against-one	95.81%	92.31%	85.13%	81.46%
one-against-all	90.52%	84.50%	81.85%	76.00%

4.4 Empirical Results Using Edge-Based Representation

For each 32×32 edge map, we extract horizontal and vertical edges (of length at least 3 pixels) and then encode as our features conjunctions of two of these edges. The number of potential features of this sort is $\binom{2048}{2} = 2,096,128$. However, only an average of 1,822 of these is active for objects in the COIL-100 dataset. To reduce the computational cost the feature vectors were further pruned and only the 512 most frequently occurring features were retained in each image.

Table 4 shows the performance of the SNoW-based method when conjunctions of edges are used to represent objects. As before, we vary the number of views of an object (n) during training and use the rest of the views ($72 - n$) of an object for testing. The results indicate that conjunctions of edges provide useful information for object recognition and that SNoW is able to learn very good object representations using these features. The experimental results also exhibit the relative advantage of this representation increases when the number of views per object is limited.

Table 4. Experimental results of SNoW classifier on the COIL-100 dataset using conjunction of edges

SNoW	# of views/object			
	36	18	8	4
	3600 tests	5400 tests	6400 tests	6800 tests
w/ conjunction of edges	96.25%	94.13%	89.23%	88.28%
w/ primitive intensity values	95.81%	92.31%	85.13%	81.46%

5 Discussion and Conclusion

We have described a novel view-based learning method for the recognition of 3D objects using SNoW. Empirical results show that the SNoW-based method outperforms other methods in terms of recognition rates except for the dense case (36 views). Furthermore, the computational cost of training SNoW is smaller.

Beyond the experimental study and developing a better understanding for how and when to compare experimental approaches, the main contribution of this work is in presenting a way to apply the SNoW learning architecture to visual learning.

Unlike previous general purpose learning methods like SVMs, SNoW learns representations for objects, which can then be used as input to other, more involved, visual processes in a hierarchical fashion. An aspect of the recognition problem that we have not addressed here is the ability to use the representation to quickly prune away objects that cannot be valid targets for a given test image so that rapid recognition from among a small set of reasonable candidates is performed. We believe that the SNoW architecture, by virtue of learning a positive definition for each object rather than a discriminator between any two, is more suitable for this problem, and this is one of the future directions we pursue. For a fair comparison among different methods, this paper uses pixel-based presentation in the experiments. However, we view the edge-based representation that was found to be even more effective and robust as another starting point for future research. We believe that pursing the direction of using complex intermediate representations will benefit future work on recognition and, in particular, robust recognition under various types of noise. We pursue this notion also as part of an attempt to provide a learning theory account for the object recognition problem using the PAC (Probably Approximately Correct) [19] framework.

Acknowledgements

Dan Roth is supported by NSF grant IIS-9801638. Ming-Hsuan Yang and Narendra Ahuja are supported by Office of Naval Research under grant N00014-96-0502. The supports are gratefully acknowledged.

References

1. A. Blum. Learning boolean functions in an infinite attribute space. *Machine Learning*, 9(4):373–386, Oct. 1992.
2. J. F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.
3. A. Carleson, C. Cumby, J. Rosen, and D. Roth. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department, May 1999.
4. C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20, 1995.
5. A. R. Golding and D. Roth. A winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34:107–130, 1999. Special Issue on Machine Learning and Natural Language.
6. M. Kearns and U. Vazirani. *Introduction to computational Learning Theory*. MIT Press, 1994.
7. J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. In *Proceedings of the Annual ACM Symp. on the Theory of Computing*, 1995.
8. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
9. N. Littlestone. Redundant noisy attributes, attribute errors, and linear threshold learning using Winnow. In *Proc. 4th Annu. Workshop on Comput. Learning Theory*, pages 147–156, San Mateo, CA, 1991. Morgan Kaufmann.
10. M. Munoz, V. Punyakanok, D. Roth, and D. Zimak. A learning approach to shallow parsing. In *EMNLP-VLC'99, the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, June 1999.
11. H. Murase and S. K. Nayar. Visual learning and recognition of 3-d objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995.
12. S. K. Nayar, S. A. Nene, and H. Murase. Real-time 100 object recognition system. In *Proceedings of IEEE International Conference on Robotics and Automation*, 1996.
13. T. Poggio and S. Edelman. A network that learns to recognize 3d objects. *Nature*, 343:263–266, 1990.
14. M. Pontil and A. Verri. Support vector machines for 3d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(6):637–646, 1998.
15. D. Roobaert and M. V. Hulle. View-based 3d object recognition with support vector machines. In *IEEE International Workshop on Neural Networks for Signal Processing*, 1999.
16. D. Roth. Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 806–813, 1998.
17. B. Schölkopf. *Support Vector Learning*. PhD thesis, Informatik der Technischen Universitat Berlin, 1997.
18. M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.
19. L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, Nov. 1984.
20. V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
21. M.-H. Yang, D. Roth, and N. Ahuja. A SNoW-based face detector. In *NIPS-12; The 1999 Conference on Advances in Neural Information Processing Systems*, pages 855–861. MIT Press, 2000.