Dynamic Visual Category Learning

Tom Yeh
MIT EECS & CSAIL
Cambridge, MA, USA
tomyeh@mit.edu

Trevor Darrell
UC Berkeley EECS & ICSI
Berkeley, CA, USA
trevor@eecs.berkeley.edu

Abstract

Dynamic visual category learning calls for efficient adaptation as new training images become available or new categories are defined, existing training images or categories become modified or obsolete, or when categories are divided into subcategories or merged together. We develop novel methods for efficient incremental learning of SVM-based visual category classifiers to handle such dynamic tasks. Our method exploits previous classifier estimates to more efficiently learn the optimal parameters for the current set of training images and categories. We show empirically that for dynamic visual category tasks, our incremental learning methods are significantly faster than batch retraining.

1. Introduction

Visual category learning is a problem that has received much attention in the computer vision community over the past decades. Most current visual category learning methods operate in a static setting, assuming the numbers of categories and training images are fixed [3, 13, 7, 9, 22]. However, real-world problems are often dynamic and incremental: the set of training images may change over time or the definition of the target classes may evolve over time. At any given moment, the best classification can only be obtained if the classifier can take full advantage of all the training examples observed up to that moment.

For example, in a robot learning scenario, we may want to teach a robot to classify objects in a home environment where the objects encountered are described with natural language interaction from the user and therefore the number of training examples and the number of categories will change over time. In a photo categorization scenario, we similarly expect a user to have interests and category definitions that evolve as the user adds new photographs. In both cases, visual category classifier needs to adapt to the updated knowledge contained in the current set of training images and categories. We prefer systems that can incrementally update themselves rather than retraining each time a new example is added or a new category is defined.

We consider **dynamic visual category learning** and present efficient incremental methods for updating SVM classifiers when new training images become available, existing training images are modified or removed, new categories are defined, and/or existing categories become obsolete, are divided into subcategories, or are merged together.

Figure 1 shows an interactive photo-organizer tool to demonstrate the features of dynamic visual category learning. This tool allows users to organize photos in a taxonomy with the help of visual category classifiers. These classifiers are created and updated incrementally as the users define and arrange the categories in the taxonomy or drag and drop training images into areas designated for each category. Given new data, these classifiers classify unlabeled images and suggest new candidate images for each category. The users can easily identify more training images from the candidate images to add to the classifiers. Our incremental methods make it possible to implement such a dynamic visual category learning tool which may be useful for interactive taxonomy construction and other tasks.

We review related work in Section 2, provide background material in Section 3, describe our incremental methods in Section 4, and show experimental results in Section 5.

2. Related Work

Many authors have reported promising results on *static* visual category learning tasks [22, 3, 13, 9, 19]. In static visual category learning, data collection is performed offline as a separate first step, whereas in dynamic visual category learning, data collection takes place online, closely intertwined with the other learning steps. Incremental learning has been explored in several visual category learning tasks, such as the use of Adaboost to incrementally learn object detectors based on edge features [16], and the use of Markov Chain Monte Carlo sampling to incrementally learn latent topic models for online photos [14]. However, recent best-performing methods for visual category learning often

978-1-4244-2243-2/08/$25.00 ©2008 IEEE

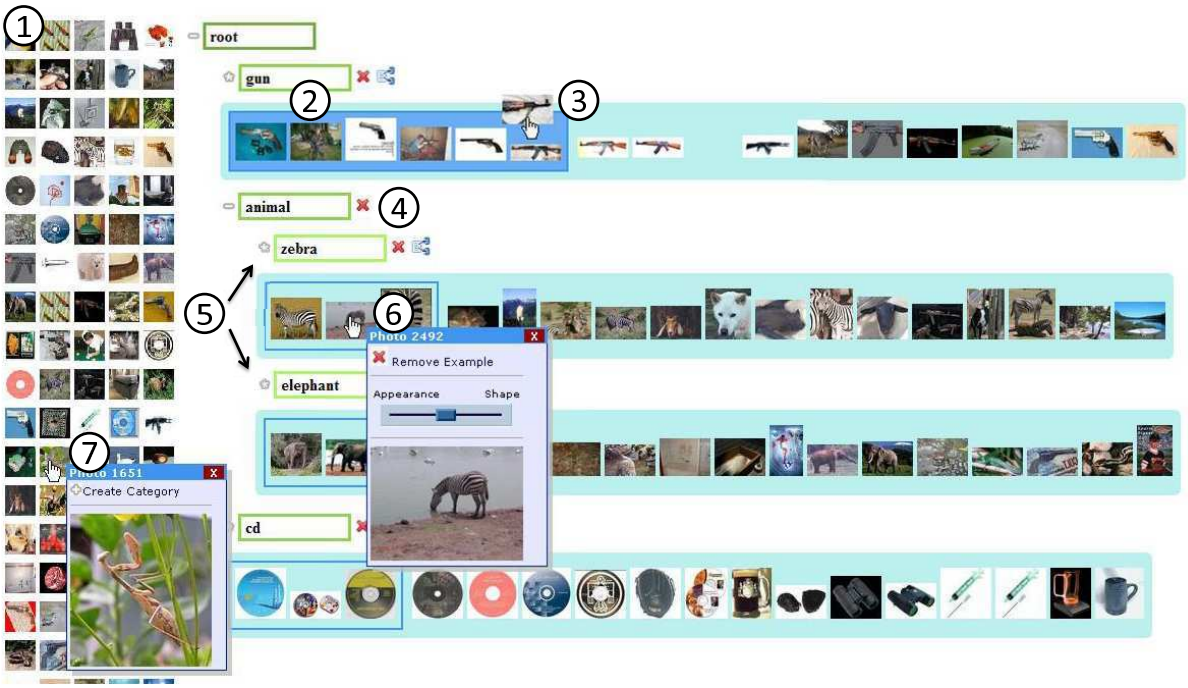

Figure 1. **An application of dynamic visual category learning**: This is an interface of an online photo categorization application that takes the full advantage of the various features of dynamic visual category learning. An online user can interactively (1) browse a photo collection, (2) select a few examples for the category (e.g., gun), (3) view examples classified by the SVM trained from the selected training examples and drag-and-drop a new training example to the gun category (Section 4.1), (4) delete a category (Section 4.5), (5) split a category into sub-categories (Section 4.6), (6) remove an example (Section 4.2) or adjust the feature weight (i.e., appearance vs. shape) of that example (Section 4.3), (7) or create a new category from an unclassified example (Section 4.4). Each user action can trigger an update to the classifier, which can be handled efficiently using our incremental methods.

use an SVM as the underlying classifier.

Various incremental methods are proposed for adding new training examples to existing SVMs [1, 4, 12, 8]. However, they do not handle cases when training examples are modified nor have they been extended to multiclass problems. [18] extends [8] to incrementally train a multiclass classifier and reports results on a synthetic dataset, but offers no method for category-level update. [6] describes a method for incrementally learning an ensemble of SVMs for OCR, using strategically updated distributions of the training set. Although this method incrementally adds new examples to the ensemble, SVMs for new categories are still trained in batch. In contrast, we propose methods that can exploit existing parameters to incrementally learn new parameters not only for new examples but also for new categories.

A learning problem closely related to our work is multiclass active learning [20, 11]. The task in active learning is to analyze unlabeled examples in a dataset and identify the example whose label is most likely to help improve the classifier. [20] describes an active framework for labeling video sequences, whereas [11] proposes a framework for image retrieval. Both works use a margin-based classifier to perform the learning task. For these frameworks to be efficient, the underlying SVM needs to be incrementally updated instead of retrained when a new label is available.

The need to update the visual vocabulary when new training images become available has been identified by [21]. But for kernel-based visual category learning, updating the visual vocabulary can result in changes in the kernel computed based on the representation using the modified vocabulary. While the vocabulary update is incremental, the underlying SVM still needs to be retrained every time. This shortcoming makes the method in [21] inefficient when applied to category learning problems. Our method overcomes this limitation because it updates SVM parameters directly given the changes in the kernel matrix; it is applicable to both static and dynamic visual vocabularies.

3. Background: SMO-based SVM learning

Our incremental approach to dynamic visual category learning is based on Sequential Minimal Optimization (SMO) [17], a fast and efficient algorithm for learning SVM parameters. *lib*SVM [5], a popular SVM package used by many recent works on visual category learning [13, 22], implements a variant of this algorithm. The memory require-

ment of this algorithm is minimal because there is no need to keep temporary matrices as in chucking-based decomposition techniques. The low memory requirement makes this algorithm attractive for large category learning problems with thousands of images, such as Caltech 101 [7] or 256 [10].

The basis of SVM learning is a QP optimization problem that seeks to find a set of weights α_i for example vectors $\vec{x}_i$ that satisfy the KTT conditions:

$$\begin{aligned} \alpha_i = 0 &\Rightarrow y_i f(\vec{x}_i) \geq 1, \\ 0 \leq \alpha_i \leq C &\Rightarrow y_i f(\vec{x}_i) = 1, \\ \alpha_i = C &\Rightarrow y_i f(\vec{x}_i) \leq 1. \end{aligned} \tag{1}$$

where y_i is the label of $\vec{x}_i$ and $f(\vec{x}_i)$ gives the margin of $\vec{x}_i$. Note that $\vec{x}_i$ is a support vector iff $0 < \alpha_i < C$.

To find the sets of weights that can satisfy the KTT condition, SMO begins by initializing the weight α_i of each example $\vec{x}_i$ to zero. It decomposes the optimization problem into the smallest possible subproblems involving only two examples, $\vec{x}_p$ and $\vec{x}_q$, jointly optimizing the objective function with respect to their weights α_p and α_q. Each optimization step is fast because the optimal weights can be found analytically as follows:

$$\begin{aligned} \alpha_q^{opt} &= \alpha_q - y_2 \tfrac{(E_p - E_q)}{\eta} \\ \alpha_p^{opt} &= \alpha_p + s(\alpha_q - \alpha_q^{opt}) \end{aligned}$$

where $s = y_p y_q$, η is the second derivative of the objective function, and E_i is the error of $\vec{x}_i$, indicating how much $\vec{x}_i$ violates the KTT conditions, which can be calculated as:

$$E_i = f(\vec{x}_i) - y \tag{2}$$

At each step, the weights of the two examples with the largest errors take part in the joint optimization. Since the error calculation dominates the computation time, the algorithm caches the error E_i of each example $\vec{x}_i$ and updates its value incrementally by:

$$E_i \leftarrow E_i + y_p \Delta\alpha_p k(\vec{x}_p, \vec{x}_i) + y_q \Delta\alpha_q k(\vec{x}_q, \vec{x}_i) + \Delta b$$

where Δb is the change in the offset. The algorithm converges when all the errors are below some small threshold.

Although the SMO algorithm requires all the weights to be initialized to zeros, non-zero initial weights can still converge to an optimal solution when the underlying kernel matrix satisfies the Mercer's condition. In fact, careful choices of initial values can significantly shorten the convergence time. This observation forms the basis of our incremental approach to dynamic visual category learning, which we turn to in the next section.

4. Dynamic Visual Category Learning

In this section, we describe how to extend the existing SVM-based approach to static visual category learning to handle the following dynamic events: a new training image becomes available (Section 4.1), an existing training image becomes irrelevant (Section 4.2) or is modified (Section 4.3), a new category becomes necessary (Section 4.4), or an existing category becomes obsolete (Section 4.5), divided into subcategories (Section 4.6), or merged together with other categories (Section 4.7).

Figure 2 shows examples of adding, removing, and modifying examples using a synthetic dataset for a two-class SVM. For a multiclass SVM, Figure 3 shows examples of adding, removing, and merging categories, and Figure 4 shows an example of creating subcategories.

4.1. Adding a new example

In dynamic visual category learning, new training images can become available over time . Given a new training example $\vec{x}$, we want to update the current SVM and obtain a new SVM that incorporates $\vec{x}$. Instead of re-estimating the parameters of the new SVM from scratch (i.e., initializing all the weights to zeros), we start the SMO procedure by reusing the current weights and setting the weight of the new example to zero. The motivation behind this method is that a new training example may modify only some parts of the current decision hyperplane; the new hyperplane is not expected to be completely different from the current one. Therefore, a new optimal solution is likely to be reasonably close to the current solution in the search space. By reusing the parameters of the current hyperplane, the optimization procedure can converge sooner to the parameters of the new hyperplane versus starting from the default starting point. When the kernel matrix is a Mercer's kernel, this method is guaranteed to converge to the same optimal solution because the optimization problem is convex.

In realistic online learning scenarios, new training examples incrementally modify the decision hyperplane in a relatively smooth manner; however, in certain pathological situations, new training examples may alter the hyperplane in a way so drastic that no parameter is worth transferring from the current hyperplane to the next hyperplane. For example, observing a new example labeled by a human as negative but predicted by the current SVM to have an extremely large positive margin may trigger fundamental changes to the structure of the current classifier in order to account for the surprising observation. In such cases, initializing the optimization procedure to the current parameter values may put the optimizer in a worse position than resetting the parameters to zeros, and the incremental method may provide no computational advantage. However, we have found empirically that these cases are rare.

4.2. Removing an existing example

To remove an existing example $\vec{x}$ from the current SVM, there are two cases we need to consider. The first is when

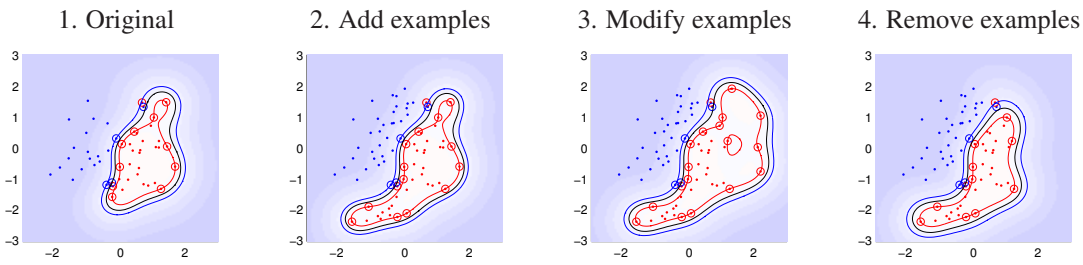

Figure 2. **Incremental update of a single SVM:** Each dot is a training point. A circle indicates that the point is a support vector. The white area is the positive region. (1) An SVM decision hyperplane separates the two groups of training points. (2) The hyperplane expands toward the lower-left as more positive points are added to that region (Section 4.1). (3) The hyperplane expands to the top-right as some existing points moved outward that region (Section 4.3). (4) The hyperplane retreats as some positive points are removed (Section 4.2).

$\vec{x}$ is not a support vector. We can simply discard the example, since $\vec{x}$ is not involved in the definition of the decision hyperplane. On the other hand, when $\vec{x}$ is a support vector, removing $\vec{x}$ in effect decreases its weight α to zero. To bring the objective function back to a new optimal state would require either some non-support vectors to assume the role of the missing support vector, or some remaining support vectors to increase their respective weights, in order to compensate for the loss of α. To achieve this, we simply find the example $\vec{x}'$ closest to $\vec{x}$ and allow $\vec{x}'$ to increase its weight α' by α without exceeding C (i.e., $\alpha' \leftarrow \min(\alpha' + \alpha, C)$). Instead of the costlier application of Equation 2, the cached error of each remaining support vector $\vec{x}_i$ can also be incrementally computed by

$$E_i \leftarrow y'\alpha k(\vec{x}, \vec{x}_i) - y\Delta\alpha' k(\vec{x}', \vec{x}_i)$$

where y and y' are labels of $\vec{x}$ and $\vec{x}'$ respectively. The optimization procedure started with these initial conditions not only can converge sooner, but also is guaranteed to find the optimal solution if the kernel satisfies Mercer's condition.

4.3. Modifying one or more existing examples

In dynamic visual category learning, one or more training examples already added to an SVM may be modified due to external circumstances, such as when the label has changed, when the weights of individual images are adjusted, or when the underlying visual vocabulary is updated. The affected SVM is no longer valid unless its parameters are also updated accordingly.

First we consider the cases when a single example $\vec{x}_k$ is modified by $\Delta\vec{x}_k$. If $\Delta\vec{x}_k$ is small, simply resuming the SMO procedure using the current parameters can quickly converge to a new set of optimal SVM parameters. The cached errors can be updated incrementally as follows:

$$E_i \leftarrow E_i + y_k \Delta\vec{x_k}, \forall i \neq k.$$

When multiple examples are modified, there are two alternatives for incrementally updating the SVM parameters. The simpler alternative is the **indirect example update** method where we sequentially remove each example and add its modified version back to the classifier.

A better alternative is the **direct kernel update** method where we analytically derive new support vector weights $\vec{\alpha}_s$ given the changes in the kernel matrix ΔK. The direct kernel update method is an extension of the method described in [4] for updating support vector weights $\vec{\alpha}_s$ when a new support vector $\vec{x}_k$ is introduced with weight α_k. Their method is based on a reformulation of the KTT conditions:

$$\begin{bmatrix} \vec{y}_s \Delta\vec{f}_s \\ 0 \end{bmatrix} = \begin{bmatrix} \vec{y}_s & \tilde{K}_{ss} \\ 0 & \vec{y}_s^T \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta\vec{\alpha}_s \end{bmatrix} + \Delta\alpha_k \tilde{K}_{ks}^T$$

where the unknown are the changes in the weights of the existing support vectors $\Delta\vec{\alpha}_s$ and the changes in the offset Δb, and the known are the weight of the new support vector $\Delta\alpha_k$, the parts of the kernel matrix corresponding to the support vectors K_{ss}, the signs of the other support vectors $\vec{y}_s$, and the changes to their margins $\vec{f}_s$.

When all the support vectors are still valid support vectors after the update, the changes to their margins $\Delta\vec{f}_s$ must be zero so that they can continue to meet the KTT condition necessary for support vectors (i.e., $yf(\vec{x}) = 1$). Knowing the right-hand size is zero, we can solve this linear equation for $\Delta\vec{\alpha}_s$ and Δb. However, sometimes adding $\Delta\vec{\alpha}_s$ to $\vec{\alpha}_s$ may cause some support vectors to become invalid (i.e., $\alpha < 0$ or $\alpha > C$). Therefore, it is important to apply $\Delta\alpha_k$ piecewise and calculate appropriate $\Delta\vec{\alpha}_s$ to maintain a set of valid support vectors at all times (see [12] for details).

We extend the above method to handle the cases when changes are not restricted to a single vector but possibly to multiple vectors. When multiple vectors are modified, all rows and columns of the kernel matrix that correspond to the modified vectors are also modified. Let ΔK denote the change in the kernel matrix. We can reformulate the KTT

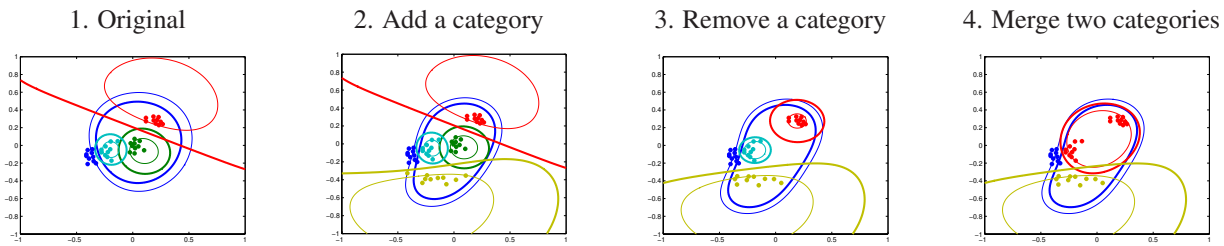

Figure 3. **Basic category-level dynamic SVM update:** (1) A four-class classifier with four 1-vs-all SVMs. Thick and thin color lines respectively mark the decision hyperplane and the side of the positive region of each category. (2) A new category (gold) is added to the classifier, by creating the fifth SVM and incrementally adding the new points to the other four SVMs as negative examples (Section 4.4). (3) An existing category (green) is removed (Section 4.5). (4) Two categories (red, aqua) are merged into one category (red) (Section 4.7).

conditions in terms of ΔK as follows:

$$\begin{bmatrix} \vec{y}_s \Delta \vec{f}_s \\ 0 \end{bmatrix} = \begin{bmatrix} \vec{y}_s & (\tilde{K}_{ss} + \Delta \tilde{K}_{ss}) \\ 0 & \vec{y}_s^T \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta \vec{\alpha}_s \end{bmatrix} + \begin{bmatrix} \Delta \tilde{K}_{ss} \\ 0 \end{bmatrix} \vec{\alpha}_s$$

Similarly, knowing that $\Delta \vec{f}_s = 0$, we can solve the linear system above to obtain $\Delta \vec{\alpha}_s$ and Δb. Also, we need to maintain the set of valid support vectors by piecewise application of ΔK in the same way as [12]. In Section 5.3 we show that our direct kernel update method was significant faster than the indirect example update alternative, especially when the number of modified examples is large.

4.4. Adding a new category

In dynamic visual category learning, a new category y can be encountered or deemed useful by a user. Suppose there are n training images of the existing categories and m new images of the category y. We want to incrementally train a new 1-vs-all SVM S_y for category y and add S_y to the current classifier ensemble Ψ:

$$\Psi^{new} = \Psi \cup \{S_y\}.$$

Our incremental method has two steps. First, a new SVM S_y is trained in batch mode (standard SMO) using the m new images as *positive* examples and the n existing images as *negative* examples. Second, each existing SVM $S_{y_i} \in \Psi$ needs to be updated incrementally by adding the m new images to S_{y_i} as *negative* examples, using the method described in Section 4.1. When $n \gg m$, our incremental method can enjoy the greatest speed advantage by reusing the knowledge embedded in the original n examples instead of relearning from scratch.

To further optimize the speed in which the images are inserted into S_{y_i} as negative examples, we give priority to those images with the largest positive margin values predicted by its decision hyperplane f. An image $\vec{x}$ with a larger margin is positioned further into the positive side of the hyperplane. Since $\vec{x}$ is to be added as a negative example, we need to shift the hyperplane toward the positive side to allow the negative size to grow in such a way that $\vec{x}$ will eventually fall under the negative side of the new hyperplane. While the negative side is expanding, some other $\vec{x}'$ with a margin smaller than $\vec{x}$ may also benefit from the expansion, entering the negative side together with $\vec{x}$; adding $\vec{x}'$ would take no time at all following the addition of $\vec{x}$. We show empirically in Section 5.4 that margin priority ordering results in a 5% to 20% extra time saving.

4.5. Removing an existing category

In dynamic visual category learning, an existing category may become obsolete and require the removal of the corresponding 1-vs-all SVM from the ensemble. Our incremental category removal method involves two steps. First, we remove the SVM S_y corresponding to the obsolete category y from the current ensemble Ψ:

$$\Psi^{new} = \Psi - \{S_y\}$$

Second, we undo the influence of the obsolete training images on the remaining SVMs, by incrementally removing them using the method described in Section 4.2.

4.6. Creating new subcategories

New categories can also be created by dividing an existing category y into subcategories $y'_1 \ldots y'_m$ (Figure 4). The original SVM S_y for the category y needs to be replaced by a set of SVMs trained for the new subcategories, which can be expressed as below:

$$\Psi^{new} = \Psi \cup \{S_{y'_1} \ldots S_{y'_m}\} - \{S_y\}$$

where $S_{y'_i}$ is the new SVM of the subcategory y'_i. Each new SVM can be derived incrementally from the original SVM S_y as follows:

1. Create a new 1-vs-all SVM $S_{y'_i}$ and copy its parameters from the original SVM S_y.

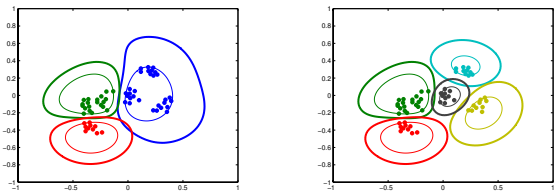

Figure 4. **Incremental subcategory creation:** (1) Three 1-vs-all SVMs separate three categories of color points. (2) The SVM for the blue category is replaced by three 1-vs-all SVMs for subcategories black, aqua, and gold. Each subcategory SVM is incrementally derived by cloning the SVM of the original category (blue) and incrementally switching the labels of the points in the other two subcategories from positive to negative (Section 4.6).

2. Incrementally modify the labels of the examples pertinent to the other subcategories from positive to negative, using the method described in Section 4.3.

4.7. Merging two existing categories

In dynamic visual category learning, existing categories can be merged into a single category. Given two categories p and q, the objective is to replace the SVMs S_p and S_q for the two categories with a new SVM $S_{p \wedge q}$ that generalizes to their superclass:

$$\Psi^{new} = \Psi \cup \{S_{p \vee q}\} - \{S_p, S_q\}$$

which can be achieved incrementally as follows:

1. Let p be the category with more training examples and q be the other category.
2. Create $S_{p \vee q}$ and copy its parameters from S_p.
3. Incrementally modify the examples of category q from negative examples to positive examples using the method described in Section 4.3.

5. Experiments

In this section, we first show the accuracy benefit of updating SVM parameters when the set of training examples changes over time (Section 5.1). Then, we present four experiments to show the speed benefit of our incremental methods for updating SVM parameters: we show incremental update is faster than batch retraining when new training examples become available (Section 5.2), our direct kernel update method is faster than indirect example update when training examples are modified (Section 5.3), margin priority ordering provides additional time benefit for category creation (Section 5.4), and incremental update is faster than batch retraining when categories change over time in an interactive taxonomy learning task (Section 5.5).

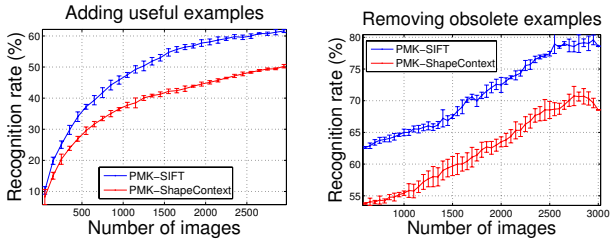

Figure 5. **Recognition rate vs. dynamic dataset:** Recognition rate can be improved as more useful training examples are added to the classifier (left) or as obsolete examples are removed from the classifier (right) (Section 5.1).

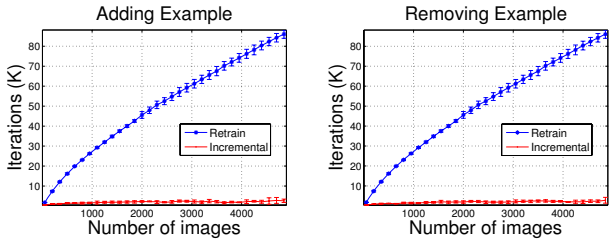

Figure 6. **Incremental update vs. batch retraining:** To add (left) or to remove (right) an example, as the size of the SVM grows (x-axis), the computation cost of retraining (blue, higher) grows dramatically, whereas the computation cost of incremental update (red, lower) stays fairly constant (Section 5.2).

5.1. Recognition rate vs. dynamic dataset

This experiment studies the accuracy benefit of updating SVM parameters when the set of training images grows or shrinks over time. We used the Caltech 101 dataset [7] with a Spatial Pyramid Match Kernel (SPMK) [13] based on two different features: SIFT [15] and ShapeContext [2]. First we measured the effect of adding useful training images on the accuracy of the classifier. We randomly chose 3000 images to form the training set and added them to the classifier in a random order. We also measured the accuracy in terms of the mean recognition rate as defined in [13]. Second, we measured the effect of removing obsolete images on the accuracy. We used the same training set to obtain the initial classifier. Then we deliberately kept only 20 categories and deemed the other 81 categories obsolete, removing the images of the obsolete categories in a random order. We measured the mean recognition rate of the 20 categories.

Figure 5 shows the that the recognition rate improved as more useful images were incorporated into the classifier or as more obsolete images were removed from the classifier, which suggests the importance of updating the classifier with the most up-to-date set of relevant training images.

5.2. Incremental update vs. retraining

This experiment examines the speed advantage of incremental update (Section 4.1) over batch retraining when a

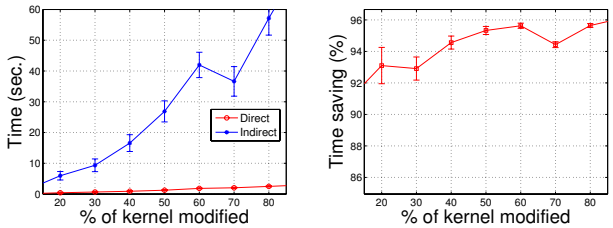

Figure 7. **Direct kernel update vs. indirect example update:** The direct kernel update takes less time (left, y-axis) than the naive method for updating the SVM parameters given a modified kernel matrix. As larger portions of the kernel (x-axis) are modified, the time-saving increases (right, y-axis) (Section 5.3).

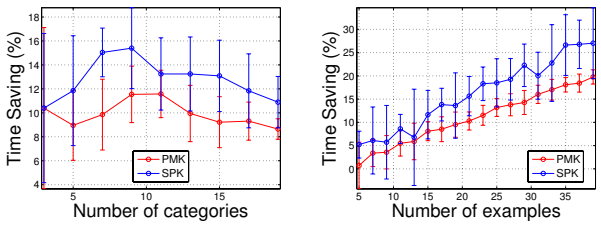

Figure 8. **Time saving by margin priority ordering:** The time saving stays fairly constant as the number of categories increases (left), but increases as the number of examples of the new category increases (right) (Section 5.4).

new training image is added to an SVM. For each n from 100 to 5000, we trained an SVM on n randomly selected images from the Caltech 101 [7] dataset, using the standard SMO with a spatial pyramid match kernel (SPMK) [13]. We measured the running time of adding the $n+1$-st example by incremental update and by batch retraining.

Figure 6 shows as the number of training images increased, incremental update achieved relatively constant time performance, whereas the computation cost of retraining rose drastically.

5.3. Direct kernel vs. indirect example update

This experiment aims to determine whether our direct kernel update method described in Section 4.3 for modifying multiple examples is faster than the indirect example update alternative that sequentially removes examples and adds them back. Intuitively, the more entries in the kernel that have been modified, the longer it will take to relearn new SVM parameters. We were interested in the effect the number of kernel changes has on the computation time to update the SVM. We simulated kernel changes by adding random noise to a fraction of the entries in the kernel. We varied the percentage from 5% to 80% and compared the computation time of our direct method with that of the indirect method. Figure 7 shows that our direct kernel update method is significantly faster than the indirect method.

5.4. Margin priority vs. random ordering

This experiment evaluates the effect of margin priority ordering on the running time of our incremental category creation method described in (Section 4.4). There are two factors that can affect the running time: the size of the current classifier (number of categories/SVMs) and the size of the new category (number of images). If a large classifier with many categories is to incorporate a new category, there are more SVMs we need to update. If a large category with many images is to be added, it may take longer to add these images to the other SVMs. We designed two experiments to study each factor. First, we varied the size of the classifier (5 to 20 categories) while keeping the size of the new category constant (10 training images). Second, we varied the size of the new category (5 to 40 examples) while keeping the size of the classifier constant (5 categories). We measured the running time of our incremental method with and without margin priority ordering, and calculated the difference as the benefit of this ordering scheme. Each experiment was repeated ten times using a pyramid match kernel (PMK) [9] and a Spatial Pyramid Kernel (SPK) [3] on Caltech 101 [7] and Caltech 256 [10] datasets respectively.

Figure 8 shows the time saving by margin priority ordering for incremental category creation. The saving was between 5% to 25%. As the size of the existing classifier increased, the saving stayed fairly constant. On the other hand, as the size of the new category increased, the saving rose steadily, which may suggest that more training images were able to benefit from the restructuring of the hyperplane caused by earlier training images with larger margins.

5.5. Interactive taxonomy learning

In this section we evaluate the speed benefit of our method for online, interactive applications, such as the one shown in Figure 1. We focus on *interactive taxonomy learning*, a scenario where a user interactively constructs a taxonomy from a set of unlabeled object images, where a dynamic visual classifier acts as an assistant along the way.

A typical scenario is as follows: a user wants to organize images into categories, but does not have a concrete idea about how many categories are necessary and how fine-grained these categories should be. Thus, the user simply creates two or three most general categories (animate, inanimate, others), and starts assigning images into these categories. Every time an image is assigned, the classifier is updated incrementally. The classifier can suggest to the user a list of probable images for each category, and constantly refreshes the list after every update. From the list the user can easily identify more training examples for each category. After assigning a sufficient number of images to a particular category (e.g., animate), the user may notice finer distinctions within the category and may desire to split the

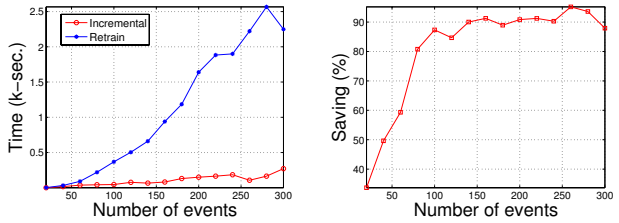

Figure 9. **Incremental update vs. retraining in interactive taxonomy learning** (Section 5.5).

category into subcategories (e.g., animal, plant, insect). As the user continues to do so, eventually the user will be able to organize the image collection into a meaningful taxonomy as so desired by the user.

To simulate the interactive session above, we used the taxonomy contained in the Caltech 256 dataset [10] and engineered a sequence of dynamic events to construct a taxonomy as if it were done by a human user. We chose a subset of the taxonomy, 30 categories at the leaf level, a branch factor of three in the internal nodes (e.g., plant → cactus, grapes, tomato) and two (i.e., root → animate, inanimate) at the top-most level. Ten training images were selected for each category. These images were added to the classifier in a random order, starting with only two SVMs for the two top-most categories (i.e., animate, inanimate). When a category had more than 10 images, it was split into three subcategories. At each event, we measured the running time of our incremental method versus that of retraining. Figure 9 shows that our incremental method was faster than retraining, especially when the taxonomy grew larger.

6. Conclusion

We described incremental methods for dynamic category learning which can efficiently update SVM parameters in visual recognition tasks. Our methods extend previous incremental techniques to handle both example and category level update. We demonstrated an interactive categorization application which uses dynamic update, evaluated our method on taxonomy formation tasks, and showed our methods are faster than retraining.

References

[1] J.-L. An, Z.-O. Wang, and Z.-P. Ma. An incremental learning algorithm for support vector machine. In *ICMLC*, 2003. 2

[2] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *PAMI*, 2001. 6

[3] A. Bosch, A. Zisserman, and X. Munoz. Representing shape with a spatial pyramid kernel. In *CIVR*, 2007. 1, 7

[4] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *NIPS*, 2000. 2, 4

[5] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines (version 2.31). 2

[6] Z. Erdem, R. Polikar, F. Gurgen, and N. Yumusak. Ensemble of svms for incremental learning. In *Multiple Classifier Systems*, pages 246–256. 2005. 2

[7] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *CVPR Workshop*, 2004. 1, 3, 6, 7

[8] G. Fung and O. L. Mangasarian. Incremental support vector machine classification. In *SDM*, 2002. 2

[9] K. Grauman and T. Darrell. The pyramid match kernel: discriminative classification with sets of image features. In *CVPR*, 2005. 1, 7

[10] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical report, California Institute of Technology, 2007. 3, 7, 8

[11] S. C. H. Hoi and M. R. Lyu. A semi-supervised active learning framework for image retrieval. In *CVPR*, 2005. 2

[12] P. Laskov, C. Gehl, S. Kruger, and K.-R. Muller. Incremental support vector learning: Analysis implementation and applications. *Journal of Machine Learning Research*, 7:1909–1936, 2006. 2, 4, 5

[13] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 1, 2, 6, 7

[14] L. Li, G. Wang, and L. Fei-Fei. Optimol: automatic online picture collection via incremental model learning. In *CVPR*, 2007. 1

[15] D. G. Lowe. Object recognition from local scale-invariant features. In *CVPR*, 1999. 6

[16] A. Opelt, A. Pinz, and A. Zisserman. Incremental learning of object detectors using a visual shape alphabet. In *CVPR*, 2006. 1

[17] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research, 1998. 2

[18] A. Tveit and M. Hetland. Multicategory incremental proximal support vector classifiers. In *Knowledge-Based Intelligent Information and Engineering Systems*. 2003. 2

[19] M. Varma and D. Ray. Learning the discriminative power-invariance trade-off. In *ICCV*, 2007. 1

[20] R. Yan, J. Yang, and A. Hauptmann. Automatically labeling video data using multi-class active learning. In *ICCV*, 2003. 2

[21] T. Yeh, J. Lee, and T. Darrell. Adaptive vocabulary forests for dynamic indexing and category learning. In *ICCV*, 2007. 2

[22] H. Zhang, A. C. Berg, M. Maire, and J. Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *CVPR*, 2006. 1, 2