Parallel incremental SVM for classifying million images with very high-dimensional signatures into thousand classes

Thanh-Nghi Doan, Thanh-Nghi Do and François Poulet

Abstract—ImageNet dataset [1] with more than 14M images and 21K classes makes the problem of visual classification more difficult to deal with. One of the most difficult tasks is to train a fast and accurate classifier on computers with limited memory resource. In this paper, we address this challenge by extending the state-of-the-art large scale classifier Power Mean SVM (PmSVM) proposed by Jianxin Wu [2] in three ways: (1) An incremental learning for PmSVM, (2) A balanced bagging algorithm for training binary classifiers, (3) Parallelize the training process of classifiers with several multi-core computers. Our approach is evaluated on 1K classes of ImageNet (ILSVRC 1000 [3]). The evaluation shows that our approach can save up to 84.34% memory usage and the training process is 297 times faster than the original implementation and 1508 times faster than the state-of-the-art linear classifier (LIBLINEAR [4]).

I. INTRODUCTION

Visual classification is one of the important topics in computer vision and machine learning. The usual frameworks involve three steps: 1) extracting local image features, 2) building codebook and encoding features, and 3) training classifiers. These frameworks are evaluated on small datasets, e.g. Caltech 101 [5], Caltech 256 [6] and PASCAL VOC [7]. In step 3, most researchers choose either linear or non-linear SVM classifiers that can be trained in a few minutes.

However, ImageNet with very large number of classes poses more challenges in training classifiers. ImageNet is much larger in scale and diversity than the other benchmark datasets. The current released ImageNet has grown a big step in terms of the number of images and the number of classes, as shown in Fig. 1 - it has 21,841 classes with more than 1000 images for each class on average.

With millions of training examples or dimensions, training an accurate classifier may take weeks or even years [8], [9]. Recent works in large scale learning classifiers converge on building linear SVM classifiers, because it is possible to train linear classifiers (e.g. LIBLINEAR) in order of seconds, even with millions training examples. However, for visual classification tasks, linear classifier is inferior in terms of accuracy, compared to non-linear classifier [2], [10], [11]. Wu [2] proposes PmSVM that outperforms LIBLINEAR and other additive kernel classifiers in terms of training time and classification accuracy. Nevertheless, there are two main drawbacks preventing PmSVM to scaleup to large scale datasets. Firstly, PmSVM loads the whole data into memory for training classifiers. Therefore, when data is larger and cannot fit into memory, PmSVM encounters a problem. For instance, the winners of ImageNet Challenge 2010 [3] report that their method train classifiers on a dataset in hundreds of giga-bytes. Secondly, the current version of PmSVM does not take into account the benefits of high performance computing (HPC). On ILSVRC 1000, it takes very long time to train binary classifiers. Therefore, it motivates us to study how to extend PmSVM for large scale visual classification. Our key contributions include:

1. An incremental learning for PmSVM. Our approach avoids loading the whole data into memory by splitting data into small blocks of rows stored in separate files and then a block of rows is loaded into memory for training classifiers at any one time.

2. A balanced bagging algorithm for training binary classifiers. Our algorithm avoids training on full block of rows and the training process of PmSVM rapidly converges to the optimal solution.

3. Parallelize the training process of these binary classifiers based on HPC models. In the training step, we apply our balanced bagging algorithm to achieve the best performance.

Our approach is evaluated on the 100 largest classes of ImageNet and 1K classes of ILSVRC 1000. The experiment shows that our approach can save up to 84.34% main memory usage and the training process is 297 times faster than the
original implementation and 1508 times faster than LIBLINEAR without (or very few) compromising classification accuracy. Therefore, it can be easily applied to dataset larger than the memory capacity of computer.

The remainder of this paper is organized as follows. Section II briefly reviews the related work on large scale visual classification. Section III introduces Power Mean SVM. In section IV, we present an incremental learning for PmSVM. Section V describes how to speedup the training process of PmSVM by using balanced bagging algorithm and take the benefits of HPC. Section VI presents numerical results before the conclusion and future work.

II. RELATED WORK

Low-level local image features, bag-of-words model (BoW [12]) and support vector machines (SVMs [13]) are the core of state-of-the-art visual classification systems. These may be enhanced by multi-scale spatial pyramids [14] on BoW or histogram of oriented gradient [15] features. Some recent works consider exploiting the hierarchical structure of dataset for image recognition and achieve impressive improvements in accuracy and efficiency [16]. Related to classification is the problem of detection, often treated as repeated one-versus-all classification in sliding windows [7], [17]. In many cases, such localization of objects might be useful for improving classification accuracy performance. However, in the context of large scale visual classification with hundreds or thousands classes, these common approaches become computationally intractable.

To address this problem, Fergus et al. [18] study semi-supervised learning on 126 hand labeled Tiny Images categories, Wang et al. [19] show classification experiments on a maximum of 315 categories. Li et al. [20] do research with landmark classification on a collection of 500 landmarks and 2 million images. On a small subset of 10 classes, they have improved BoWs classification by increasing the visual vocabulary up to 80K visual words.

However, the emergence of ImageNet makes the complexity of visual classification become much larger and very difficult to deal with. To tackle this challenge, many researchers are beginning to study strategies to improve the classification accuracy and avoid using high cost non-linear kernel SVM classifiers. The prominent works are proposed in [8], [9], [21], [22] where the data are first transformed by a nonlinear mapping induced by a particular kernel and then a linear classifier is trained in the resulting space. They argue that the classification accuracy of linear classifier with high-dimensional image signature is similar to low-dimensional BoW with non-linear classifier. In [9], each local descriptor is coded either using Local Coordinate Coding [23] or Supper-vector Coding [24], after performing spatial pyramid pooling the resulting image representation is a vector in approximately 262K dimensions. To train classifiers, they propose a parallel averaging stochastic gradient descent (ASGD) algorithm. With 1K classes of ILSVRC 1000, it takes 4 days to finish training 1K binary SVM classifiers (one-versus-all) for one feature channel on three 8-core computers. Sánchez and Perronin [22] study the impact of high dimensional Fisher vectors on large dataset. They show that the larger the training dataset, the higher the impact of the dimensionality on the classification accuracy. To get the state-of-the-art result on ILSVRC 1000, they use the spatial pyramids to increase the dimensionality of their Fisher vectors to approximately 524K dimensions and then exploit Product Quantizer [25] to compress the data before training classifiers. With this approach, training 1K SGD SVM classifiers (one-versus-all) for one feature channel takes 1.5 days on a 16-core computer.

In contrast with the approaches using linear SVM, recent works show that in visual classification tasks, non-linear SVM is superior in terms of accuracy, compared to linear rivalry. In many cases, non-linear SVM with additive kernels have significantly higher rate in accuracy performance than dot product kernel. The main drawback of these approaches is the high cost of training classifiers. It may be thousands of times higher than linear classifiers. However, some recent solutions are proposed to solve this limitation [10], [11], [2]. Additive kernel SVMs now use only few times more training time, compared to the state-of-the-art linear SVM classifiers. Recently, Wu [2] proposes an efficient algorithm for PmSVM. They show empirically that PmSVM outperforms LIBLINEAR and state-of-the-art additive kernel SVMs in terms of both training time and classification accuracy. For instance, on ILSVRC 1000, PmSVM is 5 times faster than LIBLINEAR and 2 times faster than state-of-the-art additive kernel implementations while yielding a significant improvement in classification accuracy from +4.6% to +7.1%. However, PmSVM has two main limitations: (1) it encounters a problem when data cannot fit into memory, (2) the training time is very long on ILSVRC 1000 (at least 10 hours) due to learning 1K binary classifiers sequentially, independently.

III. POWER MEAN SUPPORT VECTOR MACHINES

Let us consider a binary linear classification task with a training set \( T = \{ (x_i, y_i) \}_{i=1}^n \), \( x_i \in \mathbb{R}^d \), \( y_i \in \{+1,-1\} \). SVM classification algorithm aims to find the best separating surface as being furthest from both classes. It can simultaneously maximize the margin between the support planes for each class and minimize the error. This can be performed by solving the dual optimization problem (1).

\[
\begin{align*}
\min_{\alpha \in \mathbb{R}^n} & \quad f(\alpha) = \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\
\text{s.t.} & \quad y^T \alpha = 0 \quad \forall i = 1, 2, ..., n \\
& \quad 0 \leq \alpha_i \leq C, \quad \forall i = 1, 2, ..., n
\end{align*}
\]

\( e = [1, \ldots, 1]^T \), \( C \) is a positive constant used to tune the margin and the error, \( \alpha = (\alpha_1, \ldots, \alpha_n) \) are the Lagrange multipliers, \( Q \) is an \( n \times n \) symmetric matrix with \( Q_{ij} = y_i y_j K(x_i, x_j) \), and \( K(x_i, x_j) \) is the kernel function.

The support vectors (for which \( \alpha_i > 0 \)) are given by the optimal solution of (1), and then, the separating surface...
and the scalar $b$ are determined by the support vectors. The classification of a new data point $x$ is based on:

$$\text{sign} \left( \sum_{i=1}^{\#SV} y_i \alpha_i K(x, x_i) - b \right)$$

(2)

Variations on SVM algorithms use different classification functions. No algorithmic changes are required from the usual kernel function $K$ as a linear inner product other than the modification of the kernel evaluation, including a polynomial function of degree $d$, a RBF (Radial Basis Function) or a sigmoid function. We can get different support vector classification models.

PmSVM proposed by Wu [2] replaces the kernel function $K(x_i, x_j)$ in (1) and (2) with the power mean kernel $M_p(x_i, x_j)$, which is well-known as a general form of many additive kernels (e.g. $\chi^2$ kernel, histogram intersection kernel or Hellinger’s kernel).

$$M_p(x_i, x_j) = \sum_{z=1}^{d} (x_{i,z}^p + x_{j,z}^p)^{\frac{1}{p}}$$

where $p \in R$ is a constant.

PmSVM uses the coordinate descent method [26] for dealing with training tasks. Furthermore, the gradient computation step of the coordinate descent algorithm can be estimated approximately by polynomial regression with very low cost [2]. Therefore, PmSVM is very efficient in both training and testing tasks, compared to LIBLINEAR and other additive kernel SVMs.

IV. INCREMENTAL LEARNING FOR PmSVM

In this section, we present an incremental learning method for solving the memory usage problem of PmSVM. Our approach is inspired by the idea from the papers [27] and [28]. They show that training SVM classifier can be performed on the subsets of training set.

Let $\{B_j\}_{j=1}^m$ be a fixed partition of $T$ into $m$ blocks of rows. These blocks of rows are disjoint sets stored in $m$ separate files. At each iteration, we consider a block of rows $B_j$ and solve the problem (1) only for the samples in $B_j$. It means that the algorithm does not need to keep in memory the samples from other blocks of rows. According to memory size, we choose block size such that the samples in $B_j$ can fit into memory. Our approach is to use PmSVM to solve the sub-problems and the solution is updated in growing training data without loading the entire data into memory at once. The incremental learning for PmSVM is summarized in Algorithm 1.

Solving dual SVM by PmSVM for each block. The optimal solution of (1) can be obtained by solving the sub-

Algorithm 1: Incremental learning for PmSVM

input : A set of training samples $T = \{(x_i, y_i)\}_{i=1}^n$

output: The set of values $\{a_z\}$

1 Split $T$ into $B_1, ..., B_m$ and store data to $m$ files accordingly

2 $\alpha_i \leftarrow 0$, $1 \leq i \leq n$

3 $a_z, v \leftarrow 0$, $1 \leq z \leq d$, $0 \leq v \leq 2$

4 for $j \leftarrow 1$ to $m$

5 Read $x_r \in B_j$ from disk

6 Solve the sub-problem (4) by using PmSVM

7 Update $\alpha_i$ and $a_z$

8 end

problems (4).

$$\min_{d \in R^n} f(\alpha + d) = \frac{1}{2} (\alpha + d)^T Q (\alpha + d) - e^T (\alpha + d)$$

$$\text{s.t.} \left\{ \begin{array}{l} d_i = 0, \forall i \notin B_j \\ 0 \leq \alpha_i + d_i \leq C, \forall i \in B_j. \end{array} \right.$$ (4)

Let $d_B$ be a vector of $|B_j|$ non-zero coordinates of $d$ that correspond to the indices in $B_j$. The objective (4) is equivalent to

$$\frac{1}{2} d_B^T Q_{B_j} d_B + (Q_{B_j} \alpha - e_{B_j})^T d_{B_j},$$

where $Q_{B_j}$ is a sub-matrix of $Q$ including elements $Q_{i,r}, r \in B_j, i = 1, \ldots, n$. Obviously, $Q_{B_j}$ in Eq. 5 involves all training data. This violate the method presented in Algorithm 1. Consequently, we need to find the solution such that (5) can be solved by using only the samples in $B_j$.

For non-linear SVM, once we obtained the dual solution $\alpha$, the primal one is calculated by $w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$. Therefore, by maintaining $w$ into memory, the gradient $G$ which measures the changes in $f(\alpha + d)$ can be computed by using Eq. 6.

$$G = y_r w^T \phi(x_r) - 1 = y_r \sum_{i=1}^{n} \alpha_i y_i K(x_r, x_i) - 1.$$ (6)

We replace the kernel function in Eq. 6 by power mean kernel [2] and approximate $G$ by using Eq. 7.

$$G = y_r \sum_{z=1}^{2} \sum_{v=0}^{d} a_{z,v} (\ln(\alpha_{r,z} + 0.05))^v - 1.$$ (7)

Therefore, if $a_z$ is available in memory then only the samples associated with block of rows $B_j$ are needed to compute $G$. Suppose that $d_{B_j}^*\alpha$ is an optimal solution of (4). We can update $a_z$ by using Eq. 8.

$$a_z \leftarrow a_z + \sum_{r \in B_j} d_{r}^* y_r X^{-1} M_p(c, x_{r,z}).$$ (8)

Obviously, the update operation of $a_z$ in Eq. 8 again only involves the samples in $B_j$. The procedure for solving the sub-problem (4) is summarized in Algorithm 2.
V. EXTENSIONS OF INCREMENTAL PM SVM TO LARGE NUMBER OF CLASSES

Most SVM algorithms are only able to deal with a two-class problem. There are several extensions of a binary classification SVM solver to multi-class (k classes, \( k \geq 3 \)) classification tasks. The state-of-the-art multi-class SVMs are categorized into two types of approaches. The first one is to consider the multi-class case in an optimization problem \([29], [30]\). The second one is to decompose multi-class into a series of binary SVMs, including one-versus-all \([13]\), one-versus-one \([31]\) and Decision Directed Acyclic Graph \([32]\).

In practice, one-versus-all and one-versus-one are the most popular methods due to their simplicity. Let us consider k classes \((k > 2)\). The one-versus-all strategy builds k different classifiers where the \(i^{th}\) classifier separates the \(i^{th}\) class from the rest. The one-versus-one strategy constructs \(k(k - 1)/2\) classifiers, using all the binary pairwise combinations of the k classes. The class is then predicted with a majority vote.

When dealing with very large number of classes, e.g. hundreds classes, the one-versus-one strategy is too expensive because it needs to train many thousands classifiers. Therefore, the one-versus-all strategy becomes popular in this case. PmSVM algorithm also uses the one-versus-all approach to train independently \(k\) binary classifiers. However, the current PmSVM takes very long time to classify very large number of classes.

Due to this problem, we propose two ways for speedup learning tasks of PmSVM. The first one is to build balanced bagging classifiers with sampling strategy. The second one is to parallelize the training task of all the classifiers with several multi-core machines.

A. Balanced bagging PmSVM

In the one-versus-all approach, the learning task of PmSVM is to try to separate the \(i^{th}\) class (positive class) from the \(k - 1\) others classes (negative class). For very large number of classes, e.g. 1K classes, this leads to the extreme imbalance between the positive and the negative class. The problem is well-known as the class imbalance. As summarized by the review papers \([33], [34], [35]\) and the very comprehensive papers \([36], [37]\), solutions to the class imbalance problems were proposed both at the data and algorithmic level. At the data level, these algorithms change the class distribution, including over-sampling the minority class or under-sampling the majority class. At the algorithmic level, the solution is to re-balance the error rate by weighting each type of error with the corresponding cost. Our balanced bagging PmSVM belongs to the first approach. Furthermore, the class prior probabilities in this context are highly unequal (e.g. the distribution of the positive class is 0.1% in the 1K classes classification problem), and over-sampling the minority class is very expensive. We propose the balanced bagging PmSVM using under-sampling the majority class (negative class). For separating the \(i^{th}\) class (positive class) from the rest (negative class), the balanced bagging PmSVM trains T models as shown in algorithm 3.

Algorithm 3: Balanced bagging PmSVM

```
input : \(D_+\) the training data of the positive class
        \(D_-\) the training data of the negative class
        \(T\) the number of base learners
output : PmSVM model

1 Learn:
2   for \(k \leftarrow 1 \) to \( T \) do
3     1. \(D'_k = \text{sample}(D_-)\) (with \(|D'_k| = |D_+|\)
4     2. PmSVM\((D_+), D'_k\)
5 end
6 combine \(T\) models into the aggregated PmSVM model
```

We remark that the margin can be seen as the minimum distance between two convex hulls, \(H_+\) of the positive class and \(H_-\) of the negative class (the farthest distance between the two classes). Under-sampling the negative class \(\left(D'_k\right)\) done by balanced bagging provides the reduced convex hull of \(H_-\), called \(H'_-\). And then, the minimum distance between \(H_+\) and \(H'_-\) is larger than between \(H_+\) and \(H_-\) (full dataset). It is easier to achieve the largest margin than learning on the full dataset. Therefore, the training task of PmSVM is fast to converge to the solution. According to our experiments, by setting \(T = \sqrt{\frac{|D_+|}{|D_-|}}\), the balanced bagging PmSVM achieves good results in very fast training speed.

B. Parallel incremental PmSVM training

Although balanced bagging PmSVM deals with very large dataset with high speed, it does not take the benefits of HPC. Furthermore, balanced bagging PmSVM trains independently \(k\) binary classifiers for \(k\) classes problems. This is a nice property for parallel learning. Our investigation aims at speedup training tasks of multi-class balanced bagging PmSVM with several multi-processor computers. The idea is to learn \(k\) binary classifiers in parallel.

The parallel programming is currently based on two major models, Message Passing Interface (MPI) \([38]\) and Open
Multiprocessing (OpenMP) [39]. MPI is a standardized and portable message-passing mechanism for distributed memory systems. MPI remains the dominant model (high performance, scalability, and portability) used in high-performance computing today. However, MPI process loads the whole subset (block) into memory during learning tasks, making it wasteful. The simplest development of parallel PmSVM algorithms is based on the shared memory multiprocessing model OpenMP. However, OpenMP is not guaranteed to make the most efficient computing. Finally, we present a hybrid approach that combines the benefits from both OpenMP and MPI models. The parallel incremental learning for PmSVM is described in algorithm 4. The number of MPI processes depends on the memory capacity of the HPC system used.

Algorithm 4: Parallel incremental PmSVM training

```plaintext
input : A set of training samples \( T = \{ (x_i, y_i) \}_{i=1}^n \) 
\( P \) the number of MPI processes

output: The set of values \( \{ a_{2j} \} \)

1 Split \( T \) into \( B_1, \ldots, B_m \) and store data to \( m \) files accordingly
2 \( a^t_{1,1} \leftarrow 0 \), \( 1 \leq t \leq k \)
3 \( a^v_{2,1} \leftarrow 0 \), \( 1 \leq t \leq k \), \( 1 \leq z \leq d \), \( 0 \leq v \leq 2 \)
4 for \( j \leftarrow 1 \) to \( m \) do 
   Read \( x_r \in B_j \) from disk /* block \( j \) */
6 Learn:
7 \( MPI - PROC_1 \)
8 #pragma omp parallel for
9 for \( t_1 \leftarrow 1 \) to \( k_1 \) do /* class \( t_1 \) */
10   \( PmSVM (B_{t_1}^j, B_j \setminus B_{t_1}^j) \)
11   Update \( a^{t_1} \) and \( a^{t_1} \)
12 end
13 :
14 \( MPI - PROC_P \)
15 #pragma omp parallel for
16 for \( t_P \leftarrow 1 \) to \( k_P \) do /* class \( t_P \) */
17   \( PmSVM (B_{t_P}^P, B_j \setminus B_{t_P}^P) \)
18   Update \( a^{t_P} \) and \( a^{t_P} \)
19 end
20 end
```

VI. EXPERIMENTS

In this section, we compare our implementation with the original PmSVM and LIBLINEAR in terms of training time, memory usage, and classification accuracy. Our experiments are run on a cluster of ten computers Intel Xeon E5645 CPU with the same hardware architecture as shown in Table I. The cores in the same processor share one L2 cache and the main memory is shared among all the cores. All the computers are running Linux 2.6.32-5-amd64 (x86_64).

The extended versions of PmSVM are designed for large scale datasets, so we have evaluated our implementations on the two following datasets.

**ImageNet 100.** This dataset contains the 100 largest classes from ImageNet (183,116 images with data size 23.6GB). In each class, we sample 1000 images for training and 150 images for testing. We construct BoW histograms of images by using libHik [40] with SIFT descriptor [41], 1000 codewords and parameters “use both, grid step size 2 and split level 1”. Finally, the image is encoded as a 12000 dimensional vector. This encoding has been proven to give a good image classification performance with \( \chi^2 \) or Hellinger kernel SVM classifiers [40]. We end up with 10.5GB of training data.

**ILSVRC 1000.** This dataset contains 1K classes from ImageNet with 1.2M images (126GB) for training, 50K images (5.3GB) for validation and 150K images (16GB) for testing. To compare with the results reported in [2], we also use BoW feature set provided by [3] and the same method to encode every image as a vector in 21000 dimensions. We take \( \leq 900 \) images per class for training dataset, so the total training images is 887,816 and the training data size is 12.5GB. All testing samples are used to test SVM models.

A. Memory usage

According to the memory size of the computer used, we have split data into small blocks of rows that can fit into memory in each incremental step of PmSVM.

**ImageNet 100.** We have split this dataset into 3 and 6 blocks of rows. As shown in Table II, our implementation can run on computer with the main memory less than 4GB (iPmSVM-B-3) and less than 2GB (iPmSVM-B-6).

**ILSVRC 1000.** Due to the large size of the dataset, we have split this dataset into 10 and 40 blocks of rows, that allow training data to fit into 4GB RAM (iPmSVM-B-10) and 2GB RAM (iPmSVM-B-40) in each incremental step. As shown in Table III, LIBLINEAR and PmSVM consume a large amount of main memory (16.70GB and 21.13GB), making it intractable on computers with limited memory. On the other hand, by splitting data into many small blocks of rows, our approach is found to be very suitable for this case. For instance, iPmSVM-B-10 uses only 3.31GB RAM to train 1K classifiers of ILSVRC 1000. That means our implementation can save up to 84.34% memory usage as well. Furthermore, by setting the block size appropriately, the program does not need to swap parts of the blocks of rows between main memory and secondary memory (on the hard disk), as shown in Fig. 2.
Fig. 2. Memory usage (GB) of the balanced bagging incremental PmSVM (iPmSVM-B-10) on ILSVRC 1000.

### TABLE II

<table>
<thead>
<tr>
<th></th>
<th>LIBLINEAR</th>
<th>PmSVM</th>
<th>iPmSVM-B-3</th>
<th>iPmSVM-B-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory usage</td>
<td>11.00</td>
<td>13.75</td>
<td>3.75</td>
<td>1.80</td>
</tr>
</tbody>
</table>

Note that the training time increases if we split the data into blocks of rows with smaller size. It is because the classifiers need to load and train more blocks (Table IV, V).

### B. Training time

We have implemented two extended versions of PmSVM: 1) OpenMP balanced bagging incremental PmSVM (omp-iPmSVM-B), 2) Hybrid MPI/OpenMP balanced bagging incremental PmSVM (mpi-omp-iPmSVM-B). Incremental PmSVM is designed to handle data beyond the memory size, so the training time is considered at disk-level:

\[ \text{training time} = \text{user time to run data into memory} + \text{time to access data from disk}. \]

**ImageNet 100.** As shown in Fig. 3, on medium dataset ImageNet 100 our implementation shows a very good speedup in training process, compared to the original PmSVM and LIBLINEAR. By splitting the dataset into 3 blocks of rows and use 10 MPI process and 12 OpenMP threads per MPI process, our implementation (10mpi-omp-iPmSVM-B-3) is 55 times faster than the original PmSVM and 108 times faster than LIBLINEAR (Table IV).

**ILSVRC 1000.** Our implementations achieve a significant speedup in training process on this large dataset.

**Balanced bagging incremental PmSVM:** As shown in Table V, by splitting ILSVRC 1000 into 10 blocks of rows, the balanced bagging incremental PmSVM (omp-iPmSVM-B-10 running with 1 thread) has a very fast convergence speed in training process, it is 4 times faster than the original PmSVM.

**OpenMP balanced bagging incremental PmSVM:** By applying balanced bagging algorithm to OpenMP version of incremental PmSVM, we significantly speedup the training process of 1K binary classifiers. For instance, with the number of OpenMP threads set to 12, our implementation (omp-iPmSVM-B-10) is 31 times faster than the original PmSVM and 160 times faster than LIBLINEAR (Table V).

**Hybrid MPI/OpenMP balanced bagging incremental PmSVM:** Although OpenMP balanced bagging incremental PmSVM shows a significant speedup in training process, it does not ensure that the program achieves the most efficient high-performance computing on multi-core computers. Therefore, we explore this challenge by using a combination of MPI and OpenMP models. With this approach, our implementation achieves the impressive parallelization performance on a cluster of ten symmetric multiprocessor (SMP) nodes. The program first loads the whole block of data into nodes and each MPI process runs on one node. Therefore, one MPI process can work with their local data independently. However, we cannot increase the number of MPI processes exceed the memory capacity of a node. It is because each MPI process occupy the main memory during their computation process, resulting in a increase in the overall memory requirement. OpenMP has been proven to work effectively on shared memory systems. It is used for fine-grained parallelization within a node. Consequently, in each node we can increase the number of OpenMP threads without demanding more extra memory. In this experiment, we have set the maximum number of OpenMP threads equal to the number of cores available on a node. As shown in Fig. 4, our implementation (10mpi-omp-iPmSVM-B-10) achieves a significant speedup in training process by using 120 cores from ten SMP nodes (10 MPI processes × 12 OpenMP threads). It is 297 times faster than the original PmSVM and 1508 times faster than LIBLINEAR (Table V).

On ILSVRC 1000, we need only 2 minutes to train 1K binary classifiers, compared to the original PmSVM (∼10 hours) and LIBLINEAR (∼52 hours). This result confirms that our approach has a great ability to scaleup to full ImageNet dataset with more than 21K classes.
TABLE V
SMVs TRAINING TIME (MINUTE) ON ILSVRC 1000.

<table>
<thead>
<tr>
<th># OpenMP threads</th>
<th>1</th>
<th>6</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>3106.48</td>
<td>610.20</td>
<td></td>
</tr>
<tr>
<td>PmSVM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>omp-iPmSVM-B-10</td>
<td>144.57</td>
<td>33.25</td>
<td>19.44</td>
</tr>
<tr>
<td>omp-iPmSVM-B-40</td>
<td>148.54</td>
<td>35.07</td>
<td>20.74</td>
</tr>
<tr>
<td>5mpi-omp-iPmSVM-B-10</td>
<td>28.57</td>
<td>5.93</td>
<td>3.76</td>
</tr>
<tr>
<td>10mpi-omp-iPmSVM-B-10</td>
<td>16.53</td>
<td>3.14</td>
<td>2.06</td>
</tr>
</tbody>
</table>

Fig. 3. SVMs training time with respect to the number of threads on ImageNet 100.

Fig. 4. SVMs training time with respect to the number of threads on ILSVRC 1000.

TABLE VI
OVERALL CLASSIFICATION ACCURACY (%).

<table>
<thead>
<tr>
<th>Datasets</th>
<th>ImageNet 100</th>
<th>ILSVRC 1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>43.17</td>
<td>21.11</td>
</tr>
<tr>
<td>PmSVM</td>
<td>55.77</td>
<td>25.64</td>
</tr>
<tr>
<td>iPmSVM-B (2GB)</td>
<td>54.13</td>
<td>24.40</td>
</tr>
<tr>
<td>iPmSVM-B (4GB)</td>
<td>54.42</td>
<td>24.81</td>
</tr>
</tbody>
</table>

C. Classification accuracy

We have compared our balanced bagging incremental PmSVM with the original PmSVM and LIBLINEAR in terms of classification accuracy.

PmSVM. The original PmSVM with \( p = -1 \) (equivalent to \( \chi^2 \) kernel) and \( C = 0.01 \).

LIBLINEAR. The linear SVM from [4] with default parameter value \( C = 1 \).

iPmSVM-B. Balanced bagging incremental PmSVM with the same SVM parameters as the original PmSVM.

As shown in Table VI, in terms of classification accuracy PmSVM and iPmSVM-B outperform LIBLINEAR on medium dataset ImageNet 100 (from +10.96% to +12.60%, i.e. a relative increase of 26.05%) and very large dataset ILSVRC 1000 (from +3.70% to +4.53%, the relative improvement is more than 17.53%).

Note that iPmSVM-B runs much faster than PmSVM without (or very few) compromising classification accuracy.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have developed the extended versions of PmSVM in three ways: (1) An incremental learning for PmSVM, (2) A balanced bagging algorithm for training binary classifiers, (3) Parallelize the training process of these classifiers with several multi-core computers. Our approach was evaluated on the 100 largest classes of ImageNet and ILSVRC 1000. The experiment shows that our implementation can save up to 84.34% memory usage and the training process is 297 times faster than the original PmSVM and 1508 times faster than LIBLINEAR with 120 cores. We need only 2 minutes to train 1K binary classifiers. Furthermore, our approach can be easily applied to dataset larger than the memory capacity of computer. Obviously, this is a roadmap towards very large scale visual classification for systems with limited individual resource. The next step is to perform incremental PmSVM on 10K classes of ImageNet. With this large dataset, the training data would be much larger than the capacity of many existing HPC systems. This challenge will be addressed in the near future research.

REFERENCES


