Large Scale Support Vector Machines Algorithms for Visual Classification
Declaration of Authorship

I, Thanh-Nghi Doan, declare that this thesis titled, 'Large Scale Support Vector Machines Algorithms for Visual Classification' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.

- Where I have consulted the published work of others, this is always clearly attributed.

- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

- I have acknowledged all main sources of help.

- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ____________________________________________

Date: _______________________________________________
Abstract

Visual recognition remains an extremely challenging problem in computer vision research. Moreover, large datasets with millions of images for thousands of categories pose more challenges for the next generation of vision mechanisms, large scale visual classification. Learning an effective and efficient large scale visual classifier and constructing a robust visual representation are the most challenging issues. This dissertation aims to address these challenges with the following contributions.

Firstly, a lot of information is lost when performing the quantization step and thus the obtained bag-of-words (or bag-of-visual-words) have often not enough discriminative power for large scale visual classification. We propose a novel approach using several local descriptors simultaneously to improve the discriminative power of image representation.

Secondly, we extend the state-of-the-art large scale linear classifier LIBLINEAR SVM and nonlinear classifier Power Mean SVM in two ways. (1) The first one is to build the balanced bagging classifiers with sampling strategy. Our algorithm avoids training on the full data and the training process of classifiers rapidly converges to the optimal solution. (2) The second one is to parallelize the training process of all binary classifiers with several multi-core computers.

Thirdly, the new parallel multiclass stochastic gradient descent algorithm aims at classifying million images with very high-dimensional signatures into thousands of classes. We extend the binary stochastic gradient descent support vector machines (SVM-SGD) in several ways to develop the new multiclass SVM-SGD for efficiently classifying large image datasets into many classes. We propose: (1) a balanced training algorithm for learning binary SVM-SGD classifiers, (2) a parallel training process of all binary classifiers with several multi-core computers.

Finally, when the training data is larger (e.g. hundreds of giga-bytes) and cannot fit into main memory, the training task of SVM classifiers including linear and nonlinear kernels becomes more complicated to deal with. We address this challenge by extending both state-of-the-art large linear classifier LIBLINEAR-CDBLOCK and nonlinear classifier Power Mean SVM in these following ways: (1) an incremental learning method for Power Mean SVM, (3) a multi-class classification LIBLINEAR-CDBLOCK by using one-versus-all strategy, (3) a balanced bagging algorithm for training binary classifiers, (4) parallelize the training process of all binary classifiers with several multi-core computers. Our approaches have been evaluated on the 100 largest classes of ImageNet and ILSVRC 2010. The experiment shows that our approach can save up to 82.01% memory usage and the training process is much faster than the original implementation and the state-of-the-art linear classifier LIBLINEAR.

Keywords. Support vector machines, Incremental learning method, Stochastic gradient descent, Balanced bagging, Parallel algorithm, Large scale classification.
Acknowledgements

Over the past three years, I have been so happy to work at a great research laboratory with a fantastic supervisor and colleagues. I also had a great time when living in Rennes city, a beautiful city in the region of Brittany.

First and foremost, I wish to thank my supervisor Associate Professor at University of Rennes 1 François POULET for his continuous support and generous mentorship.

Many thanks to my wonderful collaborator Dr. Thanh-Nghi Do for sharing his ideas and freely discussion whenever I have a question.

I would like to thank Patrick GROS who has provided a perfect environment for me to pursue my PhD thesis in the TExMEx team-project.

I would like to thank Yann GUERMEUR, Florent PERRONNIN, Pierre GANÇARSK, David GROSS-AMBLARD, Vincent LEMAIRE for their attendances in my PhD committee members and for their useful comments on my thesis.

Besides, I am also grateful to all members of TExMEx team for their help and support.

This work is partially funded by Allocations de Recherches Doctorales (ARED), Région Bretagne, France.

Portions of this dissertation have resulted in the following papers:

- **Chapter 3 - Multi-feature and Multi-codebook**
  
  

- **Chapter 4 - Parallel Balanced Bagging Support Vector Machines**
  
  

- Thanh-Nghi Doan, Thanh-Nghi Do, and François Poulet. Parallel, Imbalanced Bagging Power Mean SVM for Large Scale Visual Classification. Submitted to *Transactions on Machine Learning and Data Mining*.

- Thanh-Nghi Doan and François Poulet. Algorithmes paralleles de SVMs pour la classification d’images. Submitted to *Traitement du Signal*.

- Chapter 6 - Parallel Incremental Support Vector Machines


- Thanh-Nghi Doan, Thanh-Nghi Do, and François Poulet. Big Learning with Parallel Imbalanced Incremental Multi-class LIBLINEAR SVM. Submitted to *Journal of Communication and Computer*.

Contents

Declaration of Authorship iii

Abstract v

Acknowledgements vii

List of Figures xiii

List of Tables xvii

1 Introduction 1
1.1 Visual recognition .............................................. 1
1.2 Challenges .................................................. 3
1.2.1 Image representations ..................................... 3
1.2.2 Machine learning algorithms ............................... 4
1.3 Thesis overview .............................................. 5
1.3.1 Contributions ............................................. 5
1.3.2 Outline .................................................. 6

2 State of The Art 7
2.1 The pipeline for visual classification .......................... 7
2.1.1 Extracting features ........................................ 7
2.1.2 Image representation ...................................... 13
2.1.3 Training classifiers ........................................ 17
2.2 Benchmark datasets in computer vision ...................... 18
2.3 Large scale visual classification .............................. 20
2.4 Online machine learning ..................................... 23

3 Multi-feature and Multi-codebook 27
3.1 Introduction ................................................ 27
3.2 Related work ................................................. 28
3.3 Multi-feature and multi-codebook ............................ 28
3.4 Experiment 1: (SIFT, SURF, DSIFT) + Feature map + LIBLINEAR ................................. 30
3.4.1 Datasets ............................................... 30
3.4.2 Parallel extracting feature ................................. 30
3.4.3 Codebook fast building .................................. 30
3.4.4 Parallel bag-of-packets constructing .................... 31
3.4.5 Classification accuracy .................................. 31
3.5 Experiment 2: (DSIFT, SOBEL, CENTRIST) + libHIK + PmSVM ............................. 32
4 Parallel Balanced Bagging Support Vector Machines
   4.1 Introduction ........................................... 37
   4.2 Related work ........................................... 38
   4.3 Support vector machines ................................. 39
      4.3.1 LIBLINEAR SVM ................................... 40
      4.3.2 Power mean SVM ................................... 40
   4.4 Improving SVM classifiers for large number of classes ... 41
      4.4.1 Balanced bagging SVM classifiers .................... 42
      4.4.2 Parallel SVMs training .............................. 43
   4.5 Experiment 3: the parallel versions of LIBLINEAR .......... 44
      4.5.1 Dataset ........................................... 45
      4.5.2 Training time ....................................... 45
      4.5.3 Classification accuracy .............................. 49
   4.6 Experiment 4: the parallel versions of PmSVM ............... 49
      4.6.1 Dataset ........................................... 49
      4.6.2 Training time ....................................... 50
      4.6.3 Classification accuracy .............................. 52
   4.7 Conclusion ............................................. 53

5 Parallel Stochastic Gradient Descent Algorithms Support Vector Machines 55
   5.1 Introduction ........................................... 55
   5.2 Related Work ........................................... 56
   5.3 SVM with stochastic gradient descent ....................... 57
   5.4 Extensions of SVM-SGD to large number of classes ........... 58
      5.4.1 Balanced training SVM-SGD .......................... 58
      5.4.2 Parallel multi-class SVM-SGD training ............... 60
   5.5 Experiment 5: the parallel version of SVM-SGD ............. 61
      5.5.1 Dataset ........................................... 62
      5.5.2 Training time ....................................... 62
      5.5.3 Classification accuracy .............................. 65
   5.6 Conclusion ............................................. 66

6 Parallel Incremental Support Vector Machines ................................ 69
   6.1 Introduction ........................................... 69
   6.2 Related Work ........................................... 71
   6.3 Incremental learning for SVM classifiers ..................... 72
      6.3.1 Solving dual SVM by LIBLINEAR for each block .......... 72
      6.3.2 Solving dual SVM by PmSVM for each block ............. 74
   6.4 Improving incremental SVM classifiers for large number of classes ... 74
      6.4.1 Balanced bagging incremental SVM (LIBLINEAR, PmSVM) .... 75
      6.4.2 Parallel incremental SVM training ..................... 77
   6.5 Experiment 6: the parallel incremental LIBLINEAR ........... 77
      6.5.1 Datasets ........................................... 78
      6.5.2 Memory usage ....................................... 78
      6.5.3 Training time ....................................... 81
      6.5.4 Classification accuracy .............................. 83
   6.6 Experiment 7: the parallel incremental PmSVM ............... 84
      6.6.1 Dataset ........................................... 85
      6.6.2 Memory usage ....................................... 85
List of Figures

1.1 Sample images from the PASCAL Visual Object Classes Challenge 2012 [1]. The task of image categorization systems is to distinguish such images from other photographs. .................................................. 2

2.1 The overview of the usual pipeline for visual classification task. .................. 8

2.2 Sampling of interesting points: the candidate interesting key-points are selected from either a regular grid (a) or interest regions (b). Image courtesy of James Hays (2011). .................................................. 9

2.3 Extracting image feature descriptors: each sampled local patch is transformed into a reduced representation set of features (also named features vector or descriptor), and thus each image is represented by a collection of descriptors. Image courtesy of Josef Sivic. .................................................. 9

2.4 A schematic representation of scale invariant feature transform (SIFT): The gradient orientations and magnitudes are computed at each pixel in a region around the detected key-point and weighted by a Gaussian fall-off function (blue circle). A weighted gradient orientation histograms are then computed over $4 \times 4$ subregions, using trilinear interpolation. This figure shows an $8 \times 8$ pixel patch and a $2 \times 2$ array of orientation histograms, whereas Lowe's actual experiments show that the best results are achieved by using $16 \times 16$ patches and $4 \times 4$ array of eight-bin histograms. Image courtesy of David Lowe (2004). .................. 10

2.5 Illustration of SURF descriptor: Left - Detected interest key-points for Sunflower field photo, Haar-wavelet responses are calculated in $x$ and $y$ direction, this shows the nature of the features from Hessian-nase detectors. Middle - Haar-wavelet types used for SURF. Right - Detail of the Graffiti scene showing the size of the descriptor window at different scale. Image courtesy of Herbert Bay (2008). ........ 11

2.6 Illustration of CENTRIST descriptor: (a) An example image from the 15 class scene recognition dataset, (b) A Census Transformed (CT) image is created by replacing each pixel with its CT value. The Census Transform retains global structure of the picture besides capturing the local structures. Image courtesy of Jianxin Wu (2013). .................................................. 12

2.7 Sobel convolution kernels. .................................................. 12

2.8 “Bag-of-Visual-Words” image representation: an image is abstracted by an unordered set of several local patches, each patch is assigned to one of 4 visual-words, and the frequencies of “visual words” is used to represent the image. Image courtesy of Fei-Fei Li. .................................................. 14

2.9 Illustration of building a visual codebook: (a) the local image descriptors of training images are vector-quantized by an unsupervised clustering algorithm, (b) the obtained center point of each cluster is considered as a visual-word (or code-word) of the visual codebook. Image courtesy of Josef Sivic. .................................................. 15

2.10 Two image examples from the same object class might have different BoW and SPM models (first two rows on the center). The proposed self-similarity hyper-cubes (SSH) model observes the concurrent occurrences of visual words and thus it is able to describe the structural information of BoW in an image. Image courtesy of Chih-Fan Chen. .................................................. 16
2.11 Effects of vector quantization. Informative descriptors have low frequency in database, leading to high quantization error. (a) An image from Face class in Caltech 101, (b) Quantization error of densely computed SIFT descriptors using a codebook with 6,000 visual words (red = high error; blue = low error). The most informative patches (eye, nose, etc.) have the highest quantization error. (c) The 8% of the descriptors in the image being most frequent in the database (simple edges) are indicated by green marks. (d) Magenta masks the 8% of the descriptors in the image that are least frequent in the database, mostly discriminative facial features. Image courtesy of Oren Boiman.

2.12 A comparison of ImageNet with other benchmark datasets.

2.13 Batch learning: Given a training data (left). (1) The learning algorithm (center) takes the whole data as input from the beginning. This consumes a lot memory and computation power. (2) After learning, the output model can be used to predict testing data.

2.14 Online learning: Given a training data (left). (1) At each incremental step, the algorithm draws out a sample from training data. Thus, the requirements of memory and computation are low. (2) The learning algorithm predicts the label of the example. (3) The example is removed from training data after the algorithm learned from it. (4) Anytime during the training process, one may update the current classifier when needed (but not preferred).

2.15 An extension to online learning: Given a training data (left). (1) The algorithm splits the training data into many blocks of rows and then each block is loaded into memory at a time (orange square). (2) After a learning step on the block, all samples in it are removed from the training data and flushed out of memory. Stage (2) and (4) are similar to traditional online learning algorithms.

3.1 The high intraclass variability of images in the same class of ImageNet.

3.2 Construct bag-of-packets based on multi-feature and multi-codebook approach.

3.3 ImageNet 10, overall classification accuracy (%) with LIBLINEAR.

3.4 ImageNet 100, overall classification accuracy (%) with LIBLINEAR.

3.5 ImageNet 10, overall classification accuracy (%) with PmSVM.

3.6 ImageNet 100, overall classification accuracy (%) with PmSVM.

3.7 The high intraclass variability of images in the same class of ImageNet.

4.1 Linear separation of the datapoints into two classes.

4.2 The gradient computation of PmSVM is approximated by using polynomial regression. Image courtesy of Jianxin Wu (2012).

4.3 Undersampling without replacement for SVM training.

4.4 Linear SVMs training time with respect to the number of OpenMP threads on ImageNet 100.

4.5 Linear SVMs training time with respect to the number of OpenMP threads on ILSVRC 2010.

4.6 SVMs training time with respect to the number of OpenMP threads on ImageNet 100.

4.7 SVMs training time with respect to the number of OpenMP threads on ILSVRC 2010.

4.8 Overall classification accuracy of SVM classifiers.

5.1 SVMs training time with respect to # OpenMP threads on ImageNet 10.

5.2 SVMs training time with respect to # OpenMP threads on ImageNet 100.

5.3 SVMs training time with respect to # OpenMP threads on ILSVRC 2010.

5.4 Overall classification accuracy of SGD SVM classifiers.

6.1 Balanced bagging algorithm for incremental SVM.

6.2 Memory usage (GB) of LIBLINEAR-B-8 on ILSVRC 2010.

6.3 Memory usage (%) of linear incremental SVMs.
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4</td>
<td>Saved memory (%) of linear incremental SVMs on ILSVRC 2010.</td>
<td>80</td>
</tr>
<tr>
<td>6.5</td>
<td>Linear incremental SVMs training time with respect to the number of OpenMP threads on ImageNet 100.</td>
<td>81</td>
</tr>
<tr>
<td>6.6</td>
<td>Linear incremental SVMs training time with respect to the number of OpenMP threads on ILSVRC 2010.</td>
<td>83</td>
</tr>
<tr>
<td>6.7</td>
<td>Overall classification accuracy of linear incremental SVM classifiers.</td>
<td>84</td>
</tr>
<tr>
<td>6.8</td>
<td>Memory usage (GB) of PmSVM-B-8 on ILSVRC 2010.</td>
<td>86</td>
</tr>
<tr>
<td>6.9</td>
<td>Saved memory (%) of SVMs on ILSVRC 2010.</td>
<td>86</td>
</tr>
<tr>
<td>6.10</td>
<td>Incremental SVMs training time with respect to the number of OpenMP threads on ImageNet 100.</td>
<td>87</td>
</tr>
<tr>
<td>6.11</td>
<td>Incremental SVMs training time with respect to the number of OpenMP threads on ILSVRC 2010.</td>
<td>89</td>
</tr>
<tr>
<td>6.12</td>
<td>Overall classification accuracy of incremental SVM classifiers.</td>
<td>90</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Illustration of benchmark datasets in computer vision. ImageNet'12 is much larger in terms of both the number of classes and the number of images, and more diversity than other benchmark datasets. ................................................. 21
2.2 The pros and cons of SVM classifiers for visual classification tasks. ................. 23
3.1 Parallelize bag-of-packets construction of training images on ImageNet 10 (8 cores). The image signature is converted to high-dimensional space by using feature map. ............................................................. 32
3.2 Parallelize bag-of-packets construction of training images on ImageNet 100 (8 cores). The image signature is converted to high-dimensional space by using feature map. .................................................. 32
3.3 Multiple features, overall classification accuracy (%) with LIBLINEAR. ............ 32
3.4 Parallelize bag-of-packets construction of training images on ImageNet 10 (160 cores). The image signature is constructed by using libHIK. .......................... 34
3.5 Parallelize bag-of-packets construction of training images on ImageNet 100 (160 cores). The image signature is constructed by using libHIK. ................. 34
3.6 Multiple features, overall classification accuracy (%) with PmSVM. ................. 34
4.1 The physical features of a computer of multi-core system. ............................... 45
4.2 Linear SVMs training time (minutes) on ImageNet 100. .............................. 46
4.3 Linear SVMs training time (minutes) on ILSVRC 2010. .................. 48
4.4 Linear SVMs overall classification accuracy (%). ................................. 49
4.5 PmSVMs training time (minutes) on ImageNet 100. .............................. 50
4.6 PmSVMs training time (minutes) on ILSVRC 2010. .............................. 52
4.7 SVMs overall classification accuracy (%). ........................................... 53
5.1 SVMs training time (minutes) on ImageNet 10. ....................................... 63
5.2 SVMs training time (minutes) on ImageNet 100. ....................................... 63
5.3 SVMs training time (minutes) on ILSVRC 2010. ....................................... 65
5.4 SVMs overall classification accuracy (%). ........................................... 66
6.1 Memory usage (GB) of linear incremental SVM classifiers on ImageNet 100. ........ 79
6.2 Memory usage (GB) of linear incremental SVM classifiers on ILSVRC 2010. .... 79
6.3 On ImageNet 100, LIBLINEAR-CDBLOCK-B-3 and LIBLINEAR-B-3 maintain into memory a similar size of matrix. However, on ILSVRC 2010, LIBLINEAR-CDBLOCK-B-8 maintains into memory a matrix with much larger size than that of LIBLINEAR-B-8. ................................................. 80
6.4 Linear incremental SVMs training time (minute) on ImageNet 100. ............... 81
6.5 Linear incremental SVMs training time (minute) on ILSVRC 2010. ............... 83
6.6 Overall classification accuracy (%) of linear incremental SVMs. .................. 84
6.7 Memory usage (GB) of SVM classifiers on ImageNet 100. ....................... 85
6.8 Memory usage (GB) of SVM classifiers on ILSVRC 2010. ....................... 86
6.9 Incremental SVMs training time (minute) on ImageNet 100. ....................... 87
6.10 Incremental SVMs training time (minute) on ILSVRC 2010. ....................... 88
6.11 Overall classification accuracy (%). ........................................... 89
To my parents and my wife
Chapter 1

Introduction

1.1 Visual recognition

Visual recognition is one of the important research topics in computer vision and machine learning. The ultimate goal is to ask a computer to perform, analyzing a scene, recognizing all of the constituent objects in an image, and the spatial function relations between them. This task, however, remains the most challenging in computer vision. The difficulty comes from the ample objects in the real world, which all or partly occlude another one and appear in different poses. Moreover, the high intrinsic variability within a class makes the recognition problem more difficult to deal with. Therefore, the recognition problem can be broken down into several manageable problems. For example, object detection, which is the problem of determining whether a query object appears in an image. If we have a specific rigid object we are trying to recognize (instance recognition), we can search for characteristic feature points and verify that they align in a geometrically plausible way; Image segmentation, which is the process of partitioning a digital image into multiple segments (sets of pixels, also known as superpixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics.

The most challenging version of visual recognition is general category recognition, which is related to recognize instances of extremely varied classes such as cars or bikes. Consider a set of sample images in Figure 1.1, which shows the objects taken from 20 different categories. The task of visual classification is to categorize each of these images into an appropriate class. This is the standard multiclass classification problem in machine learning. Often the task is also called image categorization or object categorization.
Image categorization has received much attention from researchers during the past few decades. However, it still remains a very challenging problem and calls for more efficient and effective methods due to these main reasons:

- **Human performance.** Biederman [2] reports that the semantic space of human brain can be used to describe more than 30 thousand visual categories. Obviously, there still exists a big gap between the most efficient vision mechanisms and human performance.

- **Enormous explosion of visual data.** The popularity of digital camera devices, smartphones and the online photo-sharing services have made raw image data rapidly increase to a huge number of instances over the past few years. For visual data, the latest estimations report that there are 6 billion photos indexed by Flickr [3] (3500 uploaded/second) and more than 10 billion photos indexed by Google.
Image Search [4] and these photos have more and more pixels meaning larger and larger dataset size (the new CMOS sensors are 41 Mp).

- **Large scale visual object dataset.** One of the most crucial component in machine learning and computer vision is visual object datasets. Currently, many enthusiastic researchers are focusing on constructing large scale well-annotated datasets and make them available for public users. For example, many large scale publicly datasets such as LabelMe [5], TinyImage [6] and ImageNet [7] are growing with much further improvements in every year, and they play a very important role for developing large scale visual recognition algorithms.

- **Scalability of existing algorithms.** Most previous works on visual classification have been evaluated only on small datasets with dozens or hundreds of categories, such as Caltech 101 [8], Caltech 256 [9], PASCAL VOC [10], etc. Although, a number of proposed methods have obtained the impressive results in terms of accuracy performance, most of them do not scale well on large datasets with many classes.

The emergence of ImageNet with millions images in thousands categories makes the existing approaches intractable and thus poses more challenges for the next generation of visual classification systems. Some of these challenges have been analyzed in the paper [11], so we do not plan to re-write it. The next section will present only the challenges that the dissertation aims to address.

## 1.2 Challenges

In this dissertation, we are interested in tackling two main research challenges that most state-of-the-art visual classification systems are facing when dealing with large scale datasets: *image representations* and *machine learning algorithms*.

### 1.2.1 Image representations

During the past decade, the proposed methods for image representations have mainly relied on *bag-of-visual-words* model [12]. Most of these methods use a certain low-level feature to represent an image, e.g. SIFT [13] descriptor. However, if one feature offers very good results on one dataset, it does not guarantee that the feature will achieve the similar results on other datasets. This figures out the fact that the performance of a visual recognition system is very sensitive to the feature we choose. For instance, the features based on texture information might perform well when classifying object class walls. On the other hand, a classifier for zebras should be invariant to the texture of the zebras. Therefore, instead of using an individual feature type for all classes it should be better to use simultaneously multiple features such as shape, color, texture, keypoint-based features, etc. However, this raises the question of how to combine these features in order to create the most efficient image representation, especially for the case
of large datasets with thousands categories. This problem still remains an open challenge in computer vision community.

1.2.2 Machine learning algorithms

*Machine learning* is about the problem of how to write computer programs that can automatically learn from data. The algorithm based on machine learning is called *machine learning algorithm* (or *learning algorithm*). In this dissertation we only consider *supervised learning* problems. In such problems, the task of supervised learning algorithm is to learn a general model by using a set of samples, called a *training set*. Each sample consists of a pair of instance and its corresponding label. After learning, the obtained model can be used to predict the label of new samples or *testing set*. Most previous approaches on vision problems have focused on support vector machines algorithms (SVM [14]) for learning models. However, for the case of large scale visual classification tasks, these approaches are facing with the following challenges:

- **Large scale learning classification model.** Support vector machines are one of the most frequently used classification models due to delivering the state-of-the-art performances in real world visual recognition and data mining. The previous approaches can choose either linear or nonlinear model because they were learning on small datasets. However, for the case of large datasets, the cost of learning nonlinear classification models is too expensive or prohibitive. Thus, most researchers have focused on training linear classifiers due to their efficiency in training and testing. Unfortunately, linear classifiers are inferior in terms of classification accuracy when compared to their nonlinear counterparts [15–17]. Consequently, many novel algorithms have been proposed to bridge the gap between the training time of nonlinear classifiers and linear classifiers. The recent papers [18–21] propose a class of additive kernel SVMs that use a few times more training time, compared to the state-of-the-art linear SVM solvers. In some large vision problems, additive kernel SVMs are even faster than linear SVMs, making them more practical for large scale visual classification tasks.

- **Memory requirement.** Most traditional SVM training methods are designed by assuming that training data can be stored in the computer main memory. However, with millions of training examples or millions of feature dimensions, these methods encounter a problem because training data is larger and cannot fit into memory any more. The first works were Fung et al. [22] and Poulet et al. [23], who used an incremental and an incremental parallel SVM algorithm to perform the classification. Very recently, Yu et al. [24] propose a block minimization framework for large linear SVMs in order to handle the data beyond the memory capacity of computer. The evaluation shows that their method can effectively train classifiers when training data is 20 times larger than memory size. However, for multi-class classification problem, it solves one single optimization problem by using [25] instead of *one-versus-all* strategy. Thus, in the context of large datasets with very
large number of images as well as classes, their method still requires a large amount of memory, making their method less useful in real world applications. Furthermore, as aforementioned, for visual classification tasks, nonlinear SVM classifiers often have consistently higher accuracy rates than linear rivalries. Therefore, the question of how to design the large scale linear and nonlinear SVM classifiers that satisfy both two requirements: i) fast training and accurate testing, ii) they can be trained on the computers/grid with limited individual memory resource, is still very challenging. This calls more effective and scalable algorithms.

- **Time consumption.** The final challenge of training SVM models is the time consumption of learning process. For very large datasets with millions of images, there can be so many examples that it can be too expensive to even go through the data once. Thus, training an accurate SVM classifier may take weeks or even years because the complexity of algorithm is supper-linear with the number of samples. Although the state-of-the-art of linear and nonlinear SVM classifiers have many improvements, the training process is still slow. In multi-core era, the platforms with several multi-core computers/grid are becoming ubiquitous and affordable. Furthermore, the advanced technologies designed for the systems where several processes have access to shared or distributed memory space have demonstrated their effectiveness in many scalable and high performance computing applications. This forces researchers study novel methods to develop distributed visual learning algorithms that can scaleup to hundreds or thousands nodes in the cloud computing platforms.

Obviously, the discussed challenges make large scale visual classification a very important and interesting problem. Hence, it would not to be so surprising if there are more and more researchers presenting their finest efforts on how to bridge the gap between vision mechanisms and human performance. The next section will present the major contributions of this dissertation.

### 1.3 Thesis overview

#### 1.3.1 Contributions

Regarding to the first challenge, we have proposed a novel *multi-feature and multi-codebook* approach that uses several low-level local image feature types simultaneously. By combining these features we can improve the dissimilarity power of image representations and thus obtain higher accuracy rate than a single feature type. Our approach is simple, yet it is effective and appropriate for large scale datasets.

For the second challenge, we have proposed several ways to improve both state-of-the-art large scale linear and nonlinear SVM classifiers for visual classification tasks.
• To solve the memory usage of large scale nonlinear classifier PmSVM [21], we propose an incremental learning method for binary classifiers. Our approach avoids loading the whole training data into memory by splitting it into small blocks of rows stored in separate files accordingly. Then, at any one time, a block of rows will be loaded into memory when needed in each incremental step of binary classifiers. For the case of linear classifier, we improve the block minimization framework for large linear SVM (LIBLINEAR-CDBLOCK [24]) by exploiting one-versus-all strategy. By the way, we can easily train large scale SVM classifiers including linear and nonlinear kernel versions on large datasets with very large number of classes and training data larger than the memory capacity of computer.

• To speedup the training process of classifiers, we propose two effective ways: i) a balanced bagging algorithm for dealing with the training task of the binary classifiers, our algorithm avoids learning on full training data and thus the training process of classifiers is fast to converge to the optimal solution; ii) a parallel learning algorithm for these classifiers based on HPC models. Therefore, our algorithm allows training SVM classifiers on several multi-core computers with limited individual memory resource.

For nonlinear classifiers, the standard SVM algorithms that solve the primal or dual optimization of SVM have similar time complexity [26]. However, in the case of large datasets with a huge number of examples, the primal optimization of linear SVMs is definitely superior [27]. This motivates us to extend the binary SVM-SGD [28] in several ways to develop the new parallel multi-class SVM-SGD algorithms for efficiently classifying large datasets into many classes. We have made two contributions: i) a balanced training algorithm for training binary SVM-SGD classifiers, our algorithm simultaneously use both two approaches at data level and algorithm level; ii) parallelize the training process of these classifiers with several computers/grid.

1.3.2 Outline

This dissertation is organized as follows. In Chapter 2, a description of the pipeline for visual classification is presented with details. We introduce the background knowledge of main components of the pipeline and their improvements in recent years. The state-of-the-art large scale classification and the related works are discussed in order to understand the main matters of dissertation. Chapter 3 presents the multi-feature and multi-codebook approach. The parallel balanced bagging algorithm for training SVM classifiers is presented in Chapter 4. Chapter 5 gives the detailed algorithms of parallel stochastic gradient descent SVM for large datasets. We describe the incremental learning algorithm for large scale SVM classifiers in Chapter 6. Finally, we conclude the dissertation and point to the area of future works in Chapter 7.
Chapter 2

State of The Art

In this chapter we introduce more details about a visual classification system that provide the necessary background knowledge and the state-of-the-art of main components of the system. This chapter is structured as follows. Section 2.1 presents the usual pipeline for visual classification task. The benchmark datasets in computer vision are introduced in Section 2.2. Section 2.3 presents the related work on large scale visual classification. In Section 2.4 we discuss about the advantages and drawbacks of online and offline machine learning methods in a large scale setup.

2.1 The pipeline for visual classification

Low-level local image features, bag-of-visual-words model and support vector machines are the core of state-of-the-art visual classification systems. The usual pipeline for visual classification task, as depicted in Figure 2.1, involves three following stages: 1) extracting features, 2) encoding images (or image representation), and 3) training classifiers.

2.1.1 Extracting features

As shown in Figure 2.1, given a set of input images, the system first extracts low-level local image features. In general, extracting features from images consists of three main steps: 1) searching the candidate interesting points (or key-points), 2) selecting the key-points, and 3) extracting the key-point descriptors.

In step 1, there are two different types of approaches to obtain such a set of interesting points. The first approach is based on key-points detection, where you may notice only some specific locations in the image such as mountain peaks, building corners, doorways, or interestingly shaped patches of bears, as shown in Figure 2.2 (b). These kinds of localized features are often called key-point features or interesting points and are often described by the appearance of patches of pixels surrounding the point location.
Therefore, the research question is how to find these localized features where we can reliably find the correspondences with other images, i.e. what are good features to track? [29, 30]. These interesting points can be identified by using the detectors such as the classic “Harris” detector [31] which detects the corners and edges of the objects in an image, and another choice is blobs [32]. “Harris-Laplace” detector, a more modern detector is proposed by [33], which simultaneously adapts location, scale and shape of a point neighborhood to obtain affine invariant points. These methods are developed for stereo-matching, which find the best matches between similar key-points in two different images. The second approach is based on dense sampling of points from the image [34]. As shown in Figure 2.2 (a), all points on a regular dense grid in different scales are used as a key-points. The main reason to use dense sampling is to avoid early removal of candidate interesting points. Conceivably, if the image is described by points from a dense grid over all possible locations, the whole image can be reconstructed from the set of selected points, and thus less information is lost. Therefore, dense sampling approach has become the de-facto standard for image classification task [35–37]. Experiment results

Figure 2.1: The overview of the usual pipeline for visual classification task.
Figure 2.2: Sampling of interesting points: the candidate interesting keypoints are selected from either a regular grid (a) or interest regions (b). Image courtesy of James Hays (2011).

Figure 2.3: Extracting image feature descriptors: each sampled local patch is transformed into a reduced representation set of features (also named features vector or descriptor), and thus each image is represented by a collection of descriptors. Image courtesy of Josef Sivic.

demonstrate that the performance increases according to the number of regions sampled from images [38, 39].

In step 2, depending on a particular computer vision task, all keypoints of the image or only the keypoints with high contrast or rich localization are selected for the next step.

In step 3, after key-points detection and selection step, the sampled local patches (around the key-points) are extracted in the image. Each patch is described by a feature vector (or descriptor), and therefore we obtain a collection of feature descriptors from the image, as shown in Figure 2.3.

To date, there are a variety of low-level image features proposed in the literature. Depending on a specific case study, the user chooses a suitable feature for their application. In this dissertation we use the following features SIFT [13], SURF [40], dense SIFT (DSIFT) [35], CENTRIST [41] and Sobel [42] for our experiments. These image features have been proven to be efficient in a range of computer vision tasks, such as object recognition, texture analysis, scene classification, etc.
Figure 2.4: A schematic representation of scale invariant feature transform (SIFT): The gradient orientations and magnitudes are computed at each pixel in a region around the detected key-point and weighted by a Gaussian fall-off function (blue circle). A weighted gradient orientation histograms are then computed over $4 \times 4$ subregions, using trilinear interpolation. This figure shows an $8 \times 8$ pixel patch and a $2 \times 2$ array of orientation histograms, whereas Lowe’s actual experiments show that the best results are achieved by using $16 \times 16$ patches and $4 \times 4$ array of eight-bin histograms. Image courtesy of David Lowe (2004).

SIFT (Scale Invariant Feature Transform)

SIFT proposed by [13] is one of the most widely used algorithms to detect and describe local features in images, as illustrated in Figure 2.4. Extracting SIFT descriptors consists of four key stages: scale-space extrema detection, key-point localization, orientation assignment and key-point descriptor.

The first stage is to use Difference-of-Gaussian function (DoG) to identify the candidate interest points that are invariant to scale and orientation of the image. DoG is used instead of Gaussian to speedup the computation. In the key-point localization stage, the candidate points are removed if they are low contrast or poorly localized along an edge. Hessian matrix is used to compute the principal curvatures and eliminate the key-points that have a ratio between the principal curvatures greater than the threshold. An orientation histogram is formed from the gradient orientations of sample points within a region around the key-point in order to get an orientation assignment. According to the paper’s experiments, the best results are achieved with a $4 \times 4$ array of histograms with 8 orientation bins in each. So the SIFT descriptor used is $4 \times 4 \times 8 = 128$ dimensions.

SURF (Speeded Up Robust Feature)

SURF is a robust image detector and descriptor presented by [40], as shown in Figure 2.5. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT. SURF is based on sums of 2D Haar wavelet responses and makes an efficient use of integral images. SURF is partly inspired by the SIFT descriptor and has slightly different ways of detecting features. It uses an integer approximation to the determinant of Hessian blob detector, which can be computed extremely quickly with an integral image. For features,
it uses the sum of the Haar wavelet response around the point of interest. Again, these can be computed with the aid of the integral image.

**DSIFT (Dense SIFT)**

A variant of SIFT descriptors that is extracted at multiple scales is proposed by [35]. It is roughly equivalent to running SIFT on a dense grid of locations at a fixed scale and orientation. This type of feature descriptors are often used for object categorization.

- **Bin size vs keypoint scale.** DSIFT specifies the descriptor size by a single parameter, size, which controls the size of a SIFT spatial bin in pixels. In the standard SIFT descriptor, the bin size is related to the SIFT keypoint scale by a multiplier, denoted magnif, which defaults to 3. As a consequence, a DSIFT descriptor with bin size equal to 5 corresponds to a SIFT keypoint of scale $\frac{5}{3}=1.66$.

- **Smoothing.** The SIFT descriptor smoothes the image according to the scale of the keypoints (Gaussian scale space). By default, the smoothing is equivalent to a convolution by a Gaussian of variance $s^2$ where $s$ is the scale of the keypoint and 0.25 is a nominal adjustment that accounts for the smoothing induced by the camera CCD.

**CENTRIST (CENsus TRansform hISTogram)**

CENTRIST is a visual descriptor for place and scene category recognition task proposed by [41]. CENTRIST is a *holistic* representation that captures structural properties by modeling distribution of local structures, as shown in Figure 2.6. It can capture rough geometrical information by using a spatial CENTRIST representation. CENTRIST also has similar descriptor vectors for images in the same place category. CENTRIST descriptor is created by computing the histogram of Census Transform (CT) values for
Chapter 2. State of The Art

Figure 2.6: Illustration of CENTRIST descriptor: (a) An example image from the 15 class scene recognition dataset, (b) A Census Transformed (CT) image is created by replacing each pixel with its CT value. The Census Transform retains global structure of the picture besides capturing the local structures. Image courtesy of Jianxin Wu (2013).

![Mountain Image](image1.png) ![Transformed Image](image2.png)

Figure 2.6: Illustration of CENTRIST descriptor: (a) An example image from the 15 class scene recognition dataset, (b) A Census Transformed (CT) image is created by replacing each pixel with its CT value. The Census Transform retains global structure of the picture besides capturing the local structures. Image courtesy of Jianxin Wu (2013).

Figure 2.7: Sobel convolution kernels.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>0</td>
<td>+2</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td></td>
</tr>
</tbody>
</table>

Gx

<table>
<thead>
<tr>
<th></th>
<th>+1</th>
<th>+2</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

Gy

the image or image patch. It can be computed very efficiently due to involving only 16 operations to compute the CT value for a center pixel.

**Sobel**

Sobel operator is used in image processing, particularly within edge detection algorithms [42], as illustrated in Figure 2.7. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. The Sobel operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical direction and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, in particular for high frequency variations in the image.
2.1.2 Image representation

In this section we present image representation used for image classification tasks with a special emphasis on the bag-of-visual-words (BoW, also known as bag-of-features or bag-of-keypoints) approach. Two main nontrivial steps of BoW approach, building a visual codebook and encoding local features, are also described and discussed in details.

Bag-of-visual-words

The BoW approach is a simplifying representation used in natural language processing and information retrieval. In this model, a text (such as a sentence or a document) is represented as an unordered collection of words, disregarding grammar and even word order. The BoW approach is commonly used in methods of document classification, where the (frequency of) occurrence of each word is used as a feature for training a classifier. The early reference to “bag of words” in a linguistic context can be found in [43]. In text document retrieval, Salton et al. [44, 45] present a Vector-Space-Model for automatic indexing. They propose an approach based on space density computations to choose an optimum indexing vocabulary for collection of documents. A vector representation of document space is created by computing the frequency of occurrences of a word in a document. This vector neglects the structure of a document, which is known as a BoW representation, so it is invariant with the word order in a document. Their evaluation results have demonstrated the usefulness of the model. Many recent works show that the BoW approach will continue to be successful in text retrieval and classification applications [46, 47].

In computer vision, the BoW approach can be applied to image retrieval and classification, by treating image features as words. As shown in Figure 2.8, this algorithm simply computes the distribution (signatures or histograms) of visual words found in the query image and compares this distribution to those found in the training images. Sivic et al. [48] is the first to introduce this approach for image retrieval. The object is represented by a set of viewpoint invariant region descriptors so that recognition can proceed successfully despite changes in viewpoint, illumination and partial occlusion. Csurka et al. [49] uses the term bag-of-keypoints to describe such approach and demonstrate the utility of frequency-based techniques for visual categorization. This method is based on vector quantization of affine invariant descriptors of image patches. By using naïve Bayesian classifier or support vector machines for classification, they show that the method is robust to background clutter and produces good accuracy performance even without exploiting geometric information. Zhang et al. [50] perform a comprehensive study on such bag-of-features systems. They present a large-scale evaluation with different keypoint detector types and feature descriptors, as well as different kernels and classifiers. Their experiments demonstrate that image representations based on distributions of local features are efficient for classification of texture and object images under challenging real-world conditions, including high intra-class variations and substantial background clutter.
Building a visual codebook

One of the important steps in BoW approach is to build a visual codebook (or dictionary). Visual codebook is a collection of visual words which is often created by an unsupervised clustering algorithm, as shown in Figure 2.9. And then the images will be described by these visual words. Therefore, the discriminative power of image representations is directly influenced by the quality of codebook. In visual classification, the most commonly used clustering algorithm for building a visual codebook is the K-means algorithm.

Given a data set with \( N \) descriptors in a \( d \)-dimensional space \( \{x_1, \cdots, x_N\} \), the K-means algorithm is to partition the data set into some number of \( K \) clusters \( \mu_1, \cdots, \mu_k \in \mathbb{R}^d \), where \( k = 1, \cdots, K \) and \( \mu_k \) is a prototype associated with the \( k \)th cluster. The goal of the algorithm is then to find an assignment of data points to clusters, as well as a set of vectors \( \{\mu_k\} \), such that the sum of the squares of the distances of each data point to its closest vector \( \sum_{i=1}^{N} \|x_i - \mu_k\|^2 \) is minimized.

The standard algorithm was first proposed by Stuart Lloyd [51] as a technique for pulse-code modulation (PCM). They propose an optimization method that alternates between seeking the best means given the assignments \( (\mu_k = \text{avg}\{x_i : q_i = k\}) \), where \( q_1, \cdots, q_N \in \{1, \cdots, K\} \); and seeking then the best assignments given the means \( q_{ki} = \text{argmin}_k \|x_i - \mu_k\|^2 \).

Many variations of K-means algorithm have been proposed to improve the similarity power of data items in the same cluster or speedup each K-means step. Bezdek et al. [52] present Fuzzy C-Means (FCM), a soft version of K-means, where each data point has a fuzzy degree of belonging to each cluster. Gaussian Mixture Models (GMM) trained with expectation-maximization algorithm (EM algorithm) maintains probabilistic assignments to clusters, instead of deterministic assignments, and multivariate Gaussian distributions instead of means. And then the set of key-points can be represented directly...
Figure 2.9: Illustration of building a visual codebook: (a) the local image descriptors of training images are vector-quantized by an unsupervised clustering algorithm. (b) the obtained center point of each cluster is considered as a visual-word (or code-word) of the visual codebook. Image courtesy of Josef Sivic.

as a probability density function, over which a kernel can be defined [53, 54]. To reduce the computation complexity of K-means, Tapas Kanungo et al. [55] propose a simple and efficient implementation of Lloyd’s K-means algorithm, called the filter algorithm. The algorithm is easy to implement and only requires that a \( kd \)-tree be build once for the given data points. Their empirical analysis shows that the algorithm runs faster as the separation between clusters increase.

Encoding local features

Recently, several works have concentrated on improving the encoding features step, the goal is to produce representations that can reduce lost information or reconstruction error in the process. The problem can be split into two small steps. In the first step, each of the local features in an image will be assigned to one or more than one nearest neighbor visual words from the learned codebook. Therefore, for each image we obtain a set of vectors (or codes) corresponding to a projection of each descriptor onto the codebook. This step is called coding step. The second step is to compute the distribution of the codes in the cells of a spatial pyramid by some well-chosen aggregation statistic, called pooling step. Finally, each image in data set is described by a vector with fixed dimensions, which is an input feature vector for classifier in classification pipeline.

Vector quantization (VQ, also known as hard-assignment) is the baseline method for encoding local image features, which can be seen as a way of turning local feature vectors into very sparse, \( 1 - of - K \) codes. Each local image feature is described by a single visual word (the nearest one) from the codebook. Then, the set of vectors is aggregated into a single vector in \( K \) dimensions by using average spatial pooling, which can be interpreted as local histograms of visual words. However, the encoding by only a single visual word leads to degradation in the performance of classification. It is because of the two following problems.

- First, Chih-Fan Chen et al. [56] present that the standard VQ is not sufficient to represent different images from the same object class. As shown in Figure 2.10, the
two BoW models are not similar to each other due to texture, appearance variations, etc. To solve this problem, they explore the self-similarity of visual words within the existing BoW model and then construct the self-similarity hypercubes (SSH) features for each image. They claim that SSH can preserve the structure information of visual words present in images.

• Second, Oren Boiman et al. [57] reveal two practical problems downgrading the classification accuracy when using standard VQ. One is the quantization error problem. They show that the most informative descriptors tend to be rare in the database. As a result, these descriptors are most likely to be regarded as noise in K-means clustering algorithm and thus have high quantization error. This problem is illustrated in Figure 2.11 on a face image from Caltech 101, even when using a large codebook of quantized descriptors.

These two problems can be referred as codeword uncertainty and codeword plausibility when tracing back to Jan C. van Gemert’s work [58].

To reduce the quantization error, various methods have been proposed to replace this hard-assignment of individual SIFT descriptors by soft-assignment [59] or sparse coding [60]. James Philbin et al. [61] explore the techniques to map each visual region to a weighted set of words, allowing the inclusion of features which were lost in the quantization stage of previous systems. The set of visual-words is obtained by selecting words based on proximity in descriptor space. This work is also known as kernel codebook encoding, it is related to the works of J.D.R. Farquhar et al. [53] and Jan C. van Gemert et al. [58]. If the codebook of visual words is represented by a mixture of Gaussians (MoG), in which each Gaussian represents a word of the codebook, then the posterior probabilities of each Gaussian can be used as weight in the soft-assignment. Another line of research is to take the benefits of sparse signal models in restoration tasks. Julien
Figure 2.11: Effects of vector quantization. Informative descriptors have low frequency in database, leading to high quantization error. (a) An image from Face class in Caltech 101, (b) Quantization error of densely computed SIFT descriptors using a codebook with 6,000 visual words (red = high error; blue = low error). The most informative patches (eye, nose, etc.) have the highest quantization error. (c) The 8% of the descriptors in the image being most frequent in the database (simple edges) are indicated by green marks. (d) Magenta masks the 8% of the descriptors in the image that are least frequent in the database, mostly discriminative facial features. Image courtesy of Oren Boiman.

Mairal et al. [62] present a discriminative approach to supervised dictionary learning that effectively exploits the corresponding sparse signal decompositions in image classification tasks. Meanwhile, Jianchao Yang et al. [60] extend spatial pyramid matching (SPM) [63] approach by computing a spatial-pyramid image representation based on sparse codes of SIFT features. Furthermore, in the pooling step, the maximum value of a feature is used to summarize its activity over a region of interest. Sparse coding enables to operate local max pooling on multiple spatial scales to incorporate translation and scale invariance. They argue that the new image representation captures more salient properties of visual patterns and work well with linear classifiers. Locality-constrained Linear Coding (LLC [64]) applies locality constraint to select similar basis of local image descriptors (the nearest visual words) from a codebook, and learns a linear combination weight of these basis to reconstruct each descriptor. Perronnin et al. [37] propose a Fisher vector which captures the average first and second order differences between the image descriptors and the centers of a MoG, which can be thought of as a soft visual vocabulary.

2.1.3 Training classifiers

Once all images in training datasets have been encoded, we will learn the concept for each class by using a certain supervised classification method. There are a variety of different methods proposed in the literature for learning classification concepts, which include parametric and non-parametric methods, such as k-nearest neighbors (k-NN) classifier, Naïve Bayes classifier, C4.5 Decision Trees, support vector machines, etc. For visual classification tasks, however, support vector machines is the most frequently used classification model due to its effectiveness.
2.2 Benchmark datasets in computer vision

There are quite a few benchmark datasets for visual classification, such as MNIST, Caltech 101, Caltech 256, PASCAL VOC, LabelMe, etc. However, there are very few multi-class image datasets with many images for more than 300 categories. In recent years, there is an agreement that it is necessary to build a large scale dataset for studying object retrieval and recognition systems.

**MNIST**

The MNIST [65] dataset is constructed from NIST's Special dataset 3 and Special dataset 1 which contain binary images for handwriting digits. It has 10 classes with 60,000 training patterns and 10,000 testing patterns. The original black and white images are normalized to fit in a 20x20 pixel box while preserving their aspect ratio. This is a good dataset for people who want to practice learning techniques and pattern recognition methods on real-world data with minimal efforts on preprocessing and formatting.

**Caltech 101**

Caltech 101 is a well-annotated dataset for testing visual object recognition algorithms. It has 102 categories and 9,144 images in total. This dataset is collected by Li Fei-Fei et al. [8] by sending the words in Webster Collegiate Dictionary [66] as queries to image search engine (i.e. Google Image Search). And then, three minimal processing are performed on the categories. Firstly, the categories such as motorbike, airplane, cannon, etc., are flipped in order to make all instances face in the same direction. Secondly, categories with a predominantly vertical structure are rotated to arbitrary angle, as the model parts are ordered by their $x$-coordinate, so have trouble with vertical structures. This rotation is used for sake of programming simplicity. Finally, images are resized to around 300 pixels wide.

**Caltech 256**

The Caltech 256 dataset is introduced by Greg Griffin et al. [9]. It has 257 categories containing a total of 30,607 images. This dataset is collected in a similar manner to Caltech 101. The images are downloaded from both Google [1] and PicSearch [2] using scripts [3]. Unlike Caltech 101, there is no artificial modification in all images (e.g. rotation, right-left alignment). Duplicate images are removed if they contain over 15 similar SIFT descriptor. And then all images are verified and rated by the three following criteria:

1. **Good:** A clear example of the visual category

---

1 http://images.google.com
2 http://www.picsearch.com
3 Based on software written by Rob Fergus
2. **Bad**: A confusing, occluded, cluttered, or artistic example

3. **Not Applicable**: Not an example of the object category

The final set of images included in Caltech 256 are the ones that satisfy the requirements such as the size, no duplication and *good* rating. As a result, Caltech 256 has several improvements, compared to Caltech 101: i) the number of categories is more than doubled, ii) the minimum number of images in any category is increased from 31 to 81, iii) artifacts due to image rotation are avoided and iv) a new and larger clutter category is introduced for testing background rejection.

**LabelMe**

LabelMe [5] is a dataset and an online annotation tool that allows the sharing of images and annotation. Most images are collected by authors using a variety of digital cameras. A small proportion of the images are contributions from the users of the database or come from the website. The label of each image is obtained by using a web-based annotation tool. When the user enters the page, an image is displayed and the user is free to label as many objects depicted in the image as they wish. The LabelMe descriptions are then extended by using WordNet [67]. WordNet organizes semantic categories into a tree such that nodes appearing along a branch are ordered, with super-ordinate and subordinate categories appearing near the root and leaf nodes, respectively. The LabelMe annotations are extended by manually creating associations between the different text descriptions and WordNet tree nodes. Finally, only images that have at least one object annotated and object classes with at least 30 annotated examples are included in the dataset and therefore LabelMe has 183 categories with 30,369 images in total.

**PASCAL VOC**

PASCAL Visual Object Classes (VOC) challenge is a well-known benchmark in visual object category recognition and detection. It provides a standard dataset of images and annotation, and standard evaluation procedures for the machine learning and pattern recognition communities. This dataset contains 20 classes. The number of total images has gradually grown from 10,000 in 2007 to 12,000 in 2011. The VOC2007 dataset consists of annotated consumer photographs collected from the Flickr[^1] photo-sharing web-site. A new dataset with ground truth annotation has been released each year since 2006. The previous benchmark datasets investigate multi-category object recognition with a limited number of simple training images. These datasets contain a lot of images without clutter, variation in pose, and the images have been manually aligned to reduce the variability in appearance. These factors make the datasets less applicable to “real world” evaluation (e.g. Caltech 101, Caltech 256). However, the objective of VOC challenge aims to measure the performance of recognition methods on a wide spectrum.

[^1]: http://www.flickr.com
of natural images. Therefore, the images in this dataset have significant variability in terms of object size, orientation, pose, illumination, position and occlusion.

Tiny Images

TinyImage [68] dataset is a collection of 79,302,017 images covering more than 200 categories. The images are collected by querying all 75,846 non-abstract nouns from WordNet on seven independent image search engines (Altavista, Ask, Flickr, Cydral, Google, Picsearch, Webshots). To reduce the storage requirement, the images in the dataset are down-sampled into $32 \times 32$ pixel version of color images. Therefore, the entire dataset occupy only 760GB on a single hard disk. Each class in the dataset contains more than 1,000 images in average. Because the images are collected from Internet without verifying the content, only 10-25% of images are labeled accurately. Due to huge amount of images, TinyImage dataset has had success with non-parametric methods even simple image indexing techniques. However, the high rate of noise and the very low resolution of images make it less useful for large scale visual object classification with very large number of classes.

ImageNet

ImageNet [7] is a large-scale ontology of images built upon the backbone of the WordNet structure. The candidate images are collected from web searches for the nouns in WordNet and then the content of these images are verified by human labelers. Consequently, ImageNet contains the images with high quality annotation (\(\sim 99\%\) precision). To take the advantages of using the more sophisticated local image feature detectors available, the images are stored in full resolution with around $400 \times 350$ pixels in average. As shown in Table 2.1, ImageNet is much larger in scale, diversity and much more accurate than other benchmark datasets. The current released ImageNet has grown a big step in terms of both the number of images and the number of classes, as shown in Figure 2.12 - it has 21,841 classes with more than 14 millions images (1000 images for each class on average). Positively, it is necessary to have many images in the same class to cover visual variances, such as positions, view points, illumination, poses, background clutter, and occlusions, even if in the dataset, some classes have only one or less than 10 images so machine learning algorithm cannot learn anything.

2.3 Large scale visual classification

Despite of its simplicity, BoW is one of the most successful approaches in visual classification tasks. It may be enhanced by multi-scale spatial pyramids [63] on BoWs or histogram of oriented gradient [69] features. Some previous works consider exploiting the hierarchical structure of dataset for image recognition and achieve impressive improvements in accuracy and efficiency [70]. Related to classification is the problem of detection,
Table 2.1: Illustration of benchmark datasets in computer vision. ImageNet’12 is much larger in terms of both the number of classes and the number of images, and more diversity than other benchmark datasets.
often treated as repeated one-versus-all classification in sliding windows \cite{10, 71}. In many cases, such localization of objects might be useful to improve classification accuracy performance. However, in the context of large scale visual classification with hundreds or thousands of classes, these common approaches become computationally intractable.

To address this problem, Fergus et al. \cite{72} study semi-supervised learning on 126 hand labeled Tiny Images categories, Wang et al. \cite{73} show classification experiments on a maximum of 315 categories. Li et al. \cite{74} 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 80,000 visual words.

Furthermore, the emergence of ImageNet makes the complexity of visual classification much larger and very difficult to deal with. To tackle this challenge, the recent works study strategies on how to improve classification accuracy and avoid using high cost nonlinear classifier. A number of prominent works can be found in \cite{11, 75-77} where the original data is first explicitly transformed from linear space to nonlinear space to ensure linear separability of the classes and then linear classifier is trained in the resulting nonlinear space (usually of a much higher dimension). They argue that the classification accuracy of linear classifier with high-dimensional image signature is similar to low-dimensional BoW with nonlinear classifier.

In \cite{75}, the team wins the first place in ImageNet Challenge 2010 (ILSVRC 2010 \cite{78}), the local descriptors of each image are encoded by using either Local Coordinate Coding \cite{79} or Super-vector Coding \cite{80}. Then, they perform spatial pyramid pooling and the resulting image signature is a vector in approximately 262,000 dimensions. To train classifiers, they propose a parallel averaging stochastic gradient descent (ASGD) algorithm.
Table 2.2: The pros and cons of SVM classifiers for visual classification tasks.

<table>
<thead>
<tr>
<th>SVM Classifier</th>
<th>Training</th>
<th>Testing</th>
<th>Accuracy</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>+ Faster</td>
<td>+ Faster</td>
<td>- Lower</td>
<td>+ Higher</td>
</tr>
<tr>
<td>Nonlinear</td>
<td>- Lower</td>
<td>- Lower</td>
<td>+ Higher</td>
<td>- Lower</td>
</tr>
<tr>
<td>Additive kernels</td>
<td>+ Faster</td>
<td>+ Faster</td>
<td>+ Higher</td>
<td>+ Higher</td>
</tr>
</tbody>
</table>

Use Additive kernels! 5

With 1000 classes of ILSVRC 2010, it takes 4 days to train 1000 binary SVM classifiers (one-versus-all) for one feature channel on three 8-core computers.

Sánchez and Perronin [77] study the impact of high dimensional Fisher vectors on large dataset. They demonstrate that the larger the training dataset, the higher the impact of the dimensionality on classification accuracy. To get the state-of-the-art result on ILSVRC 2010, they make use of the spatial pyramids to increase the dimensionality of Fisher vectors to approximately 524,000 dimensions and then exploit Product Quantizer [81] to compress the data before training classifiers. With this approach, training 1000 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 efficient linear SVM classifiers, the recent works show that in visual classification tasks, nonlinear SVMs are superior in terms of classification accuracy when compared to linear rivals. In many test cases, nonlinear SVM with additive kernels give significantly higher rates in accuracy performance than dot product kernel. The main drawback of these approaches is the high cost of training nonlinear classifiers. It may be thousands of times higher than linear classifiers. However, many recent methods have been proposed to solve this limitation [17–20]. Additive kernel SVMs now use only few times more training time, compared to the state-of-the-art linear SVM solvers. To design a fast and accurate nonlinear kernel classifier for large scale datasets, Wu [21] proposes an efficient algorithm for PmSVM. They show empirically that PmSVM outperforms LIBLINEAR and the state-of-the-art additive kernel SVMs in terms of training time and classification accuracy. The pros and cons of SVM classifiers for visual classification tasks can be summarized in Table 2.2.

2.4 Online machine learning

In machine learning, the study on how to use training samples during the learning process is still an open challenge problem for researchers. Online learning (or incremental learning) and offline learning (or batch learning) are the two common methods. While
Chapter 2. State of The Art

24

Training data Classifier Function

Testing data

Figure 2.13: Batch learning: Given a training data (left). (1) The learning algorithm (center) takes the whole data as input from the beginning. This consumes a lot memory and computation power. (2) After learning, the output model can be used to predict testing data.

the first method often provides the good approximation results, the second one is usually a suitable way of handling big datasets. This section presents the advantages and disadvantages of these methods. And then the extensions to online learning algorithms are generally described with a particular focus on SVM algorithms.

Batch learning

Batch learning is the traditional method that learns a model from a batch of training data. In training phase, the entire training set is available from the beginning and stored in memory. And then all training samples are examined to measure and average the error of the prediction function. In optimization step, batch algorithms often solve a large optimization problem in order to find the optimal solution. Such steps can be repeated until a pre-defined stop condition is fulfilled. An illustration of batch learning algorithms is given in Figure 2.13. A number of examples for batch learning can be found in [82] and [83]. The main advantage with this method is that it is generally able to provide a good global approximation in the target function and thus high accuracy in testing. However, when armed with millions datapoints in very high dimension, the computation of an average cost on all training samples makes the batch methods very expensive in computation and memory usage.

Online learning

On the opposite side, online learning is the method that learns a model incrementally/decrementally from a sequence of samples. Normally, online learning algorithm proceeds an indefinite sequence of trials. Each trial can be decomposed into four main stages. First the algorithm receives an unlabeled sample and then learns from it. Second the algorithm predicts the label of this sample. Third this sample is removed from the training set. Fourth the algorithm is told the true label of the sample. In the fourth stage the algorithm uses this label feedback to update its hypothesis when appropriate. The learning process of such algorithms can be described in Figure 2.14. Perceptron [84] and Winnow [85] are the two first and simplest online algorithms that can perform well when a hyperplane exists that splits the data into two categories. These methods are
very cheap in computation and memory as they need only to store a single sample at a time. Thus, it is an excellent property for large scale applications. Over the years, numerous online learning algorithms have been suggested in the literature. Although strong generalization guarantees for online algorithms have been proven in a number of papers [22, 86-88], online algorithms hardly obtain a good performance in global generalization as batch algorithms after a single pass on training set. Therefore, the usual solution is to pass over the training data several times. However, this increases the computation as well as the memory requirements of online learning, especially when performing on large datasets.

The usage of online learning for prediction problems has been addressed by several authors. One of the most successful applications is the online algorithms for support vector machines. The recent works have presented some promising results and claim that their methods can handle data beyond the memory capacity. However, most of them conduct the simulations with enough memory and check the number of passes to access data [28, 89, 90]. To solve this challenge, a latest work has been proposed for training large linear SVMs when training data cannot fit into memory [24]. The main idea behind their method is to partition the training data into many small blocks of data. During training process, at any one time the algorithm loads a block of data into memory for training classifiers. This method can be schematized in Figure 2.15 with the extensions to conventional online learning algorithms on Stage 1 and 3. They have developed a package and name it LIBLINEAR-CDBLOCK. The evaluation shows that their implementations achieve the comparable results in global generalization when compared with batch algorithms and one interesting point is that the user can manage the memory usage of classifiers by choosing an appropriate data block size. Unfortunately, they do not conduct the experiments on large datasets with very large number of classes (e.g. thousands classes). Therefore, there still arise some disadvantages when applying their methods for large vision problems. We will discuss these disadvantages in Chapter 6.
In this dissertation we consider to extend both state-of-the-art large linear and nonlinear SVM classifiers for large scale visual classification. Thus, our study aims at seeking the novel learning algorithms that can shares speed, scalability of online methods and generalization ability of batch methods. Moreover, the combination of parallel algorithms and online learning is also studied in order to speedup the learning process on high performance computing platforms. This is the topic of both Chapter 4 and Chapter 6.
Chapter 3

Multi-feature and Multi-codebook

3.1 Introduction

During the past few years, BoW model has been a base-line method for many significant advances in visual classification. The key ingredient in design of visual classification systems is the determining of relevant class specific aspects while being powerful in intraclass variations. For given a test image these systems have to decide which class the image belong to. In the literature, numerous works have reported the impressive results on several benchmark datasets with small number of classes and low variability of visual appearance within each class. Most of these approaches use the well-known low-level image features, such as SIFT, SURF, and dense SIFT (DSIFT). Therefore, they are mostly based on local shape information for representing images, while other information (e.g. color, texture, etc.) can also be efficient in image classification tasks. The most effective features for classifying zebras may be very different from those for classifying cars. ImageNet with thousands of classes and over 1000 images in each class on average shows an extreme high intraclass variability in the same class. Therefore, the task of recognizing a huge number of classes clearly demands more powerful and efficient image representations. Moreover, large scale visual classification requires that the novel methods should be a good balance between effectiveness and computation efficiency. In this chapter we show how to address these challenges and achieve promising results over the usual systems. Firstly, we propose an effective approach multi-feature and multi-codebook that combines multiple diverse feature types. Secondly, the parallel solutions for extracting features, creating codebooks and constructing image representations have been proposed in order to reduce the computation time of the approach.

The remainder of this chapter is organized as follows. Section 3.2 briefly reviews the related work on image representation. Our multi-feature and multi-codebook approach is described in Section 3.3. Section 3.4 presents the numerical results of SIFT, SURF and DSIFT with LIBLINEAR SVM. Section 3.5 presents the numerical results of DSIFT, SOBEL and CENTRIST with PmSVM. The conclusion of this chapter is in Section 3.6.
3.2 Related work

Creating the signature of images based on BoW model includes three steps: 1) detecting key-features, 2) describing features, and 3) generating a codebook and mapping features to visual words. Recent works have studied these steps and achieved impressive improvements. However, in each processing step there exists a significant amount of lost information, and the resulting visual-words have not enough discriminative power for visual classification. Many approaches have been proposed to solve this challenge. In step 1, multiple local features are grouped to obtain a more global and discriminative feature [91, 92]. In step 2, high-dimensional descriptors enhanced by other information have been studied to get more image information [93]. In step 3, many methods have been proposed to build efficient quantizers or codebooks that reduce quantization errors and preserve more information of feature descriptors [61]. We have a more general view for all these steps and propose a novel approach that combines both multi-feature and multi-codebook approach to construct the final image signature. Our approach aims to increase the discriminative power of image signatures by embedding more useful information from the original image features. In multi-feature and multi-codebook approach, first BoW of images for each feature channel is constructed based on their corresponding codebook. The result is a bag-of-BoW for all feature types and we call it a bag-of-visual packets or a bag-of-packets. Finally, all BoW in the bag-of-packets are concatenated to form the final image signature. Our approach is novel in the way the image signature is constructed, that improves the discriminative power of image signature. This is the major difference between our approach and previous studies.

3.3 Multi-feature and multi-codebook

The high intraclass variability of images in the same class of ImageNet is a real challenge for image classification systems, as shown in Figure 3.1. Many previous works try to design a robust image feature which is invariant to image transformation, illumination and scale changes. There are some improvements when using robust features, but it is clear that none of feature descriptors have the same discriminative power for all classes. For instance, the features based on shape information might be useful when classifying cars or chairs. However, it will not be sufficient when the images are rotated or the objects are taken a shot in different camera angles. In this case, the appropriate choice should be the features based on interesting keypoints (e.g. SIFT). Obviously, instead of using a single feature type for all classes we can combine many different feature types - such as features based on shape, texture, color information - in order to improve the discriminative power of image representation in one class from all other classes. In this section we present a novel multi-feature and multi-codebook approach and show how to combine these features.

Let a set of all different feature descriptor types extracted from an image $i$ be $F = \{f^j_i\}$, where $f^j_i$ are the descriptors of feature type $j$ extracted from image $i$, $M$ is the number
of feature types, and \( j = 1, \ldots, M \). In our approach BoW histograms of each feature type are constructed based on their corresponding codebook, as shown in Figure 3.2. Instead of using a single codebook for constructing the final image signature, we use multiple codebooks \( \{C^1, C^2, \ldots, C^M\} \) that are built from different feature types. More specifically, the codebook \( C^j \) is used to construct BoW histogram \( h^j_i \) for feature descriptors \( f^j_i \in F \). Then all BoW histograms \( h^j_i \) are concatenated to form the final image signature \( H_i \). As a result, for each image \( i \), we obtain \( H_i \) with \( M \) elements \( H_i = \{h^1_i, h^2_i, \ldots, h^M_i\} \). For simplicity, we call \( H_i \) a bag-of-packets (BoP) that is the final image signature constructed based on different codebooks of the original image \( i \). A BoP is more discriminative than an usual BoW because two BoPs \( H_i \) and \( H_j \) are considered identical if and only if their corresponding BoW are identical. Formally, it takes the intersection of the BoW elements from multiple features:

\[
(H_i = H_j) \equiv (h^1_i = h^1_j) \wedge (h^2_i = h^2_j) \wedge \ldots \wedge (h^M_i = h^M_j) \tag{3.1}
\]
3.4 Experiment 1: (SIFT, SURF, DSIFT) + Feature map + LIBLINEAR

3.4.1 Datasets

Our approach has been evaluated on the two following subsets of ImageNet datasets.

**ImageNet 10.** This dataset contains the 10 largest classes from ImageNet (24,807 images with data size 2.4GB). There are more than 2000 diversified images per class. In each class, we sample 90% images for training and 10% images for testing.

**ImageNet 100.** This dataset contains the 100 largest classes from ImageNet (183,116 images with data size 23.6GB). In each class, we sample 50% images for training and 50% images for testing.

3.4.2 Parallel extracting feature

We perform experiments on computer Intel Xeon, 2.67GHz. Depending on parameters setting, extracting time of features (e.g. SIFT) of an image ranges from 0.46 to 1 second. Therefore, it is difficult to scale up to the full ImageNet because if it takes 1 second per image for extracting features then we need 14 millions $\times$ 1 second $\approx$ 162 days. To reduce extracting time, we apply parallel algorithms.

**SIFT/DSIFT.** VLFeat, a free software version for extracting SIFTs, can be downloaded from the author’s homepage ([www.vlfeat.org](http://www.vlfeat.org)). We use 8 CPU cores to extract features in a parallel way. We need 3 hours 30 minutes to extract more than 3 billions DSIFTs from the training dataset. That means it takes $\approx$ 0.14 second to extract features from an image. Therefore, with the full ImageNet, it would take $14M \times 0.14s \approx 22$ days.

**Parallel SURF.** Parallel SURF is a fast parallel version of SURF maintained by David Gossow [94]. We also use 8 CPU cores to extract features. We need 3 hours 18 minutes to extract more than 72 millions SURFs from the training dataset. That means it takes $\approx$ 0.13 second to extract features from an image. Therefore, with the full ImageNet, it would take $14M \times 0.13s \approx 21$ days.

3.4.3 Codebook fast building

In traditional bag-of-visual-words models, one of the steps that takes a long time to perform is building codebook. With large scale datasets we need to get a large amount of datapoints to build a discriminative codebook, so performing this task will become very complex in terms of computation time. One of the popular choices to build codebook is K-means clustering algorithm. However, the original implementation of K-means takes many days to converge when performing on large scale dataset as ImageNet. So reducing
the execution time for this task is becoming an essential task when we study a new framework for large scale image classification. In this experiment, we used the parallel version of K-means from Wei Dong (http://www.cs.princeton.edu/~wdong/kmeans). This program is a re-implementation of the K-means clustering algorithm. It has the following features:

1. An out-of-core implementation that allows clustering dataset larger than the main memory of computer,
2. Support parallel reading from multiple input files to maximize input throughput (when the input files are on independent devices). With OpenMPI, multiple machines can be used,
3. Accelerate L2 distance calculation with BLAS or KD-tree.

To perform K-means clustering algorithm, we use 8 CPU cores on the same machine as in Section 3.4.2. We sample all images in training dataset to build codebook with 5,000 centers. We set the maximum iteration of K-means to 20 and the threshold to 0.001. By using parallel K-means, we build codebooks from very large dataset in reasonable time.

3.4.4 Parallel bag-of-packets constructing

To speed up the constructing of BoWs histograms of images, we take into account the implementation of randomized kd-tree forests from VLFeat toolbox. It not only improves the effectiveness of the representation in high dimensions, but enables fast medium and large scale nearest neighbor queries among high dimensional data points. Once K-means is performed, we build a hierarchical structure for vocabularies by using vl_kdtreebuild. By this way, we can use vl_kdtreequery to speed up the process of mapping visual descriptors to visual words. The computation time for applying each codebook is similar to classical approach (single codebook). When we use $n$ codebooks for constructing BoPs of images, it means we need $n$ more times to finish this process. To achieve the computation time as single codebook approach, we perform the process of constructing BoPs in a parallel way. Consequently, the whole computation time of this process is the same as the largest individual standard approach. As shown in Table 3.1 and Table 3.2, we reduce the computation time for constructing BoPs of DSIFT+SURF+SIFT with multi-codebook to the same amount of time as DSIFT with single codebook.

3.4.5 Classification accuracy

The linear kernel on a classical histogram based feature gives very poor accuracy in image classification. Therefore, once BoW histogram is constructed, some recent image classification systems use feature map to convert BoW histogram from initial space to higher-dimensional space. This step is useful when one wants to stick to efficient linear classifiers [39]. The resulting image signature in high-dimensional space ensures linear
Table 3.1: Parallelize bag-of-packets construction of training images on ImageNet 10 (8 cores). The image signature is converted to high-dimensional space by using feature map.

<table>
<thead>
<tr>
<th>Features</th>
<th>Dimension</th>
<th>Time</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT</td>
<td>15,000</td>
<td>03m06s</td>
<td>560MB</td>
</tr>
<tr>
<td>SURF</td>
<td>15,000</td>
<td>07m02s</td>
<td>1.0GB</td>
</tr>
<tr>
<td>DSIFT</td>
<td>15,000</td>
<td>58m03s</td>
<td>2.9GB</td>
</tr>
<tr>
<td>DSIFT + SURF</td>
<td>30,000</td>
<td>59m01s</td>
<td>4.0GB</td>
</tr>
<tr>
<td>DSIFT + SURF + SIFT</td>
<td>45,000</td>
<td>01h05s</td>
<td>4.5GB</td>
</tr>
</tbody>
</table>

Table 3.2: Parallelize bag-of-packets construction of training images on ImageNet 100 (8 cores). The image signature is converted to high-dimensional space by using feature map.

<table>
<thead>
<tr>
<th>Features</th>
<th>Dimension</th>
<th>Time</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT</td>
<td>15,000</td>
<td>3h05m</td>
<td>2.1GB</td>
</tr>
<tr>
<td>SURF</td>
<td>15,000</td>
<td>3h52m</td>
<td>1.7GB</td>
</tr>
<tr>
<td>DSIFT</td>
<td>15,000</td>
<td>4h10m</td>
<td>8.0GB</td>
</tr>
<tr>
<td>DSIFT + SURF</td>
<td>30,000</td>
<td>4h11m</td>
<td>9.8GB</td>
</tr>
<tr>
<td>DSIFT + SURF + SIFT</td>
<td>45,000</td>
<td>4h15m</td>
<td>12.1GB</td>
</tr>
</tbody>
</table>

Table 3.3: Multiple features, overall classification accuracy (%) with LIBLINEAR.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>ImageNet 10</th>
<th>ImageNet 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT</td>
<td>49.89</td>
<td>22.67</td>
</tr>
<tr>
<td>SURF</td>
<td>56.02</td>
<td>34.07</td>
</tr>
<tr>
<td>DSIFT</td>
<td>75.09</td>
<td>54.07</td>
</tr>
<tr>
<td>DSIFT + SURF</td>
<td>76.25</td>
<td>55.13</td>
</tr>
<tr>
<td>DSIFT + SURF + SIFT</td>
<td>77.07</td>
<td>55.68</td>
</tr>
</tbody>
</table>

separability of the classes. Notice that before training classifiers, we should normalize BoW histogram so that the image size does not influence histogram counts [95]. In this experiments, we use L1-Norm to normalize BoW histogram and then convert it to higher-dimensional space by using homogeneous kernel map from [71].

To evaluate the performance of multi-feature and multi-codebook approach, we conduct the experiments for each single feature SIFT, SURF and DSIFT. Then we perform classification by using simultaneously different feature types DSIFT + SURF and DSIFT + SURF + SIFT. As shown in Table 3.3 (row #5 col #3 vs. row #1 col #3), in the case of training LIBLINEAR on the combination of 3 feature types, we significantly improve the performance of overall classification accuracy to +33.01%, compared to a single feature type SIFT (this is a relative improvement of more than 154%).

3.5 Experiment 2: (DSIFT, SOBEL, CENTRIST) + libHIK + PmSVM

As discussed in section 3.3, a certain local image feature is not always sufficient for all types of visual object datasets, thus with different local image features the visual
recognition system gives different results. In this section we do experiments with different local image features in order to validate our proposed method. Another interesting point in this experiment is to utilize an efficient and effective visual codebook [96] and a nonlinear kernel SVM classifier to improve the recognition accuracy.

3.5.1 Dataset

We use the same 10 and 100 largest classes from ImageNet as described in section 3.4. In each class, however, we sample 1000 images for training and 150 images for testing in order to be consistent with the setting of ILSVRC 2010. We construct BoW histogram of images by using libHIK [96] with the following descriptors DSIFT, SOBEL, CENTRIST, and the combination of these features. On the dataset 10 classes, we set 1000 codewords with parameters “use both, grid step size 2 and split level 2”. However, for the case of 100 classes, we increase the vocabulary size to 2000 codewords and construct BoW histogram of images by using the same method as we do with 10 classes. This encoding has been proven to give a good image classification performance with $\chi^2$ or Hellinger kernel SVM classifiers, as reported in the works of Wu et al. [96]. They demonstrate
Table 3.4: Parallelize bag-of-packets construction of training images on ImageNet 10 (160 cores). The image signature is construct by using libHIK.

<table>
<thead>
<tr>
<th>Features</th>
<th>Dimension</th>
<th>Time</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIFT</td>
<td>31,000</td>
<td>05m35s</td>
<td>1.6GB</td>
</tr>
<tr>
<td>CENTRIST</td>
<td>31,000</td>
<td>10m58s</td>
<td>1.0GB</td>
</tr>
<tr>
<td>SOBEL</td>
<td>31,000</td>
<td>05m50s</td>
<td>2.2GB</td>
</tr>
<tr>
<td>DSIFT + SOBEL</td>
<td>62,000</td>
<td>05m55s</td>
<td>3.9GB</td>
</tr>
<tr>
<td>DSIFT + CENTRIST</td>
<td>62,000</td>
<td>10m57s</td>
<td>2.6GB</td>
</tr>
<tr>
<td>DSIFT + SOBEL + CENTRIST</td>
<td>93,000</td>
<td>11m20s</td>
<td>4.0GB</td>
</tr>
</tbody>
</table>

Table 3.5: Parallelize bag-of-packets construction of training images on ImageNet 100 (160 cores). The image signature is construct by using libHIK.

<table>
<thead>
<tr>
<th>Features</th>
<th>Dimension</th>
<th>Time</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIFT</td>
<td>62,000</td>
<td>1h30m</td>
<td>17.4GB</td>
</tr>
<tr>
<td>CENTRIST</td>
<td>62,000</td>
<td>3h13m</td>
<td>12.8GB</td>
</tr>
<tr>
<td>SOBEL</td>
<td>62,000</td>
<td>1h40m</td>
<td>28.8GB</td>
</tr>
<tr>
<td>DSIFT + SOBEL</td>
<td>124,000</td>
<td>1h52m</td>
<td>50.1GB</td>
</tr>
<tr>
<td>DSIFT + CENTRIST</td>
<td>124,000</td>
<td>3h58m</td>
<td>33.4GB</td>
</tr>
<tr>
<td>DSIFT + SOBEL + CENTRIST</td>
<td>186,000</td>
<td>4h11m</td>
<td>64.2GB</td>
</tr>
</tbody>
</table>

Table 3.6: Multiple features, overall classification accuracy (%) with PmSVM.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>ImageNet 10</th>
<th>ImageNet 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIFT</td>
<td>74.81</td>
<td>57.89</td>
</tr>
<tr>
<td>CENTRIST</td>
<td>71.13</td>
<td>52.69</td>
</tr>
<tr>
<td>SOBEL</td>
<td>66.93</td>
<td>42.45</td>
</tr>
<tr>
<td>DSIFT + SOBEL</td>
<td>74.40</td>
<td>53.53</td>
</tr>
<tr>
<td>DSIFT + CENTRIST</td>
<td>77.20</td>
<td>61.57</td>
</tr>
<tr>
<td>DSIFT + SOBEL + CENTRIST</td>
<td><strong>77.46</strong></td>
<td><strong>61.78</strong></td>
</tr>
</tbody>
</table>

that with histogram features, the Histogram Intersection Kernel (HIK) is more effective than the Euclidean distance in unsupervised learning tasks. In particular, HIK can be used to significantly improve the generation of visual codebooks.

For constructing BoW histograms of images, we use the same method as in 3.4, in which the images in datasets are encoded in parallel way. Furthermore, to speedup the process of constructing BoP, the computation of images signature for each channel is also performed on several multi-core computers.

3.5.2 Classification accuracy

As shown in Figure 3.5 and 3.6, the combination of several features based on multi-codebook approach has consistently higher recognition accuracy over the single feature type. For instance, by using three types of feature (DSIFT + CENTRIST + SOBEL), we achieve 2.65% and 3.89% improvement in terms of accuracy on 10 and 100 classes of ImageNet, compared to a single feature DSIFT (Table 3.6, row #6 col #2 vs. row #1 col #2, and row #6 col #3 vs. row #1 col #3). Obviously, the multi-features and
multi-codebooks approach improves the performance of classification accuracy on several benchmark datasets.

3.6 Conclusion

We have proposed a fast and efficient framework for large scale image classification and show how to address this challenge by using ImageNet dataset as an example. In this framework, we have presented how to use a multi-core computer to reduce the computation time of extracting feature, creating codebook and constructing image representations. We have also presented a novel approach using several different local features simultaneously to improve the classification accuracy on large scale image datasets. The evaluation with different kinds of features shows that the multi-feature and multi-codebooks provides better results than a single codebook on different datasets (the relative increase is up to 82%).
Chapter 4

Parallel Balanced Bagging Support Vector Machines

4.1 Introduction

Visual classification is one of the most important research topics in the area of computer vision and machine learning. The usual pipeline for visual classification task, as shown in Figure 2.1, has been successful in many vision applications. Most previous approaches based on this pipeline have been evaluated only on small datasets, e.g. Caltech 101 [97], Caltech 256 [9], and PASCAL VOC [10] that can fit into desktop memory. Hence, at step 3 in Figure 2.1 - training classifiers, most researchers may choose either linear or nonlinear SVM classifiers that can be trained in a few minutes.

However, ImageNet with very large number of classes and images poses more challenges in training classifiers. With millions of training examples or millions of dimensions, training an accurate classifier may take weeks or even years [11, 75]. Thus, linear kernel SVM classifiers become popular for practical applications as they enjoy both faster training and faster testing than their nonlinear kernel versions. To take the benefits of linear classifiers, the original training data in low-dimensional space is often explicitly transformed to high-dimensional space by using a nonlinear mapping function in such a way as to render the problem linearly separable. Although LIBLINEAR is a current state-of-the-art fast training linear SVM solver, its current version takes very long training time on ILSVRC 2010 due to learning 1000 classifiers sequentially, independently. Therefore, it is crucial to develop parallel versions of LIBLINEAR based on high performance computing (HPC) models.

On another line of research, the papers [17, 19, 21] show that in the context of visual classification tasks, linear classifiers are inferior in terms of accuracy when compared to nonlinear counterparts. Wu [21] proposes a nonlinear classifier Power mean SVM (PmSVM) that outperforms LIBLINEAR and other additive kernel classifiers in terms
of training time and classification accuracy. However, the current version of PmSVM does not take into account the benefits of HPC. On ILSVRC 2010, it also takes very long time to train all binary classifiers. Therefore, it motivates us to study how to speedup PmSVM for large scale visual classification.

In this chapter we present how to extend these state-of-the-art classifiers including LIBLINEAR and PmSVM for large scale visual classification tasks. Our key contributions include:

1. Propose a balanced bagging algorithm for training binary classifiers. Our algorithm avoids training on full data and thus the training process of SVM classifiers rapidly converges to the optimal solution.

2. Parallelize the training process of all binary classifiers based on HPC models. In the training step of classifiers, our balanced bagging algorithm is applied to achieve the best performance.

Our approach has been evaluated on the 100 largest classes of ImageNet and ILSVRC 2010. The experiment shows that our approach is much faster than the original implementation without (or very few) compromising classification accuracy.

The rest of this chapter is organized as follows. Section 4.2 briefly reviews related work on large scale visual classification. Section 4.3 introduces support vector machines. In Section 4.4 we present its improvement for large number of classes and describe how to speedup the training process of classifiers by using our balanced bagging algorithm and take into account the benefits of HPC. The numerical results of the parallel versions of LIBLINEAR and PmSVM are presented in Section 4.5 and 4.6, respectively. We conclude this chapter in Section 4.7.

4.2 Related work

As mentioned in Section 2.3, in the scenarios of large scale visual classification, training SVM classifiers with nonlinear kernels is too expensive or intractable. For instance, if \( n \) is the number of training examples, the overall complexity of training process is \( \mathcal{O}(n^2) \) or \( \mathcal{O}(n^3) \) \[^{98, 99}\]. Therefore, many recent approaches use explicit data embedding strategy to avoid using nonlinear classifier \[^{17, 76}\]. By the way, one may stick to efficient linear classifiers with the training time complexity is \( \mathcal{O}(n) \) \[^{28, 100, 101}\]. Furthermore, Hsieh \emph{et al.} \[^{102}\] propose a novel dual coordinate descent method for large scale linear SVM with L1 and L2 loss functions (LIBLINEAR). They show that their method outperforms state-of-the-art linear SVMs in terms of training time. However, the current version of LIBLINEAR does not take the benefits from HPC and therefore training time is very long on ILSVRC 2010. It takes more than 52 hours to train 1000 binary classifiers.

On the other hand, in visual classification tasks, nonlinear classifiers are superior in terms of classification accuracy, compared to linear classifiers. Wu \[^{21}\] proposes PmSVM that
outperforms LIBLINEAR and other additive kernel SVMs on large scale datasets. For instance, on ILSVRC 2010, 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, the current version of PmSVM does not take into account the benefits of modern chip manufacturing. On a single core of our computer, it takes more than 10 hours to train 1000 binary classifiers.

In the multi-core era, computers with multi-cores or multiprocessors are becoming more and more popular and affordable. Thus, it motivates us to investigate parallel solutions and demonstrate how LIBLINEAR and PmSVM can benefit from modern platforms. Furthermore, in the case of large number of classes, we show that our balanced bagging algorithm is very useful to speedup the training process of classifiers without (or very few) compromising classification accuracy. Our experiments show very good results and confirm that the balanced bagging algorithm and parallel solutions are very essential for large scale visual classification in terms of training time.

4.3 Support vector machines

Let us consider a linear binary classification task, as depicted in Figure 4.1, with a training set \( T = \{(x_i, y_i)\}_{i=1}^{n}, x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}\), where \( n \) is the number of data-points and \( d \) is data-point dimension. SVM classification algorithm [14] aims to find the best separating hyperplane (denoted by the normal vector \( w \in \mathbb{R}^n \) and the scalar \( b \in \mathbb{R} \)), i.e. furthest from both class \( y = +1 \) and class \( y = -1 \). It can simply maximize the distance or margin between the supporting planes for each class \( (w.x - b = +1 \text{ for class } y = +1, w.x - b = -1 \text{ for class } y = -1) \). The margin between these supporting planes is \( \frac{2}{\|w\|} \) (where \( \|w\| \) is the 2-norm of the vector \( w \)). Any point \( x_i \) falling on the wrong side of its supporting planes is considered to be an error, denoted by \( \xi_i \geq 0 \). Therefore, the SVM has to simultaneously maximize the margin and minimize the error. This can be performed by solving the dual optimization problem (4.1).

![Figure 4.1: Linear separation of the datapoints into two classes](image-url)
\[
\min_{\alpha \in \mathbb{R}^n} f(\alpha) = \frac{1}{2} \alpha^T Q \alpha - e^T \alpha
\]
\[
s.t. \left\{ \begin{array}{l}
y^T \alpha = 0 \\
0 \leq \alpha_i \leq C, \ \forall i = 1, 2, ..., n
\end{array} \right.
\]

where \( e = [1, \ldots, 1]^T \), \( C \) 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 used to transform the training data from input space into feature space.

The support vectors (for which \( \alpha_i > 0 \)) are given by the optimal solution of (4.1), 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:

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

Variations on SVM algorithms use different classification functions. To change from linear to nonlinear classifier, no algorithm changes are required, except that the linear kernel function \( K \) as a dot product is replaced by a nonlinear kernel function. The most commonly used nonlinear kernel functions in machine learning applications are a polynomial function of degree \( d \), a RBF (Radial Basis Function) or a sigmoid function. And then we can get different support vector classification models. More details about SVM and other kernel-based learning methods can be found in [103].

### 4.3.1 LIBLINEAR SVM

LIBLINEAR proposed by Hsieh et al. [102] solves the dual problem (4.1) without using kernel function and then \( Q_{ij} = y_i y_j \langle x_i, x_j \rangle \). Therefore, we can predict a new sample by using (4.2) with \( K(x, x_i) = \langle x, x_i \rangle \).

To deal with training task, LIBLINEAR uses an efficient dual coordinate descent method for large linear SVM using L1- and L2-loss functions. And then, LIBLINEAR is simple and reaches an \( \epsilon \)-accurate solution in O(\( \log(1/\epsilon) \)) iterations. The algorithm is much faster than the state-of-the-art solvers such as Pegasos [28], TRON [104] and SVMperf [100].

### 4.3.2 Power mean SVM

Power Mean SVM proposed by Wu [21] replaces the kernel function \( K(x_i, x_j) \) in (4.1) and (4.2) with the power mean kernel \( M_p(x_i, x_j) (x_i \text{ and } x_j \in R^d) \), which is well-known
as a general form of many additive kernels (e.g. $\chi^2$ kernel, histogram intersection kernel or Hellinger’s kernel).

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

(4.3)

where $p \in \mathbb{R}$ is a constant.

- $\chi^2$ kernel ($p=-1$): $M_{-1}(x, y) = K_{\chi^2}(x, y) = \frac{2xy}{x+y}$
- Histogram intersection kernel ($p=-\infty$): $M_{-\infty} = K_{HI}(x, y) = \min(x, y)$
- Hellinger’s kernel ($p=0$): $M_0(x, y) = \sqrt{xy}$

PmSVM also uses the coordinate descent method [16] for dealing with training tasks. Furthermore, the gradient computation step of the coordinate descent algorithm can be estimated approximately by using polynomial regression with very low cost, as shown in Figure 4.2 (see more detail in [21]). Therefore, PmSVM is very efficient in both training and testing tasks, compared to LIBLINEAR and other additive kernel SVMs.

## 4.4 Improving SVM classifiers for 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
The second one is to decompose multi-class into a series of binary SVMs, including one-versus-all [14], one-versus-one [107] and Decision Directed Acyclic Graph [108]. Recently, hierarchical methods for multi-class SVM [109, 110] start from the whole data set, hierarchically divide the data into two subsets until every subset consists of only one class.

In practice, one-versus-all, 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 of 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. LIBLINEAR and PmSVM algorithm also use the one-versus-all approach to train independently $k$ binary classifiers. However, the current version of LIBLINEAR and 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 LIBLINEAR and PmSVM. The first one is to build the balanced bagging classifiers with sampling strategy. The second one is to parallelize the training task of all classifiers with multi-core computers.

### 4.4.1 Balanced bagging SVM classifiers

In one-versus-all approach, the learning task of LIBLINEAR and PmSVM is to try to separate the $i^{th}$ class (positive class) from the $k-1$ other classes (negative class). For very large number of classes, e.g. 1000 classes, this leads to the extreme imbalance between the positive class and the negative class. The problem is well-known as the class imbalance. As summarized by the review papers of [111–113] and the very comprehensive papers of [114, 115], 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 [116] or under-sampling the majority class [117, 118]. 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 LIBLINEAR and PmSVM belongs to the first approach (forms of re-sampling). Furthermore, the class prior probabilities in this context are highly unequal (e.g. the distribution of the positive class is 0.1% in the 1000 classes classification problem), and over-sampling the minority class is very expensive. We propose the balanced bagging LIBLINEAR and 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 LIBLINEAR and PmSVM trains $T$ models, as illustrated in Figure 4.3.
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 \( D_-' \) 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 LIBLINEAR and PmSVM is fast to converge to the solution. According to our experiments, by setting \( T = \sqrt{\frac{|D_-|}{|D_+|}} \), the balanced bagging LIBLINEAR and PmSVM achieve good results in very fast training speed. The procedure of the balanced bagging SVM is summarized in Algorithm 1.

**Algorithm 1: Balanced bagging SVM**

**input:** \( D_+ \) the training data of the positive class
\( D_- \) the training data of the negative class
\( T \) the number of base learners

**output:** SVM model

**Learn**

\[ \text{for } t \leftarrow 1 \text{ to } T \text{ do} \]
\[ 1. \text{ The subset } D_-' \text{ is created by sampling without replacement } |D_-'| \text{ negative datapoints from } D_- \text{ (with } |D_-'| = |D_+|) \]
\[ 2. \text{ Build a SVM model using the training set (including } D_+ \text{ and } D_-') \]
\[ \text{end} \]

combine \( T \) models (averaging) into the aggregated SVM model

**4.4.2 Parallel SVMs training**

Although LIBLINEAR and PmSVM and their balanced bagging versions deal with very large dataset with high speed, they do not take into account the benefits of HPC, e.g.
multi-core computers or grids. Furthermore, LIBLINEAR and PmSVM and their balanced bagging versions train independently $k$ binary classifiers for $k$ classes problems. This is a nice property for parallel learning. Our investigation aims at speedup the training task of multi-class LIBLINEAR and PmSVM and their balanced bagging versions with multi-processor computers or grids. The idea is to learn $k$ binary classifiers in parallel.

### Algorithm 2: Hybrid MPI/OpenMP parallel SVM

**Input:** $D$ the training dataset with $k$ classes  
$P$ the number of MPI processes  

**Output:** SVM model

**Learn:**

1. $MPI - PROC_1$
2. 
   
   #pragma omp parallel for
   
   for $c_1 ← 1$ to $k_1$ do  
   
   /* class $c_1$ */
   
   Build a binary SVM solver model using the training set $D$ to separate the positive class $c_1$ from the rest.

3. $MPI - PROC_P$
4. 
   
   #pragma omp parallel for
   
   for $c_P ← 1$ to $k_P$ do  
   
   /* class $c_P$ */
   
   Build a binary SVM solver model using the training set $D$ to separate the positive class $c_P$ from the rest.

The parallel programming is currently based on two major models, Message Passing Interface (MPI) [119] and Open Multiprocessing (OpenMP) [120]. 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, a MPI process loads the whole dataset ($\sim$ 25GB) into memory during learning tasks, making it wasteful. The simplest development of parallel LIBLINEAR and PmSVM algorithms is based on the shared memory multiprocessing programming 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 PmSVM algorithm is described in Algorithm 2. The number of MPI processes depends on the memory capacity of high performance computing systems.

### 4.5 Experiment 3: the parallel versions of LIBLINEAR

We have implemented four parallel versions of LIBLINEAR: 1) OpenMP version of LIBLINEAR (omp-LIBLINEAR), 2) balanced bagging version of omp-LIBLINEAR (omp-iLIBLINEAR), 3) hybrid MPI/OpenMP version of LIBLINEAR (mpi-omp-LIBLINEAR) and 4) balanced bagging version of mpi-omp-LIBLINEAR (mpi-omp-iLIBLINEAR).
In this section we compare the parallel versions of LIBLINEAR with the original implementation in terms of training time and classification accuracy.

**LIBLINEAR.** This is the linear SVM from [102] with default parameter value $C = 1$.

**iLIBLINEAR.** This is the balanced bagging LIBLINEAR with the same SVM parameters as LIBLINEAR.

Our experiments were run on a cluster of ten computers Intel Xeon E5645 CPU with the same hardware architecture as shown in Table 4.1. 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 3.2.0-4-amd64 (x86_64).

### 4.5.1 Dataset

The parallel versions of LIBLINEAR are designed for large scale datasets, so we have evaluated the performance of our approach on the two following datasets.

1. **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 histogram of images by using libHIK [96] with SIFT descriptor [13], 1000 codewords and parameters “use both, grid step size 2 and split level 1”. Finally, the image is encoded as a 12,000-dimensional vector. This encoding has been proven to give a good image classification performance. We end up with 10.5GB of training data.

2. **ILSVRC 2010**. This dataset contains 1000 classes from ImageNet Challenge 2010 with 1.2 million images (126GB) for training, 50,000 images (5.3GB) for validation and 150,000 images (16GB) for testing. To compare with the results reported in [21], we also use BoW feature set provided by [78] and the same method to encode every image as a vector in 21,000 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.

### 4.5.2 Training time

We have evaluated only the training time of SVM classifiers excluding the time needed to load data from disk. As shown in Figure 4.4, on a medium dataset ImageNet 100, our four parallel versions show a very good speedup in training process, compared to the original implementation of LIBLINEAR (Table 4.2).
**Chapter 4. Parallel Balanced Bagging Support Vector Machines**

**Figure 4.4:** Linear SVMs training time with respect to the number of OpenMP threads on ImageNet 100.

<table>
<thead>
<tr>
<th>Method</th>
<th># OpenMP threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>omp-LIBLINEAR</td>
<td>188.97 29.13 23.88</td>
</tr>
<tr>
<td>omp-iLIBLINEAR</td>
<td>64.14 9.77 7.86</td>
</tr>
<tr>
<td>5mpi-omp-LIBLINEAR</td>
<td>38.72 6.13 4.97</td>
</tr>
<tr>
<td>5mpi-omp-iLIBLINEAR</td>
<td>14.26 2.35 1.78</td>
</tr>
<tr>
<td>10mpi-omp-LIBLINEAR</td>
<td>20.61 3.36 2.51</td>
</tr>
<tr>
<td>10mpi-omp-iLIBLINEAR</td>
<td>6.77 1.09 0.83</td>
</tr>
</tbody>
</table>

**Table 4.2: Linear SVMs training time (minutes) on ImageNet 100.**

**ILSVRC 2010.** Our implementations achieve a significant speedup in training process when performing on this large dataset.

**OpenMP LIBLINEAR**

On a multi-core computer, OpenMP version of LIBLINEAR (omp-LIBLINEAR) achieves a significant speedup in training process with 16 OpenMP threads. As shown in Figure 4.5, our implementation is 7.9 times faster than the original implementation (Table 4.3, row #1 col #4 vs. row #1 col #2). Due to the restriction of our computer (16 cores), we set the maximum number of OpenMP threads to 16. We can set more than 16 OpenMP threads, but according to our observation there is very few significant speedup in training process because there is no more available core.
Balanced bagging LIBLINEAR

As shown in Figure 4.5, the balanced bagging version of LIBLINEAR (omp-iLIBLINEAR running with 1 thread) has a very fast convergence speed in training process, it is more than 6 times faster than the original implementation (Table 4.3, row #2 col #2 vs. row #1 col #2).

OpenMP balanced bagging LIBLINEAR

With balanced bagging algorithm applied to OpenMP version of LIBLINEAR (omp-iLIBLINEAR), as illustrated in Figure 4.5, we significantly speedup the training process on this training data. For instance, with the number of OpenMP threads set to 16, omp-iLIBLINEAR is 55 times faster than the original LIBLINEAR (Table 4.3, row #2 col #4 vs. row #1 col #2).

Hybrid MPI/OpenMP LIBLINEAR

Although OpenMP LIBLINEAR 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 hybrid approach, our implementation achieves an impressive parallelization performance on a cluster of ten SMP (symmetric multiprocessor) nodes. For shorter, we use the technical term ‘node’ instead of ‘SMP node’. The program first loads the whole block of data into nodes and each MPI process runs on one node. Therefore, each MPI process can work with its local data independently. However, we cannot increase the number of MPI processes to exceed the memory capacity of a node. It is because each MPI process occupies the main memory during its computation process, resulting in an increase in the overall memory requirement. Fortunately, 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 show in Figure 4.5, our hybrid MPI/OpenMP version of LIBLINEAR (mpi-omp-LIBLINEAR) achieves a significant speedup in training process with 10 MPI processes and 16 OpenMP threads. Our implementation is 59 times faster than the original implementation (Table 4.3, row #5 col #4 vs. row #1 col #2). Due to the large size of this training data, each MPI process of LIBLINEAR needs to use $\sim$17GB main memory to train classifiers. With the memory restrictions of our computer, we can only evaluate mpi-omp-LIBLINEAR by setting the number of MPI processes to 10. In this case, we vary the number of OpenMP threads running in each MPI process. Again, with 16 cores of our computer we set the maximum number of OpenMP threads to 16.
Chapter 4. Parallel Balanced Bagging Support Vector Machines

Figure 4.5: Linear SVMs training time with respect to the number of OpenMP threads on ILSVRC 2010.

Table 4.3: Linear SVMs training time (minutes) on ILSVRC 2010.

<table>
<thead>
<tr>
<th>Method</th>
<th># OpenMP threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>omp-LIBLINEAR</td>
<td>3126.78</td>
</tr>
<tr>
<td>omp-iLIBLINEAR</td>
<td>516.75</td>
</tr>
<tr>
<td>5mpi-omp-LIBLINEAR</td>
<td>630.81</td>
</tr>
<tr>
<td>5mpi-omp-iLIBLINEAR</td>
<td>109.84</td>
</tr>
<tr>
<td>10mpi-omp-LIBLINEAR</td>
<td>419.74</td>
</tr>
<tr>
<td>10mpi-omp-iLIBLINEAR</td>
<td>70.26</td>
</tr>
</tbody>
</table>

Hybrid MPI/OpenMP balanced bagging LIBLINEAR

The most significant parallelization performance of LIBLINEAR we achieve is the combination of MPI/OpenMP and balanced bagging LIBLINEAR (mpi-omp-iLIBLINEAR). As shown in Figure 4.5, our implementation achieves a significant performance in training process with 10 MPI processes and 16 OpenMP threads. It is 409 times faster than the original LIBLINEAR (Table 4.3, row #6 col #4 vs. row #1 col #2). On ILSVRC 2010, we need only 8 minutes to train 1000 binary classifiers, compared to the original LIBLINEAR (∼ 2 days and 4 hours), as shown in Table 4.3. This result confirms that our approach has a great ability to scale up to full ImageNet dataset with more than 21,000 classes.

We could expect the computing time would decrease according to the number of processors or cores used. This is only true when this number is small enough. When it becomes larger there is an overhead in the bus when we access the same memory address. When
we increase the number of OpenMP threads, the overhead will increase. Therefore, the computation time does not decrease linearly with the number of OpenMP threads. When we use 10 MPI, 8 OpenMP vs. 10 MPI, 16 OpenMP the difference between them is very small.

4.5.3 Classification accuracy

As shown in Table 4.4, in terms of classification accuracy the balanced bagging LIBLINEAR (iLIBLINEAR) has very nearly the same rate with the original implementation on medium dataset ImageNet 100 and very large dataset ILSVRC 2010. Note that iLIBLINEAR runs much faster than LIBLINEAR.

4.6 Experiment 4: the parallel versions of PmSVM

We have implemented four parallel versions of PmSVM: 1) OpenMP version of PmSVM (omp-PmSVM), 2) balanced bagging version of omp-PmSVM (omp-iPmSVM), 3) hybrid MPI/OpenMP version of PmSVM (mpi-omp-PmSVM) and 4) balanced bagging version of mpi-omp-PmSVM (mpi-omp-iPmSVM).

Our experiments were run on the same cluster of computers as described in Table 4.1.

### Dataset

The parallel versions of PmSVM are designed for large scale datasets, so we have evaluated the performance of our approach on the same datasets as in Section 4.5.1.

---

**Table 4.4: Linear SVMs overall classification accuracy (%)**.

<table>
<thead>
<tr>
<th>Method</th>
<th>ImageNet 100</th>
<th>ILSVRC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>43.17</td>
<td>21.11</td>
</tr>
<tr>
<td>omp-LIBLINEAR</td>
<td>43.17</td>
<td>21.11</td>
</tr>
<tr>
<td>omp-iLIBLINEAR</td>
<td>43.00</td>
<td>21.08</td>
</tr>
</tbody>
</table>
Chapter 4. Parallel Balanced Bagging Support Vector Machines

Figure 4.6: SVMs training time with respect to the number of OpenMP threads on ImageNet 100.

Table 4.5: PmSVMs training time (minutes) on ImageNet 100.

<table>
<thead>
<tr>
<th>Method</th>
<th># OpenMP threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>188.97</td>
</tr>
<tr>
<td>omp-PmSVM</td>
<td>68.08</td>
</tr>
<tr>
<td>omp-iPmSVM</td>
<td>23.94</td>
</tr>
<tr>
<td>5mpi-omp-PmSVM</td>
<td>13.91</td>
</tr>
<tr>
<td>5mpi-omp-iPmSVM</td>
<td>6.09</td>
</tr>
<tr>
<td>10mpi-omp-PmSVM</td>
<td>7.57</td>
</tr>
<tr>
<td>10mpi-omp-iPmSVM</td>
<td>4.62</td>
</tr>
</tbody>
</table>

4.6.2 Training time

We have evaluated only the training time of SVM classifiers excluding the time needed to load data from disk. As shown in Figure 4.6, on medium dataset ImageNet 100, our four parallel versions show a very good speedup in training process, compared to the original implementation of PmSVM and LIBLINEAR (Table 4.5).

ILSVRC 2010. Our implementations achieve a significant speedup in training process when performing on this large dataset.

OpenMP PmSVM

On a multi-core computer, OpenMP version of PmSVM (omp-PmSVM) achieves a significant speedup in training process with 16 OpenMP threads, as shown in Figure 4.7.
Our implementation is 11 times faster than the original PmSVM and 57 times faster than LIBLINEAR (Table 4.6, row #2 col #4 vs. row # 2 col #2 and row #1 col #2). Due to the restriction of our computer (16 cores), we set the maximum number of OpenMP threads to 16. We can set more than 16 OpenMP threads, but according to our observation there is very few significant speedup in training process because there is no more available core.

**Balanced bagging PmSVM**

As shown in Figure 4.7, the balanced bagging version of PmSVM (omp-iPmSVM running with 1 thread) has a very fast convergence speed in training process, it is more than 10 times faster than the original implementation of PmSVM (Table 4.6, row #3 col #2 vs. row #2 col #2).

**OpenMP balanced bagging PmSVM**

With balanced bagging algorithm applied to OpenMP version of PmSVM (omp-iPmSVM), as illustrated in Figure 4.7, we significantly speedup the training process on this training data. For instance, with the number of OpenMP threads set to 16, omp-iPmSVM is 114 times faster than the original PmSVM and 571 times faster than LIBLINEAR (Table 4.6, row #3 col #4 vs. row #2 col #2 and row #1 col #2).

**Hybrid MPI/OpenMP PmSVM**

As show in Figure 4.7, our hybrid MPI/OpenMP version of PmSVM (mpi-omp-PmSVM) achieves a significant speedup in training process with 10 MPI processes and 16 OpenMP threads. Our implementation is 100 times faster than the original PmSVM and 499 times faster than LIBLINEAR (Table 4.6, row #6 col #4 vs. row #2 col #2 and row #1 col #2). Due to the large size of this training data, each MPI process of PmSVM needs to use ~ 25GB main memory to train classifiers. With the memory restrictions of our computers, we can only evaluate mpi-omp-PmSVM by setting the number of MPI processes to 10. In this case, we vary the number of OpenMP threads running in each MPI process. Again, with 16 cores of our computer we set the maximum number of OpenMP threads to 16.

**Hybird MPI/OpenMP balanced bagging PmSVM**

The most significant parallelization performance of PmSVM we achieve is the combination of MPI/OpenMP and balanced bagging PmSVM (mpi-omp-iPmSVM). As shown in Figure 4.7, our implementation achieves a significant performance in training process with 10 MPI processes and 16 OpenMP threads. It is 411 times faster than the original PmSVM and 2057 times faster than LIBLINEAR (Table 4.6, row #7 col #4 vs. row #2
Chapter 4. Parallel Balanced Bagging Support Vector Machines

Figure 4.7: SVMs training time with respect to the number of OpenMP threads on ILSVRC 2010.

Table 4.6: PmSVMs training time (minutes) on ILSVRC 2010.

<table>
<thead>
<tr>
<th>Method</th>
<th># OpenMP threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>LIBLINEAR</td>
<td>3126.78</td>
</tr>
<tr>
<td>omp-PmSVM</td>
<td>624.56</td>
</tr>
<tr>
<td>omp-iPmSVM</td>
<td>64.41</td>
</tr>
<tr>
<td>5mpi-omp-PmSVM</td>
<td>135.73</td>
</tr>
<tr>
<td>5mpi-omp-iPmSVM</td>
<td>13.80</td>
</tr>
<tr>
<td>10mpi-omp-PmSVM</td>
<td>71.15</td>
</tr>
<tr>
<td>10mpi-omp-iPmSVM</td>
<td>10.69</td>
</tr>
</tbody>
</table>

col #2 and row #1 col #2). On ILSVRC 2010, we need only 1.52 minutes to train 1000 binary classifiers, compared to the original PmSVM (~ 10 hours) and LIBLINEAR (~ 2 days and 4 hours), as shown in Table 4.6. This result confirms that our approach has a great ability to scale up to full ImageNet dataset with more than 21,000 classes.

4.6.3 Classification accuracy

As shown in Figure 4.8, in terms of classification accuracy PmSVM and iPmSVM 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 2010 (from +4.24% to +4.53%, the relative improvement is more than 20.1%) (Table 4.7).
### Table 4.7: SVMs overall classification accuracy (%).

<table>
<thead>
<tr>
<th>Method</th>
<th>ImageNet 100</th>
<th>ILSVRC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>43.17</td>
<td>21.11</td>
</tr>
<tr>
<td>PmSVM</td>
<td>55.76</td>
<td>25.64</td>
</tr>
<tr>
<td>omp-PmSVM</td>
<td>55.76</td>
<td>25.64</td>
</tr>
<tr>
<td>omp-iPmSVM</td>
<td>54.54</td>
<td>25.35</td>
</tr>
</tbody>
</table>

**Figure 4.8:** Overall classification accuracy of SVM classifiers.

Note that iPmSVM runs much faster than PmSVM (about 2 times faster with Image 100 and 10 times faster with ILSVRC 2010) without (or very few) compromising classification accuracy.

### 4.7 Conclusion

We have developed the parallel versions of LIBLINEAR and PmSVM to efficiently deal with large scale datasets with very large number of classes like ImageNet. To speedup the training process of SVM classifiers, we have proposed two effective ways. The first one is to build the balanced bagging classifiers with under-sampling strategy. Our algorithm avoids training on full training data, so the training process of LIBLINEAR and PmSVM rapidly converges to the optimal solution. The second one is to parallelize the training process of all classifiers with multi-core computers. We have developed the parallel versions of LIBLINEAR and PmSVM based on HPC models (OpenMP, MPI, and hybrid MPI/OpenMP). And then the balanced bagging algorithm is applied to achieve the best performance in SVM training process.

Our approach has been evaluated in terms of training time and classification accuracy on a large dataset ImageNet. On ILSVRC 2010, our implementation is thousand times
faster than the original implementation. To obtain this result, we have set the number of processes to 160 on our computers. Therefore, we can get better performance by using more resources (CPU cores, computer, etc.). Furthermore, by applying the balanced bagging approach we significantly speedup the training process of the classifiers without (or very few) compromising the overall classification accuracy. For the case of linear classifier, our implementation needs only 7.64 minutes to train 1000 binary classifiers; and especially for the case of nonlinear classifier it needs only 1.52 minutes. However, when the training data is larger and cannot fit into main memory, both LIBLINEAR and PmSVM encounter a problem. To solve this problem, we may consider two following possible solutions. The first one is to study the approach that avoids loading the whole training data into main memory, as in [121]. The other is to compress the training data and handle it on the fly, as in [77]. This challenge will be addressed in Chapter 6.

In one-versus-all approach, the larger the “all” class is, the faster the balanced bagging algorithms are, compared to non-balanced ones. This is an interesting property in the case of ImageNet dataset with more than 21,000 classes. Other sampling methods more efficient and sophisticated than simple random sampling could be used, e.g. [122–124] to minimize the small decrease in accuracy.
Chapter 5

Parallel Stochastic Gradient Descent Algorithms Support Vector Machines

5.1 Introduction

Training a fast and accurate classifier for vision tasks remains a challenge in the image classification community. For small datasets with thousands of images, one may prefer to choose nonlinear classifier because it offers higher rate in classification accuracy and the training process is in a few minutes. However, for large datasets with millions images and thousands classes, training a nonlinear classifier is an impractical task. In [11, 75], they report that with millions of images training an accurate SVM classifier may take weeks or even years. Thus, the recent works on large scale learning classifiers have focused on building linear classifiers for large scale visual classification tasks. In many test cases, linear SVM classifier is a trade-off between training time and classification accuracy [125]. Shalev-Shwartz et al. [28] and Bottou et al. [126] propose stochastic gradient descent algorithms for SVM (denoted by SVM-SGD) that shows the promising results for large scale binary classification problems. Recently, an extension of SVM-SGD [127] uses the one-versus-all strategy for dealing with large-scale images in very high-dimensional signatures and thousand classes. However, the current version of SVM-SGD does not take into account the benefits of HPC. On ILSVRC 2010, it takes very long time to train 1000 binary classifiers. Therefore, it motivates us to study how to speedup SVM-SGD for large scale visual classification. In this chapter we extend the binary SVM-SGD in several ways to develop the new parallel multi-class SVM-SGD algorithms for efficiently classifying large image datasets into many classes. The idea is to build:

1. A balanced training algorithm for binary SVM-SGD classifiers,
2. A parallel training process of classifiers with several multi-core computers/grid.
The rest of this chapter is organized as follows. Section 5.2 briefly reviews the related work on large scale visual classification. Section 5.3 introduces stochastic gradient descents for support vector machines. In Section 5.4 we present its improvement for large number of classes and describe how to speedup the training process of SVM-SGD by using our balanced training algorithm and take into account the benefits of HPC. Section 5.5 presents numerical results before the conclusion of this chapter in Section 5.6.

5.2 Related Work

ImageNet with more than million images for thousand categories makes the task of visual classification become a very large challenge. To tackle this challenge, many researchers are beginning to study strategies on how to improve the accuracy performance and avoid using high cost nonlinear kernel SVMs for training classifiers. The recent prominent works for these strategies have been proposed in [11, 75–77] where the data is first transformed by a nonlinear mapping induced by a particular kernel and then efficient linear classifiers are trained in the resulting space. They argue that the classification accuracy of linear classifiers with high dimensional image representations is similar to low dimensional BoW with nonlinear kernel classifiers. Therefore, many previous works in large scale visual classification have converged on building linear classifiers by using the state-of-the-art linear classifier LIBLINEAR [102]. However, the recent works in [28, 126] show empirically that SVM-SGD is faster than LIBLINEAR on many benchmark datasets. Sánchez et al. [77] and Perronnin et al. [127] study the impact of high dimensional Fisher vectors on large scale datasets. They show that the larger the training dataset, the higher the impact of the dimensionality on classification accuracy. To obtain the state-of-the-art results on ILSVRC 2010, they make use of the spatial pyramids to increase the dimensionality of Fisher vector and then exploit Product Quantizer [81] to compress the data before training classifiers. With this strategy, the training data can fit into main memory of a single computer (48GB). To train classifiers, they employ Stochastic Gradient Descent [126] with early stopping, reweighting data, regularization, step size and the computation of dot product is parallelized on 16 cores of their computer. Our approach is quite different with this work, we train classifiers with a sampling strategy and a smooth hinge loss function and parallelize the training task of binary classifiers with several multi-core computers.

A grid with several multi-core computers bring to us many advantages. Advanced technologies designed for the systems where several processes have access to shared or distributed memory space are becoming popular choice for high performance computing algorithms. Therefore, it motivates us to investigate parallel solutions and demonstrate how SVM-SGD can benefit from these modern platforms. Furthermore, in the case of very large number of classes, we propose the balanced training algorithm that can be applied in order to speedup the training process of classifiers without compromising classification accuracy. Our experiments show very good results and confirm that the
balanced training algorithm and parallel solutions are very essential for large scale visual classification in terms of training time.

5.3 SVM with stochastic gradient descent

Instead of solving the dual form of SVM, one may solve its primal form (5.1)

\[
\min \Psi(w, b, \xi) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \xi_i
\]

\[
s.t.: y_i(w.x_i - b) + \xi_i \geq 1
\]

\[
\xi_i \geq 0
\]

where the positive constant \( C \) is used to tune errors and margin size.

Unfortunately, the computational cost requirements of the SVM solutions in (5.1) are at least \( O(n^2) \), where \( n \) is the number of training datapoints, making classical SVM intractable for large datasets.

We can reformulate the SVM problem in quadratic programming (5.1) in an unconstrained problem. Firstly, the bias \( b \) can be ignored without generality loss. Consequently, the constraints \( y_i(w.x_i - b) + \xi_i \geq 1 \) in (5.1) are rewritten as follows:

\[
\xi_i \geq 1 - y_i(w.x_i)
\]

Then, the constraints (5.2) and \( \xi_i \geq 0 \) are rewritten by the hinge loss function:

\[
\xi_i = \max\{0, 1 - y_i(w.x_i)\}
\]

Finally, substituting for \( \xi \) from the constraint in terms of \( w \) into the objective function \( \Psi \) of the quadratic programming (5.1) yields an unconstrained problem (5.4):

\[
\min \Psi(w, [x, y]) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i(w.x_i)\}
\]

The approaches in [28, 126] have proposed the stochastic gradient descent method to solve the unconstrained problem (5.4). The stochastic gradient descent for SVM (SVM-SGD) updates \( w \) on \( T \) epochs with a learning rate \( \eta \). For each epoch \( t \), the SVM-SGD uses a single randomly received datapoint \((x_i, y_i)\) to compute the sub-gradient \( \nabla_t \Psi(w, [x_i, y_i]) \) and update \( w_{t+1} \).
As mentioned in [28, 126], the SVM-SGD algorithm quickly converges to the optimal solution due to the fact that the unconstrained problem (5.4) is convex games on very large datasets. The algorithmic complexity of SVM-SGD is linear with the number of datapoints. An example of its effectiveness is given with the classification into two classes of 780,000 datapoints in 470,000-dimensional input space in 2 seconds on a PC and the test accuracy is similar to standard SVM.

5.4 Extensions of SVM-SGD to large number of classes

For multi-class classification problem, SVM-SGD algorithm uses one-versus-all strategy. Therefore, as discussed in Section 4.4 this leads to the two problems:

1. SVM-SGD algorithm deals with the imbalanced data for building binary classifiers,
2. SVM-SGD algorithm also takes very long time to train very large number of binary classifiers in sequential mode using a single processor.

A recent multi-class SVM-SGD algorithm proposed by [127] uses the one-versus-all strategy for classifying large-scale images in very high-dimensional signatures and thousands of classes. The recommendations for this algorithm include early stopping, reweighting used to adjust the sample data, regularization, step size.

And then, our multi-class SVM-SGD algorithm also uses the one-versus-all approach to train independently $k$ binary classifiers. We propose two ways for creating the new multi-class SVM-SGD algorithm being able to handle very large number of classes in high speed. The first one is to build balanced training of binary classifiers with a sampling strategy and a smooth hinge loss function. The second one is to parallelize the training task of all classifiers with several multi-core computers/grids.

5.4.1 Balanced training SVM-SGD

In one-versus-all approach, the learning task of SVM-SGD tries to separate the $i^{th}$ class (positive class) from the $k-1$ others classes (negative class). For very large number of classes, e.g. 1000 classes, this leads to the extreme imbalance between the positive and the negative class. The problem is well-known as the class imbalance. The problem of the SVM-SGD algorithm comes from the update rule using a random received datapoint. The probability for a positive datapoint sampled is very small (about 0.001) compared with the large chance for a negative datapoint sampled (e.g. 0.999). And then, the SVM-SGD concentrates mostly on the errors produced by the negative datapoints. Therefore, the SVM-SGD has difficulty to separate the positive class from the rest.

As summarized by the review papers of [111-113] and the very comprehensive papers of [114, 115], 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 [116] or under-sampling the majority class [117, 118]. 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 training SVM-SGD simultaneously uses the two approaches. Furthermore, the class prior probabilities in this context are highly unequal (e.g. the distribution of the positive class is 0.1% in the 1000 classes classification problem), and over-sampling the minority class is very expensive. Therefore, our balanced training SVM-SGD uses under-sampling the majority class (negative class). The balanced training SVM-SGD also modifies the updating rule using the skewed misclassification costs.

Although the SVM-SGD algorithm has impressive convergence properties due to the fact that the unconstrained problem (5.4) is convex games on very large datasets. The hinge loss function

\[ L(w, [x_i, y_i]) = \max\{0, 1 - y_i(w.x_i)\} \]

is discontinuously in the derivative at \( y_i(w.x_i) = 1 \), and then the SGD’s convergence rate still can not be faster than \( O(\ln(T)/T) \) as mentioned in [128]. One way to resolve this issue is to use a surrogate smooth loss function of the hinge loss function in the unconstrained problem (5.4), this leads to achieve the optimal rate \( O(1/T) \), illustrated in [129, 130]. Then, we propose to substitute the hinge loss \( L(w, [x_i, y_i]) = \max\{0, 1 - y_i(w.x_i)\} \) in the unconstrained problem (5.4) by the smooth hinge loss [131], as follows:

\[ L^s(w, [x_i, y_i]) = \begin{cases} 
\frac{1}{2} - y_i(w.x_i) & \text{if } y_i(w.x_i) \leq 0 \\
\frac{1}{2}[1 - y_i(w.x_i)]^2 & \text{if } 0 < y_i(w.x_i) < 1 \\
0 & \text{if } 1 \leq y_i(w.x_i)
\end{cases} \]  

(5.5)

And then our balanced training SVM-SGD for binary classification tasks (described in Algorithm 3) updates \( w \) on \( T \) epochs. For each epoch \( t \), the reduced dataset \( D' \) is created by the full set of positive class \( D_+ \) and under-sampling the negative class \( D_- \), the SVM-SGD randomly picks a datapoint \( (x_i, y_i) \) from the reduced dataset \( D' \) to compute the sub-gradient \( \nabla_t \Psi(w, [x_i, y_i]) \) (according to the smooth hinge loss (5.5) and update \( w_{t+1} \) (using the skewed misclassification costs) as follows:

\[ w_{t+1} = w_t - \eta_t \nabla_t \Psi(w, [x_i, y_i]) = w_t - \eta_t (\lambda w_t + \nabla_t L^s(w, [x_i, y_i])) \]  

(5.6)

\[ \nabla_t L^s(w, [x_i, y_i]) = \begin{cases} 
-\frac{1}{|D_+|}y_i x_i & \text{if } y_i(w_t.x_i) \leq 0 \\
-\frac{1}{|D_-|}y_i x_i[1 - y_i(w_t.x_i)] & \text{if } 0 < y_i(w_t.x_i) < 1 \\
0 & \text{if } 1 \leq y_i(w_t.x_i)
\end{cases} \]  

(5.7)

where \( |D_c| \) is the cardinality of the class \( c \in \{\pm 1\} \).
Algorithm 3: Balanced training SVM-SGD for binary classification tasks

**Input**: Training data of the positive class $D_+$
Training data of the negative class $D_-$
Positive constant $\lambda > 0$
Number of epochs $T$

**Output**: SVM-SGD model $w$

1. Init $w_1 = 0$
2. for $t \leftarrow 1$ to $T$
   3. Creating the reduced dataset $D'$ from the full set of positive class $D_+$ and sampling
      without replacement $D'_-$ from dataset $D_-$ (with $|D'_-| = \sqrt{|D_-| \times |D_+|}$)
   4. Setting $\eta_t = \frac{1}{\sqrt{t}}$
   5. for $i \leftarrow 1$ to $|D_+|$ do
      6. Randomly pick a datapoint $[x_i, y_i]$ from reduced set $D'$
      7. if $y_i(w_t.x_i) \leq 0$ then
      8. \hspace{1em} $w_{t+1} = w_t - \eta_t (\lambda w_t - \frac{1}{|D'_+|} y_i x_i)$
      9. else if $y_i(w_t.x_i) < 1$ then
       10. \hspace{1em} $w_{t+1} = w_t - \eta_t (\lambda w_t - \frac{1}{|D'_+|} y_i x_i [1 - y_i(w_t.x_i)])$
      11. else
      12. \hspace{1em} $w_{t+1} = w_t - \eta_t \lambda w_t$
   13. end
   14. end
15. end
16. return $w_{T+1}$

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 ($D'_-$) done by balanced training SVM-SGD provides the reduced convex hull of $H_-$, called $H'_-$. And then, the minimum distance between $H_+$ and $H'_-$ is larger or equal than between $H_+$ and $H_-$ (full dataset). It is easier to achieve the separating boundary than learning on the full dataset. Therefore, the training task of balanced SVM-SGD is fast to converge to the solution.

5.4.2 Parallel multi-class SVM-SGD training

For $k$-classes problem, the multi-class SVM-SGD algorithm trains independently $k$ binary classifiers. Although balanced training SVM-SGD deals with binary classification tasks with high speed, the multi-class SVM-SGD algorithm does not take the benefits of high performance computing.

Our investigation aims at speedup the training tasks of multi-class SVM-SGD with several multi-processor computers. The idea is to learn $k$ binary classifiers in parallel way. We also make use of the benefits of the hybrid approach MPI/OpenMP to speedup the learning task of SVM-SGD. The parallel learning for multi-class SVM-SGD is described in Algorithm 4. The number of MPI processes depends on the memory capacity of the HPC system used.
**Algorithm 4:** Parallel multi-class SVM-SGD algorithm

**input**: $D$ the training dataset with $k$ classes  
$P$ the number of MPI processes  
**output**: SVM-SGD model

1. **Learning:**
2. $MPI \rightarrow PROC_1$
3. #pragma omp parallel for
4. for $c_1 \leftarrow 1$ to $k_1$ do /* class $c_1$ */
5. | training SVM-SGD($c_1$ vs -all)
6. end
7. :
8. $MPI \rightarrow PROC_P$
9. #pragma omp parallel for
10. for $c_P \leftarrow 1$ to $k_P$ do /* class $c_P$ */
11. | training SVM-SGD($t_P$ vs -all)
12. end

---

**5.5 Experiment 5: the parallel version of SVM-SGD**

We have implemented two parallel versions of SVM-SGD: 1) OpenMP version of balanced training SVM-SGD (Par-MC-SGD), 2) hybrid MPI/OpenMP version of balanced training SVM-SGD (mpi-Par-MC-SGD).

Franc et al. [132] have shown in their experiments that OCAS even in the early optimization steps shows often faster convergence than the so far in this domain prevailing approximative methods SGD and Pegasos [28].

We plan to compare our approach with Pegasos. Unfortunately, the original code of Pegasos from [28] offers only binary-class classification. Therefore, we have implemented a multi-class classification model (using one-versus-all strategy) for Pegasos and call it M-Pegasos. Moreover, we have found another well-developed toolbox BudgetedSVM [133] that implements multi-class model for Pegasos.

In this section we compare parallel algorithms of SVM-SGD with OCAS, LIBLINEAR, M-Pegasos and multi-class Pegasos from BudgetedSVM in terms of training time and classification accuracy.

**OCAS.** This is an optimized cutting plane algorithm for support vector machines [132] with the default parameter value $C = 1$.

**LIBLINEAR.** This is the linear SVM from [102] with default parameter value $C = 1$.

**BudgetedSVM.** This is a C++ toolbox [133] containing highly optimized implementations of three recently proposed algorithms for scalable training of SVM approximators: Adaptive Multi-hyperplane Machines (AMM), Budgeted Stochastic Gradient Descent (BSGD) and Low-rank Linearization SVM (LLSVM). The toolbox also includes Pegasos [28], a state-of-the-art linear SVM solver, as it is a special case of AMM. We set the parameters as follows $A = 0$ (equivalent to Pegasos), $\lambda = 0.0001$ and $e = 20$ epochs.
M-Pegasos. This is our implementation for multi-class Pegasos with the parameters as follows $\lambda = 0.0001$ and $e = 20$ epochs.

**Par-MC-SGD, mpi-Par-MC-SGD.** These are parallel balanced training SVM-SGD using $T = 20$ epochs and regularization term $\lambda = 0.0001$.

Our experiments were run on machine Linux 2.6.39-bpo.2- amd64, Intel(R) Xeon(R), CPU X5560, 2.8GHz, 16 cores, and 96GB main memory.

### 5.5.1 Dataset

The Par-MC-SGD, mpi-Par-MC-SGD algorithms are designed for large scale datasets, so we have evaluated the performance of our approach on the three following datasets.

**ImageNet 10.** This dataset contains the 10 largest classes from ImageNet (24,807 images with data size 2.4GB). There are more than 2000 diversified images per class. In each class, we sample 90% images for training and 10% images for testing. First, we construct bag-of-words histogram of every image by using dense SIFT descriptor (extracting SIFT on a dense grid of locations at a fixed scale and orientation), and 5000 codewords. Then, we use feature mapping from [17] to get the high-dimensional image representation in 15,000 dimensions. This feature mapping has been proven to give a good image classification performance with linear classifiers [17]. We end up with 2.6GB of training data.

**ImageNet 100.** This dataset contains the 100 largest classes from ImageNet (183,116 images with data size 23.6GB). In each class, we sample 50% images for training and 50% images for testing. We also construct bag-of-words histogram of every image by using dense SIFT descriptor and 5000 codewords. For feature mapping, we use the same method as we do with ImageNet 10. The final size of training data is 8GB.

**ILSVRC 2010.** This dataset contains 1000 classes from ImageNet with 1.2 million images (126GB) for training, 50,000 images (5.3GB) for validation and 150,000 images (16GB) for testing. We use BoW feature set provided by [78] and the method reported in [21] to encode every image as a vector in 21,000 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.

### 5.5.2 Training time

We have evaluated only the training time of SVM classifiers excluding the time needed to load data from disk. As shown in Figure 5.1 and 5.2, on small and medium datasets as ImageNet 10, ImageNet 100, our parallel versions show a very good speedup in training process, compared to OCAS, BudgetedSVM, M-Pegasos and LIBLINEAR (Table 5.1 and 5.2).
Figure 5.1: SVMs training time with respect to # OpenMP threads on ImageNet 10.

Table 5.1: SVMs training time (minutes) on ImageNet 10.

<table>
<thead>
<tr>
<th>Method</th>
<th># OpenMP threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCAS</td>
<td>6.23</td>
</tr>
<tr>
<td>BudgetedSVM</td>
<td>20.36</td>
</tr>
<tr>
<td>M-Pegasos</td>
<td>2.31</td>
</tr>
<tr>
<td>LIBLINEAR</td>
<td>2.02</td>
</tr>
<tr>
<td>Par-MC-SGD</td>
<td>3.80 0.89 0.57 0.55</td>
</tr>
<tr>
<td>2mpi-Par-MC-SGD</td>
<td>1.99 0.54 0.54 0.62</td>
</tr>
</tbody>
</table>

Table 5.2: SVMs training time (minutes) on ImageNet 100.

<table>
<thead>
<tr>
<th>Method</th>
<th># OpenMP threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCAS</td>
<td>904.43</td>
</tr>
<tr>
<td>BudgetedSVM</td>
<td>148.41</td>
</tr>
<tr>
<td>M-Pegasos</td>
<td>59.29</td>
</tr>
<tr>
<td>LIBLINEAR</td>
<td>30.41</td>
</tr>
<tr>
<td>Par-MC-SGD</td>
<td>23.00 5.17 3.55 3.12</td>
</tr>
<tr>
<td>2mpi-Par-MC-SGD</td>
<td>12.15 3.11 2.78 2.84</td>
</tr>
</tbody>
</table>
Chapter 5. Parallel Stochastic Gradient Descent Algorithms Support Vector Machines

Figure 5.2: SVMs training time with respect to # OpenMP threads on ImageNet 100.

ILSVRC 2010. Our implementations achieve a significant speedup in training process when performing on large dataset ILSVRC 2010.

Balanced training SVM-SGD. As shown in Figure 5.3, the balanced training version of SVM-SGD (Par-MC-SGD running with 1 thread) has a very fast convergence speed in training process, it is 11 times faster than M-Pegasos and 30 times faster than LIBLINEAR (Table 5.3, row #5 col #2 vs. row #3 col #2 and row #4 col #2).

OpenMP balanced training SVM-SGD. On a multi-core machine, OpenMP version of balanced training SVM-SGD (Par-MC-SGD) achieves a significant speed-up in training process with 15 OpenMP threads. As shown in Figure 5.3, our implementation is 91 times faster than M-Pegasos and 249 times faster than LIBLINEAR (Table 5.3, row #5 col #5 vs. row #3 col #2 and row #4 col #2). Due to the restriction of our computer (16 cores), we set the maximum number of OpenMP threads to 15. We can set more than 15 OpenMP threads, but there is very few significant speedup in training process due to no more available core.

Hybrid MPI/OpenMP balanced training SVM-SGD

The most significant parallelization performance of SVM-SGD we achieve is the combination of hybrid MPI/OpenMP and balanced training SVM-SGD (mpi-Par-MC-SGD). As shown in Figure 5.3, our implementation achieves a significant performance in training process with 2 MPI processes and 15 OpenMP threads. It is 98 times faster than M-Pegasos and 270 times faster than LIBLINEAR (Table 5.3, row #6 col #5 vs. row
Chapter 5. Parallel Stochastic Gradient Descent Algorithms Support Vector Machines

Figure 5.3: SVMs training time with respect to \# OpenMP threads on ILSVRC 2010.

Table 5.3: SVMs training time (minutes) on ILSVRC 2010.

<table>
<thead>
<tr>
<th>Method</th>
<th># OpenMP threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>OCAS</td>
<td>2441.20</td>
</tr>
<tr>
<td>BudgetedSVM</td>
<td>1130.44</td>
</tr>
<tr>
<td>M-Pegasos</td>
<td>3106.48</td>
</tr>
<tr>
<td>LIBLINEAR</td>
<td>51.97</td>
</tr>
<tr>
<td>Par-MC-SGD</td>
<td>103.62</td>
</tr>
<tr>
<td>2mpi-Par-MC-SGD</td>
<td>14.11</td>
</tr>
<tr>
<td></td>
<td>11.69</td>
</tr>
<tr>
<td></td>
<td><strong>11.50</strong></td>
</tr>
</tbody>
</table>

\#3 col \#2 and row \#4 col \#2). On ILSVRC 2010, we need only 12 minutes to train 1000 binary classifiers, compared to LIBLINEAR (∼ 2 days and 4 hours), as shown in Table 5.3. This result confirms that our approach has a great ability to scale up to full ImageNet dataset with more than 21,000 classes.

5.5.3 Classification accuracy

As shown in Figure 5.4, on the small datasets ImageNet 10 and medium dataset ImageNet 100, Par-MC-SGD provides very competitive performances when compared with LIBLINEAR.

We achieve a very good classification result on ILSVRC 2010. This is a large dataset with a large number of classes (1000 classes) and a huge number of samples (more than 1 million samples), and the density of dataset is about 6% (the proportion of non-zero elements of dataset in percent). Thus, it is very difficult for many state-of-the-art SVM
solv ers to obtain a high rate in classification performance. In particular, with the feature set provided by ILSVRC 2010 competition the state-of-the-art system [11] reports an accuracy of approximately 19%, but it is far above random guess (0.1%). And now our approach provides a significantly higher accuracy rate than [11] with the same feature set (21.90% vs. 19%), as shown in Table 5.4. The relative improvement is more than 15%. Moreover, we also compare our implementation with the current state-of-the-art of linear SVM classifiers LIBLINEAR to validate our approach. As shown in Table 5.4, Par-MC-SGD outperforms LIBLINEAR (±0.79%, the relative improvement is more than 3.7%).

Note that Par-MC-SGD runs much faster than LIBLINEAR while yielding higher rate in classification accuracy.

## 5.6 Conclusion

We have developed the extended versions of SVM-SGD in several ways to efficiently deal with large scale datasets with very large number of classes like ImageNet. The primary idea is to build the balanced classifiers with a sampling strategy and a smooth hinge
loss function and then parallelize the training process of these classifiers with several multi-core computers.

Our approach has been evaluated on the 10, 100 largest classes of ImageNet and ILSVRC 2010. On ILSVRC 2010, our implementation is 270 times faster than LIBLINEAR. Therefore, we can achieve higher performances by using more resources (CPU cores, computer, etc.). Furthermore, with our sampling strategy we significantly speedup the training process of the classifiers while yielding a high performance in classification accuracy. We need only 12 minutes to train 1000 binary classifiers. However, when the training data is larger, SVM-SGD requires a large amount of main memory due to loading the whole training data into main memory. This issue will be addressed in the next step. We may study the approach as reported in [121] and another possibility is to compress the training data and handle it on the fly [77].
Chapter 6

Parallel Incremental Support Vector Machines

6.1 Introduction

In order to deal with the classification of very large datasets we need to solve two problems. The first one is the limited memory available on every computer whatever it is. Several solutions to this problem exist e.g. sampling the dataset (but this may result in losing some accuracy) or incremental learning task. Incremental learning consists in splitting the datasets in blocks (of rows or columns) that can fit into the computer memory and handle one block at a time. Some approaches like this have already been recently presented [134, 135]. However, first of all, all the learning algorithms can not be transformed in an incremental manner, they need the whole dataset to perform the classification task [28, 89, 90], so their results will be approximations compared to the original ones. Some others can perform exactly the same computation task with the same results on the same data [134], we will focus on this latter approach in this chapter. The second one is the time needed to perform the learning task, it can be up to several months or years of computing time. This has many implications e.g. the task must be able to be suspended and restarted at the breaking point (this will have an additional computing cost) or must be able to restart from any crash (whatever it is: hardware or software or network, they do happen with such a computing time, etc.). Computers architecture has evolved these last years, instead of increasing the (only one) CPU speed, they increase CPU cores at the same speed (most computers today, even mobile phones have several cores). Two main ways of taking advantage of this for parallel programming are to use OpenMP (same processes on the same computer) and MPI to use the same processes on different computers. In this chapter we will mix both of them to perform high performance algorithms in the learning task of classification of images (this is the main topic of this thesis).
A number of recent works in learning large scale classifiers for visual classification tasks have concentrated on building linear classifiers, because it is possible to train linear SVM classifiers (e.g. LIBLINEAR [102]) in order of seconds, even with millions training examples. To stick to efficient linear classifiers, the original training data in low-dimensional space is often explicitly transformed to high-dimensional space by a nonlinear mapping induced by a particular kernel function. Hence, this leads to an extreme increase of storage/main memory cost when performing on large datasets. For instance, the winners of ILSVRC 2010 report that their methods train classifiers on a dataset in hundreds of giga-bytes. However, when training data is larger and cannot fit into the main memory of computer, most existing linear classifiers encounter a problem. To solve this problem, Yu et al. [24] propose a block minimization framework for large linear classifier (LIBLINEAR-CDBLOCK), that can be applied to data beyond the memory capacity of computer. They show empirically that their method can handle data sets 20 times larger than the memory size. Nevertheless, the current version of LIBLINEAR-CDBLOCK has two main drawbacks that prevent it scaleup to large scale datasets with many classes. Firstly, for the case of multi-class classification, LIBLINEAR-CDBLOCK solves one single optimization problem by using [25] instead of one-versus-all strategy. This would still require a large amount of computer memory when both the number of classes and training samples are large. Secondly, LIBLINEAR-CDBLOCK does not take into account the benefits of HPC and thus the training time is very long.

Although linear classifiers are popular for efficiency reasons, the recent works [17, 19–21] show that in the context of visual classification based on comparing histograms of low level features, nonlinear classifiers often provide better accuracy. Wu [21] proposes a nonlinear classifier PmSVM that outperforms LIBLINEAR and other additive kernel classifiers in terms of training time and classification accuracy. However, there are two main drawbacks preventing PmSVM scaleup to large scale datasets. Firstly, PmSVM loads the whole data into memory for training classifiers. Thus, when training data is larger and cannot fit into memory, PmSVM also encounters a problem. Secondly, the current version of PmSVM does not take into account the benefits of HPC.

Obviously, the current releases of both LIBLINEAR-CDBLOCK and PmSVM do not scale well when training on large datasets with very large number of examples as well as classes. In this chapter we propose several ways in order to solve the main drawbacks of these SVM classifiers, which consists of the following contributions:

1. An incremental learning method for PmSVM. Our approach avoids loading the whole training data into memory by splitting it 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. Improve LIBLINEAR-CDBLOCK for large number of classes by using one-versus-all strategy.
3. A balanced bagging algorithm for training the incremental binary SVM classifiers. Our algorithm avoids training on full block of rows and thus the training process of these classifiers rapidly converges to the optimal solution.

4. Parallelize the training process of all incremental binary SVM classifiers based on HPC models. Moreover, in each incremental training step our balanced bagging algorithm is applied to achieve the best performance.

Our approach has been evaluated on the 100 largest classes of ImageNet and 1000 classes of ILSVRC 2010. The experiment shows that our approach can save up to 82.01% main memory usage and the training process is much faster than the original implementation and LIBLINEAR without (or very few) compromising classification accuracy. Therefore, it can be easily applied to very large datasets with many classes and training data larger than the memory capacity of the computer.

The remainder of this chapter is organized as follows. Section 6.2 briefly reviews related work on large scale classifiers for visual classification tasks. In Section 6.3 we present the incremental learning method for SVM classifiers. Section 6.4 presents its improvement for large number of classes. We describe how to speedup the training process of incremental SVMs by using our balanced bagging algorithm and taking into account the benefits of HPC. In Section 6.5 and 6.6 we present the numerical results of parallel incremental LIBLINEAR and PmSVM, accordingly. Finally, this chapter is concluded in Section 6.7.

6.2 Related Work

ImageNet makes the complexity of training classifiers much larger and very difficult to deal with. The current state-of-the-art of large scale visual classification are detailed and presented in [75, 77]. However, their method involves training classifiers on a dataset in hundreds giga-bytes. Therefore, it cannot be easily applied to the computers with limited individual memory resource. There exist two possible solutions to solve this challenge. The first one is to compress the training data and handle it on the fly [77, 127]. However, the compressed data might cause a loss of some useful information and thus downgrades the accuracy of classification. The other possibility is to apply the incremental learning method for dealing with training task. Yu et al. [24] propose LIBLINEAR-CDBLOCK that can handle data larger than the memory size of computer. Nevertheless, LIBLINEAR-CDBLOCK has two main limitations: 1) multi-class classification with one-versus-all strategy has not been studied yet, 2) it does not take into account the benefits of HPC. Therefore, the training time is very long on ILSVRC 2010 (at least 32 hours).

In contrast with the approaches using linear SVMs, the recent works show that in visual classification tasks, nonlinear SVMs are superior in terms of accuracy when compared to linear rivalries. Very recently, Wu [21] proposes an efficient algorithm for PmSVM.
They show empirically that PmSVM outperforms LIBLINEAR and state-of-the-art additive kernel SVMs in terms of training time and classification accuracy. For instance, on ILSVRC 2010, 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 2010 (at least 10 hours) due to learning 1000 binary classifiers sequentially, independently.

6.3 Incremental learning for SVM classifiers

In this section we present an incremental learning method for solving the memory usage problem of classifiers including linear and nonlinear SVMs. Our approach is inspired by the idea from the papers [134] and [24]. They show that the training task of SVM classifiers can be performed on the successive 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 (6.1) 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. The incremental learning for SVM classifiers is summarized in Algorithm 5.

**Algorithm 5:** Incremental learning for SVM classifiers

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input</strong></td>
<td>A set of training samples ( T = {(x_i, y_i)}_{i=1}^n )</td>
</tr>
<tr>
<td><strong>output</strong></td>
<td>The value ( \alpha ) or ( (w \text{ or the set of values } {a_z}) )</td>
</tr>
<tr>
<td>1</td>
<td>Split ( T ) into ( B_1, ..., B_m ) and store data in ( m ) files accordingly</td>
</tr>
<tr>
<td>2</td>
<td>( \alpha \leftarrow 0 )</td>
</tr>
<tr>
<td>3</td>
<td>( w \leftarrow 0 ) or ( a_{z,v} \leftarrow 0 ), ( 1 \leq z \leq d, \ 0 \leq v \leq 2 )</td>
</tr>
<tr>
<td>4</td>
<td>for ( j \leftarrow 1 ) to ( m ) do</td>
</tr>
<tr>
<td>5</td>
<td>Read ( x_r, \forall r \in B_j ) from disk</td>
</tr>
<tr>
<td>6</td>
<td>Solve the sub-problem (6.1) by using LIBLINEAR or PmSVM</td>
</tr>
<tr>
<td>7</td>
<td>Update ( \alpha ) or ( (w \text{ or } a_z) )</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

6.3.1 Solving dual SVM by LIBLINEAR for each block

In some applications, an SVM problem may appears with a bias term \( b \). To solve the problem, we need the constraint \( y^T \alpha = 0 \) as shown in the optimization problem (4.1). However, one often deal with this term by appending each instance with an additional dimension \( x^T \leftarrow [x^T, 1] \) and \( w^T \leftarrow [w^T, b] \). Consequently, we can remove the constraint
$y^T\alpha = 0$ and then the optimal solution $\alpha$ of (4.1) can be obtained by solving the sub-problems (6.1).

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

subject to

$$u_i = 0, \forall i \notin B_j$$

$$0 \leq \alpha_i + u_i \leq C, \forall i \in B_j$$

(6.1)

Let $u_{B_j}$ be a vector of $|B_j|$ non-zero coordinates of $u$ that correspond to the indices in $B_j$. The objective function (6.1) is equivalent to

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

(6.2)

where $Q_{B_j}$ is a sub-matrix of $Q$ including elements $Q_{ri}, r \in B_j, i = 1, \ldots, n$. Obviously, $Q_{B_j}$ in Eq. 6.2 involves all training data. This violates the method presented in Algorithm 5.

However, by maintaining $w = \sum_{i=1}^{n} \alpha_i y_i x_i$ into memory, we can compute the gradient $G$ which measures the changes in $f(\alpha + u)$ (6.1) by using the Eq. 6.3.

$$G = Q_{B_j} \alpha - 1 = y_r w^T x_r - 1, \forall r \in B_j$$

(6.3)

Suppose that $u_{B_j}^*$ is an optimal solution of (6.1). We can update $w$ by using Eq. 6.4.

$$w \leftarrow w + \sum_{r \in B_j} u_{B_j}^* y_r x_r$$

(6.4)

Clearly, this operation involves only the samples in $B_j$. The procedure for solving the sub-problem (6.1) by LIBLINEAR is summarized in Algorithm 6.

**Algorithm 6: LIBLINEAR for solving the sub-problem**

**Input**: $B_j$ from the $j^{th}$ file

The values $w$ and $\alpha$

**Output**: The updated values $w$ and $\alpha$

1. $Q_{rr} \leftarrow \|x_r\|^2, \forall r \in B_j$
2. while $\alpha$ is not optimal do
3.   for $r \leftarrow 1$ to $n_B$ do
4.     Compute $G = y_r w^T x_r - 1$ in Eq. 6.3
5.     $\alpha_{j,r} \leftarrow \alpha_{j,r}$
6.     $\alpha_{j,r} \leftarrow \min(\max(\alpha_{j,r} - G/Q_{rr}, 0), C)$
7.     $w \leftarrow w + y_r (\alpha_{j,r} - \overline{\alpha}_{j,r}) x_r$
8.   end
9. end
6.3.2 Solving dual SVM by PmSVM for each block

A nonlinear classifier maps each sample $x$ into higher dimensional vector via a nonlinear function $\phi(x)$. Therefore, once we have obtained the dual solution $\alpha$, the primal one can be calculated by using Eq. 6.5.

$$w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$$ \hspace{1cm} (6.5)

Thus, by maintaining $w$ into memory, the gradient $G$ can be computed by using Eq. 6.6.

$$G = y_r w^T \phi(x_r) - 1 = y_r \sum_{i=1}^{n} \alpha_i y_i K(x_r, x_i) - 1 = y_r g(x_r) - 1$$ \hspace{1cm} (6.6)

Substituting the power mean kernel (4.3) into Eq. 6.6, $g(x_r)$ can be approximated with a small cost $O(d)$ by using Eq. 6.7

$$g(x_r) = \sum_{i=1}^{n} \alpha_i y_i M_p(x_r, x_i) \approx \sum_{z=1}^{d} \sum_{v=0}^{2} a_{z,v} (\ln(x_{r,z} + 0.05))^v$$ \hspace{1cm} (6.7)

Therefore, if $a_z$ is available into memory then only the samples associated with block of rows $B_j$ are needed to compute $G$. We can update $a_z$ by using Eq. 6.8.

$$a_z \leftarrow a_z + \sum_{r \in B_j} u_r^* y_r X^{-1} M_p(c, x_{r,z})$$ \hspace{1cm} (6.8)

where $c = [0.01, 0.06, 0.75]$, $X$ is a regression matrix and

$$X^{-1} = \begin{pmatrix} 0.3137 & -0.5220 & 1.2083 \\ 1.5480 & -2.5249 & 0.9769 \\ 0.6369 & -0.8315 & 0.1946 \end{pmatrix}$$ \hspace{1cm} (6.9)

Obviously, the update operation of $a_z$ in Eq. 6.8 again only involves the samples in $B_j$. The procedure for solving the sub-problem (6.1) by PmSVM is summarized in Algorithm 7.

6.4 Improving incremental SVM classifiers for large number of classes

For each block of data, the incremental SVM classifiers also solve multi-class problem with a large of number classes. Therefore, one-versus-all strategy is the straightforward direction of learning tasks.
Algorithm 7: PmSVM for solving the sub-problem

input : $B_j$ from the $j^{th}$ file
The set of values $\{a_z\}$ and $\alpha$
output : The set of updated values $\{a_z\}$ and $\alpha$

1. $Q_{rr} \leftarrow \|x_r\|_{l_1}$, $\forall r \in B_j$
2. while $\alpha$ is not optimal do
   3. for $r \leftarrow 1$ to $n_{B_j}$ do
      4. Compute $G = y_r w^T \phi(x_r) - 1$ using Eq. 6.7
      5. $\alpha_{j,r} \leftarrow \alpha_{j,r}$
      6. $a_{j,r} \leftarrow \min(\max(\alpha_{j,r} - G/Q_{rr}, 0), C)$
      7. $a_z \leftarrow a_z + (\alpha_{j,r} - \alpha_{j,r}) y_r X^{-1} M_p(c, x_{r,z}), \forall z$
   end
end

LIBLINEAR-CDBLOCK. For multi-class problem, LIBLINEAR-CDBLOCK solves one single optimization problem [25] by using a dual coordinate descent method [136]. Unfortunately, this causes two major problems when performing classification on large datasets with a large number of classes (e.g. 1000 classes) as well as a huge number of samples (e.g. 1 millions samples). Firstly, for each block $B_j$, LIBLINEAR-CDBLOCK needs to maintain in memory the matrix $M$ of size the number of classes ($k$) by the number of samples in $B_j$ ($n_{B_j}$). If both $k$ and $n_{B_j}$ are large, this leads to consume a large amount of memory. Secondly, the training time is very long due to learning the classifiers sequentially and independently. However, by reducing a multi-class problem to a set of multiple binary classification problems and make use of one-versus-all strategy, we need only to keep in memory a weighting matrix $W$ of size $k \times d$ in global scope, where $W = \{w^c\}$ and $d$ is the number of attributes (or dimension) of sample. Matrix $W$ does not depend on the number of samples $n_{B_j}$, thus when $k$ and $n_{B_j}$ are very large the memory cost of maintaining matrix $W$ is clearly much less expensive than that of matrix $M$. Furthermore, learning multi-class classification model by using one-versus-all strategy can be easily parallelized with several multi-core computers. Therefore, it motivates us to improve the original implementation of LIBLINEAR-CDBLOCK by using one-versus-all strategy. The procedure of our approach is described in Algorithm 8.

PmSVM. 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 these problems, we propose two ways for speedup the learning tasks of SVM classifiers. The first one is to build the balanced bagging classifiers with under-sampling strategy. The second one is to parallelize the training task of all binary classifiers with several multi-core computers.

6.4.1 Balanced bagging incremental SVM (LIBLINEAR, PmSVM)

We propose the balanced bagging incremental SVM using under-sampling the majority class (negative class). For separating the $i^{th}$ class (positive class) from the rest (negative
Algorithm 8: The incremental LIBLINEAR with one-versus-all strategy

- **input**: Training data \( T = \{(x_i, y_i)\}_{i=1}^n \)
- **output**: The updated values of \( \alpha \) and \( w \)

1. Split \( T \) into \( B_1, ..., B_m \) and store data in \( m \) files accordingly
2. \( \alpha^c \leftarrow 0, \ w^c \leftarrow 0, \ 1 \leq c \leq k \)
3. for \( j \leftarrow 1 \) to \( m \) do
   4. Read \( x_r, \forall r \in B_j \) from disk /* block \( j \) */
   5. for \( c \leftarrow 1 \) to \( k \) do /* class \( c \) */
      6. LIBLINEAR \( (B^c_j, B_j \setminus B^c_j) \);
      7. Update \( \alpha^c \) and \( w^c \)
   8. end
9. end

Algorithm 9: Balanced bagging incremental SVM

- **input**: \( B_+ \) the training samples of the positive class in block \( B_j \), \( B_- \) the training samples of the negative class in block \( B_j \), \( T \) the number of base learners
- **output**: SVM model

1. for \( t \leftarrow 1 \) to \( T \) do
   2. 1. \( B'_- = \text{sample}(B_-) \) (with \( |B'_-| = |B_+| \))
   3. 2. SVM Solver\( (B_+, B'_-) \)
4. end
5. combine \( T \) models into the aggregated SVM model
6.4.2 Parallel incremental SVM training

Although the incremental SVM classifier and balanced bagging incremental SVM classifier deal with very large dataset with high speed, they do not take the benefits of HPC. Furthermore, both incremental and balanced bagging incremental SVM classifier trains independently $k$ binary classifiers for $k$ classes problems. This is a nice property for parallel learning. Our investigation aims at speedup the training task of multi-class incremental SVM classifier and balanced bagging incremental SVM classifier with several multi-processor computers. The idea is to learn $k$ binary classifiers in parallel way.

The hybrid approach MPI and OpenMP has demonstrated its effectiveness for many high performance computing, therefore we develop the parallel versions of incremental SVM classifiers based on this model. The hybrid MPI/OpenMP parallel incremental SVM algorithm is described in Algorithm 10. The number of MPI processes depends on the HPC system used.

**Algorithm 10:** Hybrid MPI/OpenMP parallel incremental SVM

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

output: The value $\alpha$ or $(w$ or the set of values $\{a_z\})$

1 Split $T$ into $B_1, ..., B_m$ and store data in $m$ files accordingly
2 $\alpha_c \leftarrow 0$, $1 \leq c \leq k$
3 $w^c \leftarrow 0$ or $a_{c,v} \leftarrow 0$, $1 \leq c \leq k$, $1 \leq z \leq d$, $0 \leq v \leq 2$
4 for $j \leftarrow 1$ to $m$ do
5    Read $x_r, \forall r \in B_j$ from disk /* block $j$ */
6   Learn:
7   $MPI - PROC_1$
8    #pragma omp parallel for
9    for $c_1 \leftarrow 1$ to $k_1$ do /* class $c_1$ */
10       SVM_Solver $(B^c_1, B_j \setminus B^c_1)$
11       Update $\alpha^{c_1}$ and $(w^{c_1}$ or $a^{c_1}$)
12    end
13   : 
14   $MPI - PROC_P$
15    #pragma omp parallel for
16    for $c_P \leftarrow 1$ to $k_P$ do /* class $c_P$ */
17       SVM_Solver $(B^c_P, B_j \setminus B^c_P)$
18       Update $\alpha^{c_P}$ and $(w^{c_P}$ or $a^{c_P}$)
19    end
20 end
```

6.5 Experiment 6: the parallel incremental LIBLINEAR

We have implemented three extended versions of LIBLINEAR-CDBLOCK: 1) OpenMP incremental LIBLINEAR (omp-LIBLINEAR-B), 2) OpenMP balanced bagging incremental LIBLINEAR (omp-iLIBLINEAR-B), 3) Hybrid MPI/OpenMP balanced bagging
incremental LIBLINEAR (mpi-omp-iLIBLINEAR-B).

In this section we compare our implementations with LIBLINEAR-CDBLOCK and LIBLINEAR in terms of training time, memory usage and classification accuracy.

LIBLINEAR. This is the linear SVM solver from [102] with default parameter value $C = 1$.

LIBLINEAR-CDBLOCK-B. This is the block minimization framework for LIBLINEAR [24] with parameters $C = 1$ and $s = 4$ (multi-class support vector machines by Crammer and Singer).

LIBLINEAR-B. This is the incremental LIBLINEAR with the same SVM parameters as LIBLINEAR. However, multi-class classification model is implemented by using one-versus-all approach.

iLIBLINEAR-B. This is the balanced bagging incremental LIBLINEAR.

Our experiments were run on a cluster of ten computers with the same hardware architecture as shown in Table 4.1. 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 3.2.0-4-amd64 (x86_64).

6.5.1 Datasets

The parallel incremental versions of LIBLINEAR are designed for large scale datasets, so we have evaluated our implementations on the two datasets as described in Section 4.5.1, Chapter 4.

6.5.2 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 LIBLINEAR.

ImageNet 100. We have split this dataset into 3 and 6 blocks of rows. As shown in Table 6.1, our implementation can run on the computers with main memory less than 4GB (LIBLINEAR-B-3) and less than 2GB (LIBLINEAR-B-6).

ILSVRC 2010. Due to the large size of this dataset, we have split it into 8 and 24 blocks of rows, that allows training data to fit into 4GB RAM (LIBLINEAR-B-8) and 2GB RAM (LIBLINEAR-B-24) in each incremental step. As shown in Table 6.2, LIBLINEAR and LIBLINEAR-CDBLOCK consume a large mount of memory, making them intractable on computers with limited memory resource. Although LIBLINEAR-CDBLOCK-B-3 and LIBLINEAR-B-3 have a similar cost of memory on ImageNet 100 (3.78GB and 3.71GB respectively, Table 6.1), LIBLINEAR-CDBLOCK-B-8 requires much larger memory than LIBLINEAR-B-8 when performing on large dataset ILSVRC 2010 (9.68GB and 3.23GB).
Chapter 6. Parallel Incremental Support Vector Machines

Figure 6.2: Memory usage (GB) of LIBLINEAR-B-8 on ILSVRC 2010.

Table 6.1: Memory usage (GB) of linear incremental SVM classifiers on ImageNet 100.

<table>
<thead>
<tr>
<th>Method</th>
<th>ImageNet 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>11.00</td>
</tr>
<tr>
<td>LIBLINEAR-CDBLOCK-B-3</td>
<td>3.78</td>
</tr>
<tr>
<td>LIBLINEAR-CDBLOCK-B-6</td>
<td>1.92</td>
</tr>
<tr>
<td>LIBLINEAR-B-3</td>
<td>3.71</td>
</tr>
<tr>
<td>LIBLINEAR-B-6</td>
<td>1.86</td>
</tr>
</tbody>
</table>

Table 6.2: Memory usage (GB) of linear incremental SVM classifiers on ILSVRC 2010.

<table>
<thead>
<tr>
<th>Method</th>
<th>ILSVRC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>16.70</td>
</tr>
<tr>
<td>LIBLINEAR-CDBLOCK-B-8</td>
<td>9.68</td>
</tr>
<tr>
<td>LIBLINEAR-CDBLOCK-B-24</td>
<td>7.74</td>
</tr>
<tr>
<td>LIBLINEAR-B-8</td>
<td>3.23</td>
</tr>
<tr>
<td>LIBLINEAR-B-24</td>
<td>1.29</td>
</tr>
</tbody>
</table>

respectively, Table 6.2). This phenomenon is shown with more details in Table 6.3. Obviously, these results are consistent with our analysis in Section 6.4. Therefore, by splitting data into many small blocks of rows and exploiting one-versus-all approach for multi-class problem, our approach is found to be very suitable for the user who wants to perform classification on large datasets and training data larger than memory size. As illustrated in Figure 6.3, the memory cost of our implementation (LIBLINEAR-B) are consistently stable when evaluated on different datasets. By splitting ILSVRC 2010 into 8 blocks of rows, our implementation (LIBLINEAR-B-8) can save from 66.63% to 80.66% memory usage when compared to LIBLINEAR-CDBLOCK and LIBLINEAR, as shown in Figure 6.4. 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) when running on a computer with 4GB memory, as shown in Figure 6.2.

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 6.4 and 6.5).
Chapter 6. Parallel Incremental Support Vector Machines

Figure 6.3: Memory usage (%) of linear incremental SVMs.

Table 6.3: On ImageNet 100, LIBLINEAR-CDBLOCK-B-3 and LIBLINEAR-B-3 maintain in memory a similar size of matrix. However, on ILSVRC 2010, LIBLINEAR-CDBLOCK-B-8 maintains in memory a matrix with much larger size than that of LIBLINEAR-B-8.

<table>
<thead>
<tr>
<th>Method</th>
<th>k</th>
<th>( n_{B_j} )</th>
<th>d</th>
<th>Matrix size</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR-CDBLOCK-B-3</td>
<td>100</td>
<td>33,333</td>
<td>21,000</td>
<td>3,333,333 ((k \times n_{B_j}))</td>
</tr>
<tr>
<td>LIBLINEAR-B-3</td>
<td>100</td>
<td>33,333</td>
<td>21,000</td>
<td>2,100,000 ((k \times d))</td>
</tr>
<tr>
<td>LIBLINEAR-CDBLOCK-B-8</td>
<td>1000</td>
<td>110,977</td>
<td>21,000</td>
<td>110,977,000 ((k \times n_{B_j}))</td>
</tr>
<tr>
<td>LIBLINEAR-B-8</td>
<td>1000</td>
<td>110,977</td>
<td>21,000</td>
<td>21,000,000 ((k \times d))</td>
</tr>
</tbody>
</table>

Figure 6.4: Saved memory (%) of linear incremental SVMs on ILSVRC 2010.
Chapter 6. Parallel Incremental Support Vector Machines

6.5.3 Training time

The incremental LIBLINEAR 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 Figure 6.5, on medium dataset ImageNet 100 our implementation shows a very good speedup in training process, compared to the original implementation. For instance, by splitting the dataset into 3 blocks of rows and use 10 MPI processes and 16 OpenMP threads per MPI process, our implementation (10mpi-omp-iLIBLINEAR-B-3) is 495 times faster than LIBLINEAR-CDBLOCK-B-3 (Table 6.4, row #9 col #4 vs. row #2 col #2).
ILSVRC 2010. Our implementations achieve a significant speedup in training process on this large dataset.

OpenMP incremental LIBLINEAR

On a multi-core computer, OpenMP version of incremental LIBLINEAR (omp-LIBLINEAR-B-8) achieves a significant speedup in training process with 16 OpenMP threads. As shown in Table 6.5 (row #4 col #4 vs. row #2 col #2 and row #1 col #2), our implementation is 14 times faster than the original implementation and 23 times faster than LIBLINEAR. Due to the restriction of our computer (16 cores), we set the maximum number of OpenMP threads to 16. Setting the number of threads greater than available cores do not provide better performance.

Balanced bagging incremental LIBLINEAR

As shown in Figure 6.6, by splitting ILSVR C 2010 into 8 blocks, the balanced bagging incremental LIBLINEAR (omp-iLIBLINEAR-B-8 running with 1 thread) has a very fast convergence speed in training process, it is 11 times faster than LIBLINEAR-CDBLOCK-B-8 (Table 6.5, row #6 col #2 vs. row #2 col #2).

OpenMP balanced bagging incremental LIBLINEAR

By applying balanced bagging algorithm to OpenMP version of incremental LIBLINEAR, we significantly speedup the training process of 1000 binary classifiers, as illustrated in Figure 6.6. With the number of OpenMP threads set to 16, our implementation (omp-iLIBLINEAR-B-8) is 127 times faster than LIBLINEAR-CDBLOCK-B-8 (Table 6.5, row #6 col #4 vs. row #2 col #2).

Hybrid MPI/OpenMP balanced bagging incremental LIBLINEAR

As shown in Figure 6.6, our implementation (10mpi-omp-iLIBLINEAR-B-8) achieves a significant speedup in training process by using 160 cores from ten nodes (10 MPI processes × 16 OpenMP threads). It is 732 times faster than LIBLINEAR-CDBLOCK-B-8 and 1193 times faster than LIBLINEAR (Table 6.5, row #9 col #4 vs. row #2 col #2 and row #1 col #2). We need only 2.62 minutes to train 1000 binary classifiers, compared to LIBLINEAR-CDBLOCK-B-8 (∼32 hours) and LIBLINEAR (∼52 hours). This result confirms that our approach has a great ability to scale up to full ImageNet dataset with more than 21,000 classes.
Chapter 6. Parallel Incremental Support Vector Machines

Figure 6.6: Linear incremental SVMs training time with respect to the number of OpenMP threads on ILSVRC 2010.

Table 6.5: Linear incremental SVMs training time (minute) on ILSVRC 2010.

<table>
<thead>
<tr>
<th>Method</th>
<th># OpenMP threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>LIBLINEAR</td>
<td>3126.78</td>
</tr>
<tr>
<td>LIBLINEAR-CDBLOCK-B-8</td>
<td>1917.37</td>
</tr>
<tr>
<td>LIBLINEAR-CDBLOCK-B-24</td>
<td>2533.33</td>
</tr>
<tr>
<td>omp-LIBLINEAR-B-8</td>
<td>1287.27</td>
</tr>
<tr>
<td>omp-LIBLINEAR-B-24</td>
<td>1716.00</td>
</tr>
<tr>
<td>omp-iLIBLINEAR-B-8</td>
<td>174.12</td>
</tr>
<tr>
<td>omp-iLIBLINEAR-B-24</td>
<td>210.22</td>
</tr>
<tr>
<td>5mpi-omp-iLIBLINEAR-B-8</td>
<td>38.14</td>
</tr>
<tr>
<td>10mpi-omp-iLIBLINEAR-B-8</td>
<td>23.89</td>
</tr>
</tbody>
</table>

6.5.4 Classification accuracy

As shown in Figure 6.7, on medium dataset ImageNet 100, iLIBLINEAR-B (4GB) is 1.74% worse than LIBLINEAR-CDBLOCK-B (4GB) in terms of classification accuracy. However, on large dataset ILSVRC 2010, the classification accuracy obtained by iLIBLINEAR-B (4GB) is nearly the same as LIBLINEAR-CDBLOCK-B (4GB) (19.10% vs. 19.99%, it is 0.89% worse than the original implementation) (Table 6.6).

Note that iLIBLINEAR-B runs much faster than LIBLINEAR-CDBLOCK-B. This result shows that our balanced bagging algorithm is very useful when one wants to speedup the training process of classifiers on large scale datasets without (or very few) compromising classification accuracy.
Chapter 6. Parallel Incremental Support Vector Machines

Figure 6.7: Overall classification accuracy of linear incremental SVM classifiers.

6.6 Experiment 7: the parallel incremental PmSVM

We have implemented three extended versions of PmSVM: 1) OpenMP incremental PmSVM (omp-PmSVM-B), 2) OpenMP balanced bagging incremental PmSVM (omp-iPmSVM-B), 3) Hybrid MPI/OpenMP balanced bagging incremental PmSVM (mpi-omp-iPmSVM-B).

In this section we compare our implementations with the original PmSVM and LIBLINEAR in terms of training time, memory usage and classification accuracy.

LIBLINEAR. This is the linear SVM solver from [102] with default parameter value $C = 1$.

PmSVM. This is the original PmSVM with the parameters $p = -1$ (equivalent to $\chi^2$ kernel) and $C = 0.01$.

iPmSVM. This is the balanced bagging PmSVM with the same SVM parameters as PmSVM.
Chapter 6. Parallel Incremental Support Vector Machines

Table 6.7: Memory usage (GB) of SVM classifiers on ImageNet 100.

<table>
<thead>
<tr>
<th>Method</th>
<th>ImageNet 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>11.00</td>
</tr>
<tr>
<td>PmSVM</td>
<td>13.75</td>
</tr>
<tr>
<td>PmSVM-B-3</td>
<td>3.75</td>
</tr>
<tr>
<td>PmSVM-B-6</td>
<td>1.80</td>
</tr>
</tbody>
</table>

iPmSVM-B. This is the balanced bagging incremental PmSVM with the same SVM parameters as PmSVM.

Our experiments were run on a cluster of ten computers Intel Xeon E5-2450 CPU with the same hardware architecture as shown in Table 4.1. 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 3.2.0-4-amd64 (x86_64).

6.6.1 Dataset

The extended versions of PmSVM are designed for large scale datasets, so we have evaluated our implementations on the two datasets as described in Section 4.5.1, Chapter 4.

6.6.2 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 6.7, our implementation can run on the computers with main memory less than 4GB (PmSVM-B-3) and less than 2GB (PmSVM-B-6).

ILSVRC 2010. Due to the large size of this dataset, we have split it into 8 and 24 blocks of rows, that allows training data to fit into 4GB RAM (PmSVM-B-8) and 2GB RAM (PmSVM-B-24) in each incremental step. As shown in Table 6.8, LIBLINEAR and PmSVM consume a large mount of main memory (16.70GB and 21.13GB), making them 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, by splitting the dataset into 8 blocks of rows, our implementation (PmSVM-B-8) uses only 3.80GB RAM to train 1000 classifiers on ILSVRC 2010. That means our implementation can save up to 82.01% memory usage as well (Figure 6.9), compared to the original implementation. 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) when running on a computer with 4GB memory, as shown in Figure 6.8.
Chapter 6. Parallel Incremental Support Vector Machines

6.6.3 Training time

The 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 Figure 6.10, on medium dataset ImageNet 100 our implementation shows a very good speedup in training process, compared to the original implementation (PmSVM) and LIBLINEAR. By splitting the dataset into 3 blocks of rows and use 10 MPI processes and 16 OpenMP threads per MPI process, our implementation (10mpi-omp-iPmSVM-B-3) is 70 times faster than the original implementation.
Chapter 6. Parallel Incremental Support Vector Machines

Figure 6.10: Incremental SVMs training time with respect to the number of OpenMP threads on ImageNet 100.

Table 6.9: Incremental SVMs training time (minute) on ImageNet 100.

<table>
<thead>
<tr>
<th>Method</th>
<th># OpenMP threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>LIBLINEAR</td>
<td>188.97</td>
</tr>
<tr>
<td>omp-PmSVM</td>
<td>68.08</td>
</tr>
<tr>
<td>omp-iPmSVM</td>
<td>23.94</td>
</tr>
<tr>
<td>omp-PmSVM-B-3</td>
<td>118.38</td>
</tr>
<tr>
<td>omp-PmSVM-B-6</td>
<td>136.74</td>
</tr>
<tr>
<td>omp-iPmSVM-B-3</td>
<td>51.72</td>
</tr>
<tr>
<td>omp-iPmSVM-B-6</td>
<td>59.17</td>
</tr>
<tr>
<td>5mpi-omp-iPmSVM-B-3</td>
<td>11.84</td>
</tr>
<tr>
<td>10mpi-omp-iPmSVM-B-3</td>
<td>7.67</td>
</tr>
</tbody>
</table>

and 195 times faster than LIBLINEAR (Table 6.9, row #9 col #4 vs. row #2 col #2 and row #1 col #2).

ILSVRC 2010. Our implementations achieve a significant speedup in training process on this large dataset.

OpenMP incremental PmSVM

On a multi-core computer, OpenMP version of incremental PmSVM (omp-PmSVM-B-8) achieves a significant speedup in training process with 16 OpenMP threads. As shown in Table 6.10 (row #4 col #4 vs. row #2 col #2 and row #1 col #2), our implementation is 9 times faster than the original implementation and 43 times faster than LIBLINEAR.
Table 6.10: Incremental SVMs training time (minute) on ILSVRC 2010.

<table>
<thead>
<tr>
<th>Method</th>
<th># OpenMP threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBLINEAR</td>
<td>3126.78</td>
</tr>
<tr>
<td>omp-PmSVM</td>
<td>624.56</td>
</tr>
<tr>
<td>omp-iPmSVM</td>
<td>64.41</td>
</tr>
<tr>
<td>omp-PmSVM-B-8</td>
<td>943.47</td>
</tr>
<tr>
<td>omp-PmSVM-B-24</td>
<td>969.22</td>
</tr>
<tr>
<td>omp-iPmSVM-B-8</td>
<td>130.34</td>
</tr>
<tr>
<td>omp-iPmSVM-B-24</td>
<td>146.07</td>
</tr>
<tr>
<td>5mpi-omp-iPmSVM-B-8</td>
<td>26.24</td>
</tr>
<tr>
<td>10mpi-omp-iPmSVM-B-8</td>
<td>14.87</td>
</tr>
</tbody>
</table>

Balanced bagging incremental PmSVM

As shown in Figure 6.11, by splitting ILSVRC 2010 into 8 blocks of rows, the balanced bagging incremental PmSVM (omp-iPmSVM-B-8 running with 1 thread) has a very fast convergence speed in training process, it is 5 times faster than the original implementation (Table 6.10, row #6 col #2 vs row #2 col #2).

OpenMP balanced bagging incremental PmSVM

With balanced bagging algorithm is applied to OpenMP version of incremental PmSVM, we significantly speedup the training process of 1000 binary classifiers, as shown in Figure 6.11. For instance, with the number of OpenMP threads set to 16, our implementation (omp-iPmSVM-B-8) is 61 times faster than the original implementation and 304 times faster than LIBLINEAR (Table 6.10, row #6 col #4 vs. row #2 col #2 and row #1 col #2).

Hybrid MPI/OpenMP balanced bagging incremental PmSVM

As shown in Figure 6.11, our implementation (10mpi-omp-iPmSVM-B-8) achieves a significant speedup in training process by using 160 cores from ten nodes (10 MPI processes \( \times 16 \) OpenMP threads). It is 434 times faster than the original implementation and 2171 times faster than LIBLINEAR (Table 6.10, row #9 col #4 vs. row #2 col #2 and row #1 col #2). On ILSVRC 2010, we need only 1.44 minutes to train 1000 binary classifiers, compared to the original PmSVM (\( \sim 10 \) hours) and LIBLINEAR (\( \sim 52 \) hours). This result shows that our approach has found wide spread applications in many large scale visual classification tasks.

6.6.4 Classification accuracy

As shown in Figure 6.12, in terms of classification accuracy PmSVM and iPmSVM-B outperform LIBLINEAR on medium dataset ImageNet 100, i.e. from +11.25% to
Figure 6.11: Incremental SVMs training time with respect to the number of OpenMP threads on ILSVRC 2010.

Table 6.11: Overall classification accuracy (%).

<table>
<thead>
<tr>
<th>Method</th>
<th>ImageNet 100</th>
<th>ILSVRC 2010</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>omp-PmSVM</td>
<td>55.77</td>
<td>25.64</td>
</tr>
<tr>
<td>omp-iPmSVM</td>
<td>54.54</td>
<td>25.35</td>
</tr>
<tr>
<td>PmSVM-B (2GB)</td>
<td>54.17</td>
<td>24.61</td>
</tr>
<tr>
<td>PmSVM-B (4GB)</td>
<td>54.45</td>
<td>25.01</td>
</tr>
<tr>
<td>iPmSVM-B (2GB)</td>
<td>54.13</td>
<td>24.55</td>
</tr>
<tr>
<td>iPmSVM-B (4GB)</td>
<td>54.42</td>
<td>24.95</td>
</tr>
</tbody>
</table>

+12.60% (Table 6.11, row #8 col #2 and row #2 col #2 vs. row #1 col #2). The relative increase is more than 26.05%. We achieve the same performance picture when performing on very large dataset ILSVRC 2010, i.e. from +3.84% to +4.53% (Table 6.11, row #8 col #3 and row #2 col #3 vs. row #1 col #3). The relative improvement is more than 18.19%.

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

6.7 Conclusion

In this chapter, we have developed the extended versions of the state-of-the-art SVM classifiers including linear and nonlinear classifier in these following ways: (1) an incremental
This kind of approach is interesting whatever the dataset size is, we can split it into blocks according to the computer memory size and perform the training task. This solves the space complexity problem when dealing with large datasets. We solve the time complexity problem by using several cores/computers simultaneously to perform the training task in a reasonable amount of time.
Chapter 7

Conclusion and Future Work

7.1 Conclusion

This dissertation has made two major contributions for large scale visual classification. The first one is image representation. We have proposed a novel method of combination multiple of different features for image classification task (Chapter 3). The other is large scale learning algorithms. In particular, we have developed the parallel versions of both state-of-the-art linear and nonlinear SVM classifiers (Chapter 4). In addition, we have proposed a novel algorithm to extend stochastic gradient descent SVM for large scale learning on large datasets (Chapter 5). Especially, a class of large scale incremental SVM classifiers have been developed in order to perform classification tasks on large datasets with very large number of classes and training data that can not fit into memory any more (Chapter 6).

7.1.1 Multi-feature and Multi-codebook

We have proposed a fast and efficient framework for large scale image classification and show how to address this challenge by using ImageNet dataset as an example. In this framework, we have proposed a novel approach using several different local features simultaneously to improve the classification accuracy on a large scale image dataset (the relative increase is up to 82%). To speedup the process of extracting features, we have presented how to use a multi-core computer to reduce the computation time of feature extraction.

7.1.2 Parallel Balanced Bagging Support Vector Machines

We have developed the extended versions of SVM classifiers (PmSVM and LIBLINEAR) to efficiently deal with large scale datasets with very large number of classes like ImageNet. To speedup the training process of the SVM classifiers, we have proposed two
effective ways. The first one is to build the balanced bagging classifiers with under-
sampling strategy. Our algorithm avoids training on full training data, so the training
process of binary classifiers rapidly converges to the optimal solution. The second one
is to parallelize the training process of all classifiers with several multi-core comput-
ers. We have developed the parallel versions of SVM classifiers based on HPC models
(OpenMP, MPI, and hybrid MPI/OpenMP). In each parallel version of SVM classifiers,
our balanced bagging algorithm is applied to obtain the best performance in the training
process.

Our approach has been evaluated in terms of training time and classification accuracy
on a large dataset ImageNet. On ILSVRC 2010, our implementation is thousand times
faster than the original implementation. To obtain this result, we have set the number of
processes to 160 on our computers. Therefore, we can get better performance by using
more resources (CPU cores, computer, etc.). Furthermore, by applying the balanced
bagging approach we significantly speedup the training process of the classifiers without
(or very few) compromising the overall classification accuracy. For the case of linear
classifier, our implementation needs only 7.64 minutes to train 1000 binary classifiers;
and especially for the case of nonlinear classifier it needs only 1.52 minutes. Obviously,
this is a roadmap towards very large scale visual classification. However, when the
training data is larger and cannot fit into main memory, both LIBLINEAR and PmSVM
encounter a problem. To solve this problem, we may consider two following possible
solutions. The first one is to study the approach that avoids loading the whole training
data into main memory, as in [121]. The other is to compress the training data and
handle it on the fly, as in [77].

7.1.3 Parallel Stochastic Gradient Descent Algorithms Support Vector
Machines

We have developed the extended versions of SVM-SGD in several ways to efficiently deal
with large scale datasets with very large number of classes like ImageNet. The primary
idea is to build the balanced classifiers with a sampling strategy and a smooth hinge
loss function and then parallelize the training process of these classifiers with several
multi-core computers.

Our approach has been evaluated on the 10, 100 largest classes of ImageNet and ILSVRC
2010. On ILSVRC 2010, our implementation is 270 times faster than LIBLINEAR.
Therefore, we can achieve higher performances by using more resources (CPU cores,
computer, etc.). Furthermore, with our sampling strategy we significantly speedup the
training process of the classifiers while yielding a high performance in classification ac-
curacy. We need only 12 minutes to train 1000 binary classifiers. However, when the
training data is larger, SVM-SGD requires a large amount of main memory due to loading
the whole training data into main memory.
7.1.4 Parallel Incremental Support Vector Machines

We have developed the extended versions of the state-of-the-art SVM classifiers including linear and nonlinear classifier in three ways: (1) an incremental learning for SVM classifiers, (2) a balanced bagging algorithm for training binary classifiers, (3) parallelize the training process of these classifiers with several multi-core computers. Our approach has been evaluated on the 100 largest classes of ImageNet and ILSVRC 2010. Experiments show that: 1) in the case of linear classifier (LIBLINEAR-CDBLOCK), our implementation can save up to 66.63% memory usage and the training process is 732 times faster than the original implementation and 1193 times faster than LIBLINEAR. We need only 2.62 minutes to train 1000 binary classifiers, 2) in the case of nonlinear classifier (PmSVM), our implementation can save up to 82.01% memory usage and the training process is 434 times faster than the original implementation and 2171 times faster than LIBLINEAR. We need only 1.44 minutes to train 1000 binary classifiers. Therefore, our approach can be easily applied to datasets with very large number of classes and the training data larger than the memory capacity of computer. Obviously, this is a roadmap towards large scale visual classification for the systems with limited individual resource.

7.2 Future Work

Although recent years have witnessed a lot of research efforts on how to reduce the well-known semantic gap between human-understandable high-level semantics and machine-generated low-level features, the study on large scale visual classification is still attractive and calls for more sophisticated and scalable algorithms. According to the results found in this dissertation, we propose the following future directions.

7.2.1 Image representation

The performance of recognizing thousands of classes strongly depend on the powerful and efficient image representations. This challenge is still a very interesting problem and needs to be studied systematically and carefully. We plan to study how to combine effectively the global features (e.g. contour, texture, shape, etc.) with the local features to get more discriminative power of image representation. Encoding spatial information of the interesting keypoints of image will be also studied. Another promising direction is to study a high-level image representation [137] for large scale visual classification tasks.

7.2.2 Large scale classifier

The next step is to perform the incremental SVM classifiers on 10,000 classes of ImageNet. With this large dataset, the training data would be much larger than the capacity of many existing HPC systems. However, by partitioning the training data into many
blocks we can easily solve the memory usage problem of SVM training. To improve the accuracy of classification, we are looking for image signature methods that have a good result with kernel versions of SVMs in large scale setup. In order to answer the question of what is a good way to keep online training samples and support vectors into memory, the integration of budgeted online SVM algorithms [133, 138] with selecting online informative training samples [139, 140] will be studied in both linear and nonlinear SVMs. We hope that this approach not only reduces the memory requirements, speedups the convergence process but also improves the accuracy performance of SVM classifiers.

For the memory problem, this dissertation provides a decomposition able to iterate over independent block of data. Thus, a special note on a potential for parallelism at this level is marked, i.e. the training process of incremental SVM classifiers may be parallelized at disk-level. However, parallel training SVM models on distributed blocks of data is certainly much more complicated than Algorithm 5.

Incremental learning algorithms that process one sample at a time have shown the advantages when dealing with very large datasets. SVM-SGD is such an algorithm and becomes a recently popularized approach [28, 141, 142] for large scale SVM training. SVM-SGD solves the primal form of SVM and the primal variable $w$ does not correspond to data samples, therefore we cannot use the same setting as Algorithm 5 in Chapter 6 to have a sub-problem like (6.1). However, SVM-SGD possesses the nice property that at each step only certain data points are used. This points out another direction of our research, with the extensions of incremental learning method for multi-class SVM-SGD will be explored. Moreover, the application of our approach to SVM-SGD equipped with kernel functions would be a very promising research.

The focus of this thesis was visual classification, the application domains of SVM is much larger than that. They are widely used in consumer analytics, health care, retail, banking, social media and may other applications. In all these applications the size of the data keep on growing and growing. The large scale linear and non linear SVMs algorithms, presented in Chapter 4, 5 and 6 could be used on almost any kind of large datasets. So all the application domains cited previously could benefit from these large scale SVM algorithms in the new era of big data.
Résumé étendu

Introduction

La plupart des méthodes de classification d’images actuelles comportent trois étapes principales : 1) l’extraction de caractéristiques locales de bas niveau dans l’image, 2) la construction d’un sac de mots visuels et 3) l’apprentissage par un algorithme de classification. L’évaluation se fait ensuite sur des ensembles de données tels que Caltech 101, Caltech 256 ou PASCAL VOC de tailles suffisamment restreinte pour tenir en mémoire vive. Mais l’arrivée de nouveaux ensembles de données tels qu’ImageNet (21841 classes et 14,2 millions d’images) pose de nouveaux challenges. Avec une telle quantité d’images, l’exécution d’un algorithme de classification efficace peut nécessiter plusieurs semaines voire mois. Les algorithmes d’apprentissage les plus utilisés dans ce cas sont les algorithmes de (Support Vector Machine) SVM linéaires, pour leur rapidité, mais les SVM linéaires donnent de moins bons résultats que les non-linéaires dans le cas de la classification d’images. nous présentons donc des améliorations de ces algorithmes qui tirent parti des architectures matérielles d’aujourd’hui : les machines multi cœur / processeurs / grilles de calcul, pour améliorer leurs performances. Ce manuscrit est organisé comme suit.

Dans le chapitre deux, nous présentons en détail les différentes étapes du processus de classification d’images. Pour chaque étape, nous présentons les méthodes de l’état de l’art et leurs récentes améliorations. Les principales approches pour la classification d’images à grande échelle sont passées en revue afin de bien situer les apports de nos travaux. Le chapitre trois présente notre approche multi descripteurs et multi sac de mots visuels. Un nouvel algorithme parallèle de bagging équilibré de SVMs est présenté dans le chapitre quatre. Le chapitre cinq détails la mise en œuvre d’algorithmes parallèles de SVM à descente de gradient stochastique pour les ensembles de données de grande taille. Enfin, nous présentons un algorithme incrémental et parallèle pour la classification d’images à grande échelle à l’aide de SVM dans le chapitre six avant la conclusion et les travaux futurs dans le chapitre sept.

Chapitre 2

Dans ce chapitre, nous présentons les différentes étapes d’un système de classification d’images. Pour chaque étape, nous détaillons les principales approches utilisées pour les principaux composants du système. Nous commençons par la présentation du processus global de classification d’images. Puis nous étudions en détail les trois principales étapes afin de montrer la complexité des méthodes de classification d’images. Les ensembles de données d’images couramment utilisés pour la classification à grande échelle sont aussi présentés dans ce chapitre.

Chapitre 3

Dans les travaux récents le modèle de sac de mots (visuels) est à la base d’avancées significatives en classification d’images. L’ingrédient clé est de concevoir un système de classification d’images performant, qui détermine les spécificités d’une classe tout en permettant une grande variation intra classe. Pour un ensemble d’images test il faut alors pouvoir prédire avec la plus grande précision possible l’appartenance à la classe de chaque image. Les travaux précédents dans le domaine ont montré des résultats de qualité sur des ensembles de données relativement simples,
avec un petit nombre de classes et une faible variation intra classe. La plupart de ces approches utilisent des descripteurs d’images de bas niveau comme les SIFT (Scale Invariant Feature Transform), les SURF (Speed Up Robust Features) ou les SIFTs denses (DSIFT). Ils sont donc basés sur une représentation de bas niveau et locale des images alors que d’autres informations comme la couleur ou la texture pourraient aussi être efficaces pour un système de classification d’images.

L’apparition récente d’ensembles de données tel qu’ImageNet, avec plusieurs milliers de classes (avec en moyenne 1000 images par classe) et une très forte variabilité intra classe implique que la classification d’un tel nombre de classes demande une représentation efficace des caractéristiques des images. De plus la classification d’un tel nombre d’images impose aussi que les méthodes soient efficaces dans un temps raisonnable. Dans ce chapitre nous nous attaquons à ces deux problèmes et obtenons des résultats prometteurs par rapport aux approches de l’état de l’art.

Notre approche consiste à utiliser simultanément plusieurs caractéristiques de bas niveau des images pour créer autant de sacs de mots visuels. Pour réduire le temps d’exécution, nous proposons une parallélisation des processus d’extraction des caractéristiques de bas niveaux des images, de créations des sacs de mots visuels et d’encodage de la représentation des images.

Notre approche a été validée sur les 10 et 100 plus grandes classes de l’ensemble de données ImageNet. Dans le cas de l’apprentissage de LIBLINEAR avec la combinaison de trois descripteurs de bas niveau (DSIFT, SIFT et SURF) nous améliorons significativement la précision (jusque +33.01%) comparé à l’utilisation d’un seul descripteur SIFT. En combinant DSIFT, CENTRIST et SOBEL, avec cette fois-ci la classification utilisant l’algorithme de PmSVM (Power Mean SVM) nous obtenons une amélioration de la précision de 2.65% et 3.89%. Ces premiers résultats montrent l’intérêt de notre approche pour obtenir une meilleure précision de la tâche de classification d’images. Nous étudierons aussi s’il est possible de combiner efficacement des caractéristiques globales des images (comme les textures, formes ...) avec des caractéristiques locales pour améliorer le pouvoir discriminant d’une telle représentation des images. L’encodage d’information spatiale des points d’intérêt est aussi une voie à étudier.

**Chapitre 4**

Dans la troisième étape du processus de classification d’images, l’apprentissage d’un classifieur, dans la plupart des cas, c’est un algorithme de SVM à noyau linéaire ou non linéaire qui est utilisé car il ne prend que quelques minutes sur les petits ensembles de données et donne de bons résultats.

Mais des ensembles de données tels qu’ImageNet avec un très grand nombre de classes et d’images sont un challenge pour la tâche d’apprentissage. Avec des millions d’individus ou dimensions l’apprentissage d’un classifieur efficace peut alors nécessiter des semaines voire des années de calcul [11, 75]. Les travaux récents en classification d’images à grande échelle convergent vers l’utilisation de classifieurs linéaires car il est possible alors d’effectuer l’apprentissage de SVM linéaires (par exemple LIBLINEAR) en quelques secondes même avec des millions d’individus. Pour obtenir des résultats de qualité, les données originales sont alors transformées explicitement dans un espace de dimension augmentée par une transformation non linéaire à l’aide d’une fonction de noyau particulière. Bien que LIBLINEAR soit une des références pour l’apprentissage rapide de SVM linéaire, il demande un très long temps d’apprentissage sur l’ensemble de données.
Résumé étendu

ILSVRC 2010 (1000 classes d’ImageNet). En effet, il doit dans ce cas effectuer 1000 apprentissages (un par classe) séquentiellement. Il est donc intéressant d’en proposer une parallélisation.

D’un autre côté des articles [17, 19, 21] on a montré que, dans le contexte de la classification d’images, les classifi cateurs linéaires sont moins performants, en terme de précision, que leur version non-linéaire. Wu [21] a proposé un algorithme de classification non-linéaire, le Power Mean SVM (PmSVM) qui a de meilleures performances que les algorithmes linéaires ou à noyau additif, à la fois en ce qui concerne le temps de classification et la précision. Cependant la version courante de PmSVM nécessite tout de même un temps d’apprentissage conséquent sur des ensembles de données tels que ILSVR C 2010. Nous nous proposons d’étudier comment accélérer l’apprentissage de PmSVM pour la classification d’images à grande échelle.

Dans ce chapitre nous présentons des améliorations des algorithmes de l’état de l’art incluant LIBLINEAR et PmSVM pour la classification d’images à grande échelle. Nos contributions sont :

1. un algorithme de bagging équilibré pour l’apprentissage, cet algorithme évite l’apprentissage de l’ensemble de données complet et donc la tâche d’apprentissage de SVM converge rapidement vers la solution optimale.

2. la parallélisation du processus d’apprentissage des classifi cateurs basée sur le modèle de calcul haute performance (High Performance Computing – HPC). Dans l’étape d’apprentissage nous utilisons cette version de notre algorithme de bagging équilibré pour obtenir de meilleures performances.

Notre approche est évaluée, à la fois en ce qui concerne le temps de calcul et la précision, sur un ensemble de données basé sur ImageNet. L’apprentissage sur ILSVR C 2010 est plus de 2000 fois plus rapide que la version originale. Pour obtenir ce résultat nous n’avons utilisé que 160 cœur pour le calcul, donc nous pouvons aisément améliorer ces performances en utilisant plus de ressources (machines / cœur). De plus, en utilisant notre algorithme de bagging équilibré, nous améliorons de manière significative la durée d’apprentissage sans compromis (ou très peu) sur la précision du modèle. Dans le cas de classifieur linéaire l’apprentissage des 1000 cas binaires nécessite seulement 7,64 minutes et dans le cas non linéaire seulement 1,52 minutes (contre 10h pour la version originale de PmSVM et un peu plus de trois jours pour LIBLINEAR).

On peut noter que, dans l’approche un contre le reste, plus la classe correspondant au reste est grande par rapport à la classe courante, plus notre algorithme de bagging équilibré est rapide par rapport à l’approche non équilibrée. Cette une propriété intéressante dans le cas de l’ensemble de données ImageNet, qui contient plus de 21000 classes. Par ailleurs la précision pourrait être sans doute améliorée avec l’utilisation de méthodes d’échantillonnages plus performantes et sophistiquées [122-124].

Cependant, dans le cas où les données ne peuvent plus tenir en mémoire, LIBLINEAR comme PmSVM se trouvent confrontés à un problème. Pour y remédier nous pouvons envisager au moins deux solutions. La première est celle qui consiste à éviter de charger l’ensemble des données en
mémé vive, comme dans l’approche [121], la seconde est de compresser les données et les décompresser à la volée lors de l’apprentissage comme dans [77].

Chapitre 5

L’apprentissage d’un classifieur rapide et précis demeure un challenge dans le cadre de la classification d’images à grande échelle. Comme nous l’avons vu, l’apprentissage d’un classifieur linéaire est une des solutions la plus souvent préférée [125] car elle présente le meilleur compromis entre le temps d’apprentissage et la précision. Shalez-Shwartz et al. [28] et Bottou et al. [126] utilisent des algorithmes de descente de gradient stochastique pour les SVM (appelé SVM-SGD) qui donnent des résultats prometteurs. Récemment une extension de SVM-SGD [127] a été proposée pour la classification d’images à grande échelle, mais elle ne tire pas avantage des particularités des architectures récentes. Nous proposons donc des extensions de cette approche, l’idée étant d’en faire une version équilibrée (dans l’approche un contre le reste la classe minoritaire ne représente que 0.1% des données pour ILSVR C 2010) et de paralléliser l’apprentissage.

L’évaluation sur l’ensemble de données ILSVR C 2010 montre que notre approche est 270 fois plus rapide que LIBLINEAR, mais l’algorithme de SVM-SGD nécessite une très grande quantité de mémoire pour pouvoir s’exécuter (plus de 10Go pour ILSVR C), ce qui le rend impossible à utiliser sur des machines courantes.

Chapitre 6

Pour le passage à l’échelle des algorithmes de classification, nous devons résoudre deux problèmes : la complexité spatiale et la complexité temporelle. Le premier est lié à la quantité de mémoire vive disponible et limitée quelle que soit la machine utilisée. Plusieurs solutions peuvent être envisagées, par exemple l’échantillonnage de l’ensemble de données (mais cela peut avoir comme conséquence une baisse de la précision de l’algorithme de classification) ou un apprentissage incrémental. L’apprentissage incrémental consiste à partitionner l’ensemble de données en plusieurs blocs (de lignes ou colonnes) tels qu’un bloc peut tenir en mémoire vive et à traiter un bloc après l’autre. Cependant tous les algorithmes de classification ne peuvent être ainsi transformés, certains nécessitent tout l’ensemble de données pour effectuer l’apprentissage [28, 89, 90], donc leurs résultats ne seront que des approximations de l’algorithme original. Quelques algorithmes peuvent cependant effectuer exactement le même calcul par blocs [134].

La seconde est le temps d’apprentissage qui peut atteindre des semaines ou des années de calcul.

Les vainqueurs d’ILSVRC 2010 ont ainsi utilisé une méthode nécessitant plusieurs centaines de Go de mémoire vive. Pour résoudre ce problème Yu et al. [24] ont proposé une classification par blocs (LIBLINEAR-CDBLOCK) qui permet de dépasser les capacités mémoire de la machine utilisée mais qui utilise une méthode d’optimisation [23] au lieu de l’approche un contre le reste. Cette méthode n’est efficace que pour des ensembles de données jusqu’à 20 fois la capacité mémoire de la machine utilisée, mais ce n’est pas suffisant dans notre cas. Pour cela nous proposons plusieurs améliorations de ces algorithmes :

1. un apprentissage incrémental de PmSVM, qui divise l’ensemble de données initial en blocs de lignes et les traitent les uns après les autres,
2. une version un contre le reste de LIBLINEAR-CDBLOCK,
3. un algorithme de bagging équilibré pour l’apprentissage de SVM incrémental,
4. une parallélisation des algorithmes d’apprentissage incrémentaux.

Les expérimentations sur les 100 plus grandes classes d’ImageNet et ILSVRC 2010 montrent que dans le cas de la version incrémentale de LIBLINEAR, nous gagnons de 65.63% à 80.66% d’utilisation mémoire tout en étant 732 fois plus rapide que la version originale de LIBLINEAR-CDBLOCK et 1193 fois plus rapide que LIBLINEAR. En ce qui concerne le PmSVM parallèle et incrémental, nous gagnons 82.01% d’occupation mémoire tout en étant 434 fois plus rapide que le PmSVM et 2171 fois plus rapide que LIBLINEAR.

En ce qui concerne la précision, nous passons de 43.17% (version originale de LIBLINEAR) à 54.45% (PmSVM en blocs) sur les 100 plus grandes classes d’ImageNet et de 21.11% (version originale de LIBLINEAR) à 25.01% (PmSVM en blocs) sur ILSVRC 2010 (soit des améliorations relatives de 26.05% et 18.19%).

Conclusion et perspectives

Dans ce manuscrit nous avons présenté deux contributions majeures :

- une combinaison de plusieurs descripteurs pour la classification d’images à grande échelle,
- le développement de versions incrémentales et parallèles d’algorithmes de classification.

Dans le chapitre 3, nous avons présenté une nouvelle méthode pour la combinaison de différentes caractéristiques de bas niveaux pour la classification d’images. Dans le chapitre suivant nous nous sommes intéressé à des versions parallèles d’algorithmes de l’état de l’art tels que LIBLINEAR et PmSVM. Nous avons, de plus proposés (chapitre 5) un algorithme de SVM avec descente de gradient stochastique pour la classification d’images à grande échelle. Plus spécialement un algorithme incrémental et parallèle de classification (chapitre 6) est présenté lorsque les données ne peuvent plus tenir en mémoire vive.

Le thème de cette thèse est la classification d’images, nous nous sommes donc concentrés sur ce sujet, mais les domaines d’applications des SVMs sont plus larges que cela. Ils sont utilisés entre autres dans les domaines du suivi de clientèle, la prévention de la santé, la banque, les médias sociaux etc. Dans toutes ces applications, la taille des données ne cesse de croître. Les algorithmes présentés dans les chapitres 4, 5 et 6 peuvent bien sûr être utilisés avec n’importe quel type de grands ensembles de données, donc tous les domaines précédés peuvent donc bénéficier des améliorations des SVMs présentées dans la nouvelle ère du Big Data.
List of Publications


Bibliography


