Large-scale Image Classification: Fast Feature Extraction and SVM Training

Yuanqing Lin, Fengjun Lv, Shenghuo Zhu, Ming Yang, Timothee Cour and Kai Yu
NEC Laboratories America, Cupertino, CA 95014

Liangliang Cao and Thomas Huang
Beckman Institute, University of Illinois at Urbana-Champaign, IL 61801

Abstract

Most research efforts on image classification so far have been focused on medium-scale *datasets, which are often defined as datasets that can fit into the memory of a desktop (typically 4G∼48G). There are two main reasons for the limited effort on* large-scale *image classification. First, until the emergence of ImageNet dataset, there was almost no publicly available large-scale benchmark data for image classification. This is mostly because class labels are expensive to obtain. Second, large-scale classification is hard because it poses more challenges than its medium-scale counterparts. A key challenge is how to achieve efficiency in both feature extraction and classifier training without compromising performance. This paper is to show how we address this challenge using ImageNet dataset as an example. For feature extraction, we develop a Hadoop scheme that performs feature extraction in parallel using hundreds of mappers. This allows us to extract fairly sophisticated features (with dimensions being hundreds of thousands) on 1.2 million images* within one day. *For SVM training, we develop a parallel* averaging stochastic gradient descent *(ASGD) algorithm for training one-against-all 1000-class SVM classifiers. The ASGD algorithm is capable of dealing with terabytes of training data and converges very fast – typically 5 epochs are sufficient. As a result, we achieve state-of-the-art performance on the ImageNet 1000-class classification,* i.e.*,* 52.9% *in classification accuracy and* 71.8% *in top 5 hit rate.*

1. Introduction

It is needless to say how important of image classification/recognition is in the field of computer vision – image recognition is essential for bridging the huge semantic gap between an image, which is simply a scatter of pixels to untrained computers, and the object it presents. Therefore, there have been extensive research efforts on developing effective visual object recognizers [10]. Along the line, there are quite a few benchmark datasets for image classification, such as MNIST [1], Caltech 101 [9], Caltech 256 [11], PASCAL VOC [7], LabelMe[19], etc. Researchers have developed a wide spectrum of different local descriptors [17, 16, 5, 22], bag-of-words models [14, 24] and classification methods [4], and they compared to the best available results on those publicly available datasets – for PASCAL VOC, many teams from all over the world participate in the PASCAL Challenge each year to compete for the best performance. Such benchmarking activities have played an important role in pushing object classification research forward in the past years.

In recent years, there is a growing consensus that it is necessary to build general purpose object recognizers that are able to recognize many different classes of objects – e.g. this can be very useful for image/video tagging and retrieval. Caltech 101/256 are the pioneer benchmark datasets on that front. Newly released ImageNet dataset [6] goes a big step further, as shown in Fig. 1 – it further increases the number of classes to 1000[1], and it has more than 1000 images for each class on average. Indeed, it is necessary to have so many images for each class to cover visual variance, such as lighting, orientation as well as fairly wild appearance difference within the same class – like different cars may look very differently although all belong to the same class.

However, compared to those previous medium-scale datasets (such as PASCAL VOC datasets and Caltech101&256, which can fit into desktop memory), large-scale ImageNet dataset poses more challenges in image classification. For example, those previous datesets

[1]The overall ImageNet dataset consists of 11,231,732 labeled images of 15589 classes by October 2010. But here we only concern about the subset of ImageNet dataset (about 1.2 million images) that was used in 2010 ImageNet Large Scale Visual Recognition Challenge

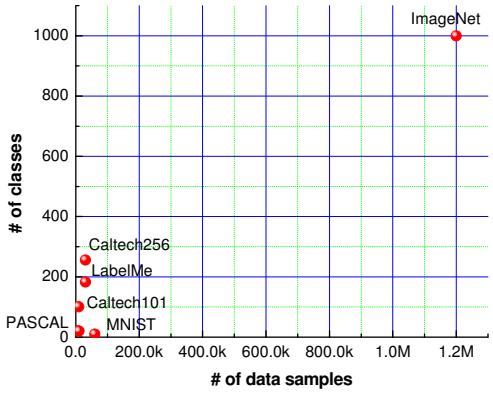

Figure 1. The comparison of ImageNet dataset with other benchmark datasets for image classification. ImageNet dataset is significantly larger in terms of both the number of data samples and the number of classes.

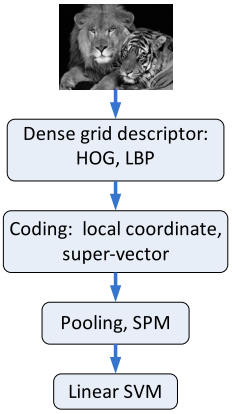

Figure 2. The overview of our large-scale image classification system. This system represents an image using a bag-of-words (BoW) model and performs classification using a linear SVM classifier. Given an input image, the system first extracts dense local descriptors, HOG [5] or LBP (local binary pattern [22]). Then, each local descriptor is coded either using local coordinate coding (LCC) [26] or Gaussian model supervector coding [28]. The codes of the descriptors are then passed to weighted pooling or max-pooling with spatial pyramid matching (SPM) to form a vector for representing the image. Finally, the feature vector is fed to a linear SVM for classification.

have at most 30,000 or so images, and it is still feasible to exploit kernel methods for training nonlinear classifiers, which often provide state-of-the-art performance. In contrast, the kernel methods are prohibitively expensive for ImageNet dataset that consists of 1.2 million images. Therefore, a key new challenge for the ImageNet large-scale image classification is how to efficiently extract image features and train classifiers without compromising performance. This paper is to show how we address the challenge and achieve so far the state-of-the-art classification performance on the ImageNet dataset.

The major contribution of this paper is to show how to train an image classification system on large-scale datasets in a system level. We develop a fast feature extraction scheme using Hadoop [21]. More importantly, following [23], we develop a parallel averaging stochastic gradient descent (ASGD) algorithm with proper step size scheduling to achieve fast SVM training.

2. Classification system overview

For ImageNet large-scale image classification, we employ a classification system shown in Fig. 2. This system follows the approaches described in a number of previous works [24, 28] that showed state-of-the-art performance on medium-scale image classification datasets (such as PASCAL VOC and Caltech101&256). Here, we attempt to integrate the advantages from those previous systems. The contribution of this paper is not to propose a new classification paradigm but to develop efficient algorithms to gain similar performance on *large-scale* ImagetNet dataset as those achieved by the state-of-the-art methods on medium-scale datasets.

Extending the methods for medium-scale datasets to large-scale imageNet dataset is not easy. For the reported best performers on the medium-scale datasets [28, 24], extracting image features on one image takes at least a couple of seconds (and even minutes [24]). Even if it takes 1 second per image for feature extraction, in total it would take 1.2×10^6 seconds $\approx$ 14 days. Even more challenging is SVM training. Let's use PASCAL VOC 2010 for comparison. The PASCAL dataset consists of about 10,000 images in 20 classes. To our experience, training SVM for this PASCAL dataset (e.g. using LIBLINEAR [8]) would take more than 1 hour if we use the features that are employed in those state-of-the-art methods (without dimensionality reduction, e.g., by kernel trick). This means we would need at least $1 \times 50 \times 120$ hours $= 250$ days in computation – not counting the often most painful part, memory constraints and file I/O bottlenecks. Indeed, we need new thinking on existing algorithms: mostly, more parallelization and efficiency for computation, and faster convergence for iterative algorithms, particularly, SVM training. In the following two sections, Section 3 and Section 4, we will show how to implement the new thinking into image feature extraction and SVM training, which are the two major functional blocks in our classification system (as shown in Fig. 2).

3. Feature extraction

As shown in Fig. 2, given an input image, our system first extracts dense HOG (histogram of oriented gradients [5])

and LBP (local binary pattern [22]) local descriptors. Both features have been proven successful in various vision tasks such as object classification, texture analysis and face recognition, etc. HOG and LBP are complementary in the sense that HOG focuses more on shape information while LBP emphasizes texture information within each patch. The advantage of such combination was also reported in [22] for human detection task. For images with large size, we downsize them to no more than 500 pixels at either side. Such normalization not only considerably reduces computational cost, but more importantly, makes the representation more robust to scale difference. We used three scales of patch size for computing HOG and LBP, namely, 16×16, 24×24 and 32×32. The multiple patch sizes provide richer coverage of different scales and make the features more invariant to scale changes.

After extracting dense local image descriptors, denoted by $z \in \mathbb{R}^d$, we perform the 'coding' and 'pooling' steps, as shown in Fig. 2, where the coding step encodes each local descriptor z via a nonlinear feature mapping into a new space, then the pooling step aggregates the coding results fallen in a local region into a single vector. We apply two state-of-the-art 'coding + pooling' pipelines in our system, one is based on local coordinate coding (LCC) [26], and the other is based on super-vector coding (SVC) [28]. For simplicity, we assume the pooling is global. But spatial pyramid pooling is simply implemented by applying the same operation independently within each partitioned block of images.

3.1. Local Coordinate Coding (LCC)

Let $B = [b_1, \ldots, b_p] \in \mathbb{R}^{d\times p}$ be the codebook, where d is the dimensionality of descriptors z and p is the size of the codebook. Like many coding methods, LCC seeks a linear combination of bases in B to reconstruct z, namely $z \approx B\alpha$, and then use the coefficients α as the coding result for z. Typically α is sparse and its dimensionality is higher than that of z. We note that the mapping $\phi(z)$ from z to α is usually nonlinear. The theory of LCC points out that, under a mild manifold assumption, a good coding should satisfy two properties:

- The approximation $z \approx B\alpha$ is sufficiently accurate;
- The coding α should be sufficiently local – only those bases close to z are activated;

Based on the theory, we develop a very simple algorithm here. We first use K-means algorithm to learn a codebook B and then for encoding z do the following:

1. Ensure sufficient locality: find z's κ nearest neighbors in B, typically $\kappa = 20$, and denote the found bases as $B_z \in \mathbb{R}^{d\times\kappa}$;
2. Ensure tight approximation: solve the optimization problem

$$\min_{\alpha_z} \|z - B_z\alpha_z\|^2, \quad \text{subject to } \alpha_z^\top e = 1, \quad (1)$$

where e is a vector of ones. The problem has a closed form solution.

Then the coding result $\alpha \in \mathbb{R}^p$ of z is obtained by placing the elements of α_z into the corresponding positions of a p-dimensional vector and leaving the rest to be zeros. The algorithm can be seen as one way of sparse coding, because α is very sparse. But the implementation is much simpler and the computation is much faster than traditional sparse coding, because there is no need to solve the L1-norm regularized optimization problem. On the other hand, we empirically find that the performance of this simple LCC coding is often comparable or better than traditional sparse coding for image classification. In addition, the algorithm can also be explained as a simple extension of vector quantization (VQ) coding, which can be recovered by setting $\kappa = 1$.

3.2. Super-vector Coding (SVC)

SVC is another way to extend VQ, which explores the geometry of data distribution. Suppose a codebook $B = [b_1, \ldots, b_p] \in \mathbb{R}^{d\times p}$ is obtained by running K-means algorithm. For a descriptor z, the coding procedure is

1. Find z's nearest basis vector in B, whose index is $i = \arg\min_j \|z - b_j\|^2$;
2. Obtain the VQ coding $\gamma \in \mathbb{R}^p$, where its i-th element is one, and all others are zeros.
3. Obtain the SV coding result

$$\beta = \left[\Big(\gamma_1 s, \gamma_1(z - b_1)\Big) \ldots \Big(\gamma_p s, \gamma_p(z - b_p)\Big)\right], \quad (2)$$

where $\beta \in \mathbb{R}^{(d+1)p}$, and s is a predefined small constant. The SVC can be seen as expanding VQ with local tangent directions, and is thus a smoother coding scheme.

At the pooling step, a linear pooling method has been derived by smoothing the Bhattacharyya kernel. Let $Z = [z_i]_{i=1}^n$ be the set of local descriptors of an image, and $[\beta_i]_{i=1}^n$ be their SV codes. Assigning z_i into those p vector quantization bins, we partition Z into p groups, with sizes proportional to ω_k, $\sum_{k=1}^p \omega_k = 1$. Then the pooling result for this image is

$$x = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\sqrt{\omega_{\delta(i)}}}\beta_i,$$

where $\delta(i)$ indicates the index of the group z_i belongs to.

Sets	Coding scheme	Descriptor	Coding dimension	SPM	Feature dimension	Data set Size(GB)
1	Local coordinate coding	HOG+LBP	8,192	10	81,920	167*
2		HOG	16,384	10	163,840	187*
3		HOG+LBP	20,480	10	204,800	260*
4	Super-vector coding	HOG	32,768	8	262,144	1374
5		HOG+LBP	51,200	4	204,800	1073
6		HOG	65,536	4	262,144	1374

Table 1. Extracted feature sets from ImageNet images for SVM training. The datasets with $*$ were compressed to reduce data size.

3.3. Parallel feature extraction

Depending on coding settings, the computation time for feature extraction of one image ranges from 2 seconds to 15 seconds on a dual quad-core 2GHz Intel Xeon CPU machine with 16G memory (single thread is used in computation). To process 1.2 million images, it would take around 27 to 208 days. Furthermore, feature extraction yields terabytes of data. It is very difficult for a single computer to handle such huge computation and such huge data. To speedup the computation and accommodate the data, we choose Apache Hadoop [21] to distribute computation over 20 machines and store data on Hadoop distributed file system (HDFS). Hadoop is an open source implementation of MapReduce computation framework and a distributed file system [3]. Because there is no interdependence in feature extraction tasks, MapReduce computation framework is very suitable for feature extraction. The HDFS distributes images over all machines and performs computation on the images located at local disks, which is called colocation. Colocation can speedup the computation by reducing overall network I/O cost. The most important advantage of Hadoop is that, it provides a reliable infrastructure for large scale computation. For example, a task can be automatically restarted if it encounters some unexpected errors, such as network issues or memory shortage. In our Hadoop cluster, we only use 6 workers on each machine because of some limitation of the machines. Thus, we have 120 workers in total.

We totally extracted six sets of features, as shown in Table 1. With the help of Hadoop parallel computing, the feature sets took 6 hours to 2 days to compute, depending on coding settings.

4. ASGD for SVM training

After feature extraction, we ended up with terabytes of training data, as shown in Table. 1. In general, the features by LCC are sparse even after pooling and they are much smaller than the ones generated by supervector coding. The largest feature set is 1.37 terabytes and non-sparse. While one may concatenate those features to learn an overall SVM for classification, we train SVMs separately for each feature set and then combine SVM scores to yield final classification. However, even training SVM for the smallest feature set here (about 160 GB) would not be easy. Furthermore, because the ImageNet dataset has 1000 categories, we need to train 1000 binary classifiers – the decision of using one-against-all SVMs is because training a joint multi-class SVM is even harder and may not have significant performance advantage. To our best knowledge, training SVMs on such huge datasets with so many classes has never been reported before.

Although there exist many off-the-shelf SVM solvers, such as SVM^{light} [12], SVM^{perf} [13] or LibSVM/LIBLINEAR [8], they are not feasible for such huge training data. This is because most of them are batch methods, which require to go through all data to compute gradient in each iteration and often need many iterations (hundreds or even thousands of iterations) to reach a reasonable solution. Even worse, most off-the-shelf batch-type SVM solvers require to pre-load training data into memory, which is impossible given the size of the training data we have. Therefore, such solvers are unsuitable for our SVM training. Indeed, LIBLINEAR recently released an extended version that explicitly considered the memory issue [25]. We tested it with a simplified image feature set (HOG descriptor only with coding dimension of 4,096, which generated 80GB training data). However, even on such a small dataset (as compared to our largest one, 1.37TB), the LIBLINEAR solver was not able to provide useful results after 2 weeks of running on a dual quad-core 2GHz Intel Xeon CPU machine with 16G memory. The slowness of the LIBLINEAR solver is not only due to its inefficient inner-outer loop iterative structure but also because it needs to learn as many as 1000 binary classifiers. Therefore, we need a (much) better SVM solver, which should be memory efficient, converge fast and have some parallelization scheme to train 1000 binary classifiers in parallel. To meet these needs, we propose a parallel *averaging stochastic gradient descent* (ASGD) algorithm for training SVM classifiers.

4.1. Averaging stochastic gradient descent

Let's use binary classification as an example for describing the ASGD algorithm [18][23]. We have training data that consists of T feature-label pairs, denoted as $\{\mathbf{x}_t, y_t\}_{t=1}^{T}$, where $\mathbf{x}_t$ is a $d \times 1$ feature vector representing

an image and $y_t \in \{-1, +1\}$ is the label of the image. Then, the cost function for binary SVM classification can be written as

$$\begin{aligned} L &= \sum_{t=1}^{T} L(\mathbf{w}, b, \mathbf{x}_t, y_t) \\ &= \sum_{t=1}^{T} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \max\left[0, 1 - y_t(\mathbf{w}^T\mathbf{x}_t + b)\right], \end{aligned} \quad (3)$$

where $\mathbf{w}$ is $d \times 1$ SVM weight vector, λ (nonnegative scalar) is a regularization parameter, and b (scalar) is a bias term. Then, the gradient of $\mathbf{w}$ and $\mathbf{b}$ are

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b, \mathbf{x}_t, y_t) = \begin{cases} \lambda\mathbf{w} - y_t\mathbf{x}_t & \text{if } \Delta_t < 1 \\ \lambda\mathbf{w} & \text{if } \Delta_t \geq 1 \end{cases}$$

$$\nabla_{\mathbf{b}} L(\mathbf{w}, b, \mathbf{x}_t, y_t) = \begin{cases} -y_t & \text{if } \Delta_t < 1 \\ 0 & \text{if } \Delta_t \geq 1 \end{cases}, \quad (4)$$

where $\Delta_t = y_t(\mathbf{w}^T\mathbf{x}_t + b)$ is the margin of the data pair $\{\mathbf{x}_t, y_t\}$.

The ASGD algorithm is a modification of conventional stochastic gradient descent (SGD) algorithms [15, 27]. For conventional SGD, training data are fed to the algorithm one by one, and the update rule for $\mathbf{w}$ and b respectively are

$$\begin{aligned} \mathbf{w}_t &= (1 - \lambda\eta)\mathbf{w}_{t-1} + \eta y_t \mathbf{x}_t \\ b_t &= b_{t-1} + \eta y_t \end{aligned} \quad (5)$$

if margin Δ_t is less than 1; otherwise, $\mathbf{w}_t = (1 - \lambda\eta)\mathbf{w}_{t-1}$ and $b_t = b_{t-1}$. The parameter η is step size. The above SGD algorithm is easy to implement, but it often takes many iterations to reach a good solution.

The ASGD algorithm is to add an averaging scheme to the above SGD algorithm. The averaging scheme is

$$\begin{aligned} \bar{\mathbf{w}}_t &= (1 - \alpha_t)\bar{\mathbf{w}}_{t-1} + \alpha_t \mathbf{w}_t \\ \bar{b}_t &= (1 - \alpha_t)\bar{b}_{t-1} + \alpha_t b_t, \end{aligned} \quad (6)$$

where α_t (e.g. $\alpha_t = 1/t$) is averaging parameter. Note that the averaging scheme does not affect the SGD update rule in Eq. 5, and the averaged SVM weights, $\bar{\mathbf{w}}_T$ and $\bar{b}_T$, will be output as the result of SVM training, not $\mathbf{w}_T$ and b_T.

The ASGD algorithm is known to have potential to achieve the theoretically optimal performance of stochastic gradient descent algorithms. It was shown that, asymptotically the ASGD algorithm is able to achieve similar convergence rate as second-order stochastic gradient descent algorithm [18], which is often much faster than its first-order counterpart. However, unlike the second-order SGD that needs to compute the inverse of Hessian matrix, the averaging is extremely simple to compute.

Despite the fact that the ASGD method has the potential to converge fast and is simple to implement, it has not been popular. We believe there are two main reasons. First, the ASGD algorithm achieves asymptotic convergence property (to gain similar performance as the second-order stochastic gradient descent) only when the number of data samples is sufficiently large. In fact, with insufficient data samples, ASGD can be inferior to regular SGD. This probably explains – it may not be able to observe the superiority of the ASGD method when dealing with medium-scale data. Second, for the ASGD algorithm to achieve fast convergence, the step size η needs to be carefully scheduled. We adopt the following step size scheduling [23]:

$$\eta = \eta_0 \frac{1}{(1 + \gamma\eta_0 t)^c}, \quad (7)$$

where η_0 (e.g. $\eta_0 = 10^{-2}$), γ and c are some positive constants, and they are problem-dependent. Typical values for c are 1 or 2/3. Recent analysis [23] shows that it is a good strategy to set γ to be the smallest eigenvalue of the Hessian matrix of a stochastic objective function. Therefore, for solving the SVM problem in Eq. 3, we set $\gamma = \lambda$ for the step size in Eq. 7.

There is an important implementation trick to significantly reduce the computation cost at each iteration of ASGD [23]. A plain implementation of ASGD would need *five* scalar-vector multiplications or dot products at each iteration: one for computing margin, two for updating $\mathbf{w}_t$ (Eq. 5) and two for averaging (Eq. 6). We choose to perform the following variable transform:

$$\begin{aligned} \mathbf{w}_t &= P^t_{1,1}\mathbf{v}_t \\ \bar{\mathbf{w}}_t &= P^t_{2,1}\mathbf{v}_t + P^t_{2,2}\mathbf{u}_t, \end{aligned} \quad (8)$$

where $\mathbf{P}_t = \begin{bmatrix} P^t_{1,1} & 0 \\ P^t_{2,1} & P^t_{2,2} \end{bmatrix}$ is a 2×2 projection matrix, and $\mathbf{v}_t$ and $\mathbf{u}_t$ are updated in the following manner:

$$\begin{aligned} \mathbf{v}_t &= \mathbf{v}_{t-1} + \eta y_t R_{1,1}\mathbf{x}_t; \\ \mathbf{u}_t &= \mathbf{u}_{t-1} + \eta y_t (R_{2,1} + \alpha_t R_{2,2})\mathbf{x}_t, \end{aligned} \quad (9)$$

where $\mathbf{R}_t = \begin{bmatrix} R^t_{1,1} & 0 \\ R^t_{2,1} & R^t_{2,2} \end{bmatrix} = \mathbf{P}_t^{-1}$, and $\mathbf{P}_t = \mathbf{T}_t\mathbf{P}_{t-1}$ with $\mathbf{T}_t = \begin{bmatrix} 1 - \lambda\eta & 0 \\ \alpha_t(1 - \lambda\eta) & 1 - \alpha_t \end{bmatrix}$ with $\mathbf{P}_1$ being an identity matrix, $\mathbf{w}_1 = \mathbf{v}_1$ and $\bar{\mathbf{w}}_1 = \mathbf{u}_1$. It is easy to check that the update in Eq. 8 is equivalent to the update in Eq. 5 and Eq. 6 but with only *three* scalar-vector multiplications or dot products: one for computing margin, and two for the computation in Eq. 9 – the transform in Eq. 8 is not computed until the last iteration when to output result.

4.2. Parallel training

Another important issue is how to parallelize the computation for training 1000 binary SVM classifiers [2].

Apparently, the training of the binary classifiers can be done independently. However, in contrast to the case where a single machine is used to train all classifiers and the major bottleneck is on computation, using a large number of machines for training will suffer from file I/O bottleneck since one of our training datasets is as large as 1.37 terabyte. If the file I/O bandwith is 20MB/second, simply loading/copying the dataset would take 19 hours. Therefore, we hope to load data as less times as possible while not incurring the computation bottleneck.

Our strategy here is to do memory sharing on each multicore machine. Each machine launches several programs to train different subset of the 1000 binary classifiers, and the programs are synchronized to train on the same chunk of training data. The training data is shared by all the programs on a machine through careful memory sharing. Therefore, the multiple programs only need to load data once (for each epoch). Such memory sharing scheme significantly reduces file loading traffic and speeds up SVM training dramatically.

5. Results

5.1. The performance of ASGD method for SVM training

As aforementioned, the major challenge of the large-scale ImageNet classification is on training SVMs with terabytes of training data and as many as 1000 categories. This paper proposes a parallel ASGD method that is aimed to have fast convergence and parallel computation. Fig. 3 shows the convergence comparison between the ASGD method and the regular SGD method. Both methods were performed on the dataset 5 in Table 1 – it is about 1 terabyte in total. We see that the ASGD method converged very fast. It reached fairly good solution after 5 iterations. In contrast, SGD (without averaging) converges much more slowly. It would take tens of iterations to reach a similarly good solution. For this specific dataset, each epoch took about 20 hours on three 8-core machines (only 6 programs running in parallel on each machine due to some limitations). Therefore, ASGD took about 4 days to finish SVM training while the regular SGD would have taken weeks if not months.

5.2. ImageNet classification results

With the proposed ASGD method and 12 eight-core machines, we were able to train 1000-class SVM classifiers for all those 6 feature sets listed in Table 1 within *one week*. Classification on each feature set outputs a set of SVM sores, and we combined them linearly to yield final prediction.

As a result, our classification system achieved 52.9% in classification accuracy and 71.8% in top 5 hit rate.

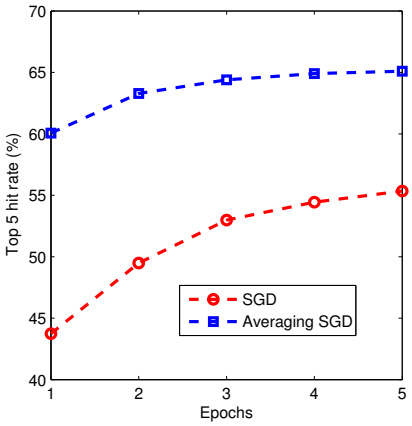

Figure 3. The convergence comparison between ASGD and regular SGD.

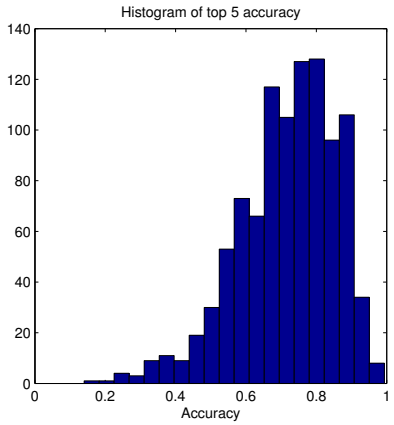

Figure 4. The histogram of the top 5 hit rate of the 1000 classes in ImageNet dataset.

Indeed, we see a huge improvement in performance from the baseline that was reported recently [6], which achieved about 20% in classification rate. Fig. 4 shows the histogram of the top 5 hit rate on 1000 classes. We see that the top 5 hit rate is mostly concentrated in the range of $60 \sim 90\%$ while it is over 90% for some classes but below 30% for some other classes. The easy classes include odometer, monarch butterfly, cliff dwelling, lunar crater, bonsai, trolleybus, geyser, snowplow, etc; the difficult classes include China tree, logwood tree, shingle oak, red beech, cap opener, Kentucky coffee tree, teak, alder tree, iron tree, grass pink, etc. The detailed top 5 hit rate for each of the 1000 classes is illustrated in Fig. 5.

6. Discussion

We have shown how to train an image classification system on the large-scale ImageNet dataset (1.2 million images) with many classes (1000 classes). We achieved state-

of-the-art performance on the ImageNet dataset: 52.9% in classification accuracy and 71.9% in top 5 hit rate. The key factors in our system are fast feature extraction and SVM training. We developed a parallel averaging stochastic gradient descent (ASGD) algorithm for SVM training, which is able to handle terabytes of data and 1000 classes.

In this paper, we observed very fast convergence from the ASGD algorithm. But we are still not able to quantitatively connect the superior empirical performance with existing theoretical analysis, most of which focuses on analyzing the asymptotic convergence property of ASGD. We will study how many training data samples would be needed for ASGD to enter its asymptotic convergence regime. Work in [23] has done some very interesting analysis in this regard. Meanwhile, we plan to systematically compare the ASGD method with other SGD methods (such as Pegasos [20]) for large-scale image classification.

References

[1] http://yann.lecun.com/exdb/mnist/.

[2] E. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui. Psvm: Parallelizing support vector machines on distributed computers. *Advances in Neural Information Processing Systems*, 20:16, 2007.

[3] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, page 281. The MIT Press, 2007.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[5] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[6] J. Deng, A. Berg, K. Li, and L. Fei-Fei. What does classifying more than 10,000 image categories tell us? ECCV, 2010.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.

[8] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9, June 2008.

[9] L. Fei-Fei, R. Fergus, and P. Peron. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *IEEE. CVPR 2004, Workshop on Generative-Model Based Vision*, 2004.

[10] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, Wisconsin, WI, June16 - 22 2003.

[11] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[12] T. Joachims. Making large-scale svm learning practical. LS8-Report, 24, Universitt Dortmund, LS VIII-Report, 1998.

[13] T. Joachims. Training linear svms in linear time. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*, 2006.

[14] S. Lazebnik, C. Achmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 2169–2178, New York City, June17 - 22 2006.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, 1998.

[16] D. G. Lowe. Distinctive image features from scale invariant keypoints. *Int'l Journal of Computer Vision*, 60(2):91–110, 2004.

[17] T. Ojala, M. Pietikainen, and D. Harwood. A comparative study of texture measures with classification based on feature distributions. *Pattern Recognition*, 29:51–59, 1996.

[18] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim*, 30, July 1992.

[19] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1-3):157–173, 2008.

[20] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, 2007.

[21] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc, 2010.

[22] S. Y. X. Wang, T. X. Han. An hog-lbp human detector with partial occlusion handling. In *ICCV*, 2009.

[23] W. Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. Technical Report 2009-L102, NEC Labs America.

[24] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[25] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, 2010.

[26] K. Yu, T. Zhang, and Y. Gong. Nonlinear learning using local coordinate coding. In *NIPS' 09*, 2009.

[27] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, 2004.

[28] X. Zhou, K. Yu, T. Zhang, and T. Huang. Image classification using super-vector coding of local image descriptors. In *ECCV*, 2010.

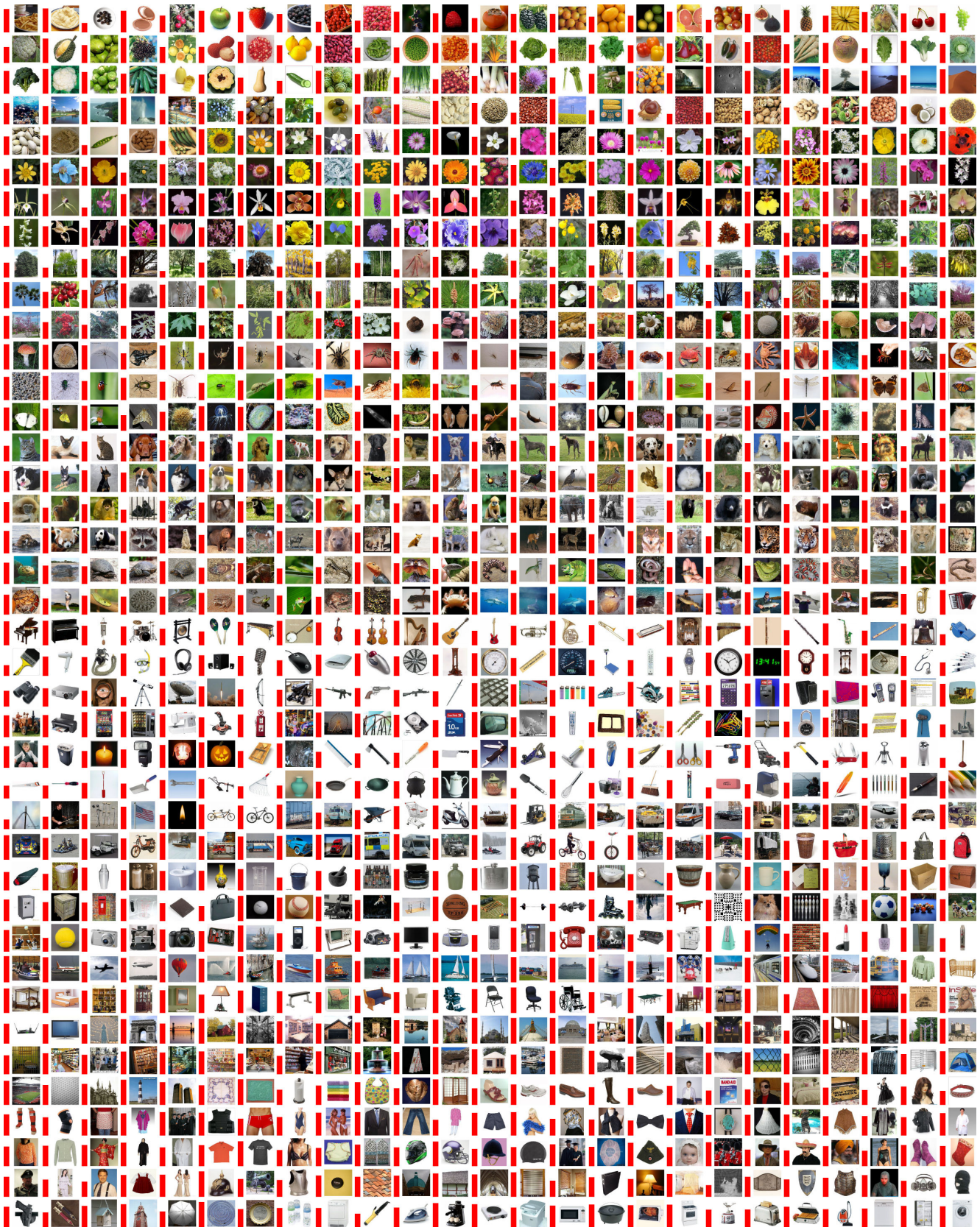

Figure 5. The top 5 hit rates on the 1000 categories in the ImageNet Challenge. The hit rate of each category is indicated by a red bar left to the image representing the category.