Large-Scale Slow Feature Analysis Using Spark for Visual Object Recognition

Da Li[1,2], Zhang Zhang[1,2(✉)], and Tieniu Tan[1]

[1] CRIPAC, Institute of Automation, Chinese Academy of Sciences, Beijing, China
da.li@cripac.ia.ac.cn, {zzhang,tnt}@nlpr.ia.ac.cn
[2] University of Chinese Academy of Sciences (UCAS), Beijing, China

Abstract. Data-driven feature learning has achieved great success in various visual recognition tasks. However, to handle millions of training image/video data efficiently, a high performance parallel computing platform combining with powerful machine learning algorithms plays a fundamental role in large-scale feature learning. In this paper, we present a novel large-scale feature learning architecture based on Slow Feature Analysis (SFA) and Apache Spark, where the slowness learning principle is implemented to learn invariant visual features from millions of local image patches. To validate the effectiveness of the proposed architecture, extensive experiments on pedestrian recognition have been performed on the INRIA pedestrian dataset. Experimental results show that the performance on pedestrian recognition can be promoted significantly with the growth of training patches, which demonstrates the necessity of large scale feature learning clearly. Furthermore, in comparisons with classical Histogram of Oriented Gradients (HOG) and Convolutional Neural Network (CNN) features, the slow features learnt by large-scale training patches can also achieve comparable performance.

Keywords: Large-scale feature learning · Slow feature analysis · Spark Parallel computing · Pedestrian recognition

1 Introduction

Extracting robust visual feature is an essential step towards achieving high performance in real-world visual recognition tasks. Recently, large-scale feature learning has achieved great success in various visual recognition tasks, in which powerful machine learning algorithms are integrated closely with high performance parallel computing platforms, clusters or GPU, so as to support data intensive computing from millions of training images. In this paper, we propose a novel large-scale feature learning architecture based on Slow Feature Analysis (SFA) [15] and Apache Spark [2], where the slowness learning principle is implemented to learn invariant visual features from millions of local image patches.

SFA is an unsupervised feature learning method, which intends to learn the invariant and slowly varying features from input signals. It has been successfully

J. Yang et al. (Eds.): CCCV 2017, Part III, CCIS 773, pp. 132–142, 2017.
https://doi.org/10.1007/978-981-10-7305-2_12

used for action recognition [12,21], trajectory clustering [20], handwriting digital recognition [7] and dynamic scenes classification [14]. Recently, Escalante-B and Wiskott [16] extend the SFA to supervised dimensionality reduction and the notion of slowness is generalized from temporal sequences to training graphs (a.k.a GSFA). Different with previous work, e.g., Sparse coding [9] and Autoencoder [6], most of which learn features to minimize the reconstruction error, we utilize the GSFA to explore the neighbor relationships in large-scale local patches from a view of manifold learning, where the learnt features will encode the intrinsic local geometric structures existing in a large number of training local patches.

Feature learning by the GSFA is to construct a feature space, in which the nearby points in original space will have similar locations. Thinking about a gray image with 16×16 pixels, and the value of each pixel ranges from 0 to 255. It means that there may exits 256^{256} possibilities [19] in the input space. Thus, the feature space learnt with limited data may have two issues: (1) Losing mush useful information makes visual samples belong to different patterns locate nearly in the feature space, i.e., inaccuracy. (2) Incomplete distribution of the training samples makes the learnt features are not effective in practical application, i.e., overfitting. On the other hand, it is the most important step of GSFA to construct the local neighbor relation graph in input space. Such graph is usually constructed through finding the top k nearest neighbors (k-NN). The hypothesis, near patches have similar patterns, is established only with large-scale data. While the inaccurate k-NN results will result in inaccurate slow feature functions.

As mentioned above, it is necessary to learn robust features with large-scale data. Meanwhile, relation graph construction in the input space is the key step of GSFA. However, it is a $O\left(n^2\right)$ problem [13] to construct it with k-NN searching. So it is a computational challenge with such large-scale data (about 10 million in this paper). The available of high performance hardware and parallel computing platform make it possible to alleviate this problem. Now, Hadoop [1] and Spark [2] attract much attention in big data tasks. The advantages of Spark, i.e., in-memory computation, sufficient APIs and good compatibility, make it faster and more convenient than MapReduce in Hadoop. Recently, some projects on deep learning with Spark are released, such as CaffeOnSpark [17] and SparkNet [10].

Motivated by above reasons, in this paper we construct a parallel architecture for large-scale slow feature learning using Spark. In summary, the efforts of this paper includes:

(1) We present a new large-scale feature learning architecture based on SFA and Spark, which is used to train invariant features with more than 10 million of local patches. The task can be completed efficiently within 140 h.
(2) We study the effect of the quantity of training patches for the visual recognition task on pedestrian recognition based on this architecture. The experimental results demonstrate that the performance is indeed improved significantly with the growth of training patches consistently.

(3) The learnt slow features can achieve superior performance than that of classical hand-crafted feature, i.e., Histogram of Oriented Gradients (HOG) [3] feature, which validate the effectiveness of large-scale slow feature learning.

2 Method

In this section, we first give a brief introduction on SFA, mainly about its essential problem and mathematical representation. Then, we explain the details about the parallel computing architecture for large-scale SFA with Spark. Finally, we will discuss how to use the learnt slow feature functions for an instance of visual recognition tasks, i.e., pedestrian recognition.

2.1 Slow Feature Analysis

Thinking about an object move through one's visual field, the signals fall on his retina change rapidly. However, the relevant abstract information, e.g. identity, changes slowly in a timescale [16]. It is known as slowness principle which first formulated by Hinton in 1989 [5]. Based on the slowness principle, SFA is to learn a group of slow feature functions to transform the input signal to a new feature space, in which the transformed signal varies as slow as possible. It is first proposed by Wiskott [15]. GSFA [16] is the extension of SFA, which generalizes the slowness principle from sequences to a training graph. Because the lack of temporal information in the training local patches, we adopt the GSFA to learn slow features in this paper. Mathematically, GSFA is formulated as follows:

Given a training graph $G = (V, E)$ with a set of nodes $V = \{\boldsymbol{x}(1), .., \boldsymbol{x}(N)\}$ and a set of edges $E := (\boldsymbol{x}(n), \boldsymbol{x}(n'))$, where $\langle n, n' \rangle$ denotes a pair of samples with $1 \leq n, n' \leq N$. GSFA aims at learning a group of input-output functions $\{g_j\}, 1 \leq j \leq J$ such that the J-dimensional output signals $y_j(n) = g_j(\boldsymbol{x}(n))$ have the minimum difference with their nearest neighbors,

$$minimize\ \Delta_j = \frac{1}{R} \sum_{n,n'} \gamma_{n,n'} \left(y_j\left(n'\right) - y_j\left(n\right) \right)^2 \tag{1}$$

s.t.

$$\frac{1}{Q} \sum_{n} \nu_n y_j(n) = 0 \quad weighted\ zero\ mean\ , \tag{2}$$

$$\frac{1}{Q} \sum_{n} \nu_n \left(y_j\left(n\right) \right)^2 = 1 \quad weighted\ unit\ variance\ , \tag{3}$$

$$\frac{1}{Q} \sum_{n} \nu_n y_j(n) y_{j'}(n) = 0 \quad weighted\ decorrelation\ , \tag{4}$$

with

$$R = \sum_{n,n'} \gamma_{n,n'}\ and\ Q = \sum_{n} \nu_n\ , \tag{5}$$

where ν and γ denote the weight of node and the weight of edge respectively. And $\gamma_{n,n'} = \gamma_{n',n}$ means that the edges are undirected. Constraint (2) is used to make constraint (3) and (4) with concise form. Constraint (3) avoids the trivial solution. And constraint (4) ensures that different functions g_j take different properties of the input signal. Meanwhile constraint (4) enforces the functions g_j are ordered based on their slowness, in which the first is the slowest one.

For the linear GSFA, $g_j(\boldsymbol{x}) = \boldsymbol{w}_j^T \boldsymbol{x}$. This optimization problem equals to solve a generalized eigenvalue problem,

$$AW = BW\Lambda, \tag{6}$$

with

$$A = \frac{1}{R} \sum_{n,n'} \gamma_{n,n'} (\boldsymbol{x}(n') - \boldsymbol{x}(n))(\boldsymbol{x}(n') - \boldsymbol{x}(n))^T \tag{7}$$

and

$$B = \frac{1}{Q} \sum_{n} \nu_n (\boldsymbol{x}(n) - \widehat{\boldsymbol{x}})(\boldsymbol{x}(n) - \widehat{\boldsymbol{x}})^T \tag{8}$$

where $\widehat{\boldsymbol{x}} = (1/Q) \sum_n \nu_n \boldsymbol{x}(n)$ is the weighted mean of all the samples. For the nonlinear GSFA, the inputs can be firstly mapped to a nonlinear expansion space and then solve the problem with the steps in the linear case. The nonlinear expansion function is defined by

$$\boldsymbol{h}(\boldsymbol{x}) := [h_1(\boldsymbol{x}), ..., h_M(\boldsymbol{x})] \ . \tag{9}$$

So the slow feature functions of GSFA can be calculated with three steps:

(1) Construct the undirected graph $G = (V, E)$; and confirm the weights of nodes and edges. In this paper, $\nu_n = 1$ and $\gamma_{n,n'} = 1/N_{edges}$.
(2) Map the inputs to a nonlinear space with a nonlinear function and centralize them. In this paper, we use the quadratic expansion. For a I-dimensional input signal, $\boldsymbol{h}(\boldsymbol{x}) = [x_1, ..., x_I, x_1x_1, x_1x_2, ..., x_1x_I]$.
(3) Solve the generalized eigenvalue problem $AW = BW\Lambda$. The eigenvectors corresponding to the first J smallest eigenvalues are selected as slow feature functions.

2.2 Large-Scale SFA in Spark

The proposed parallel computing architecture for large-scale SFA runs on a cluster consists of 10 machines with 96 CPU cores, as shown in Fig. 1, in which one for master node and the other nines for worker nodes. And the total memory is 510 GB. Since the advantages of Spark in big data processing, we choose it as our computational platform to implement SFA with large-scale data. For a Spark platform, the core concept is Resilient Distributed Datasets (RDD) [18], which is a distributed memory abstraction for in-memory computations in cluster. Furthermore, the application running on Spark consists of a driver program and many executors, in which driver program runs the main function and executors

run different tasks. The driver program launches different tasks to workers with a variance of RDD transformations and actions (shown in the brace of Fig. 1). The launched tasks will be implemented by the executors in each worker, and the results will return to driver program or produce new RDDs. Our large-scale SFA parallel architecture based on above principle as well.

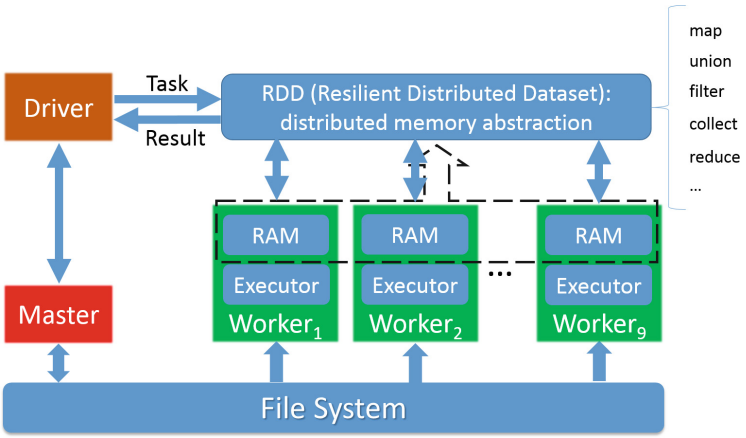

Fig. 1. An overview of the computing platform with Spark.

An overview of the architecture for large-scale SFA is shown in Fig. 2. We may summarize the large-scale SFA with Spark as five tasks:

(1) Data loading and patches extraction. Load the images data from hard disks to Spark RDD. Then extract patches from each image by dense sampling in parallel. The extracted patches will be stored as new Spark RDDs.
(2) Pre-processing. Implement PCA whitening on extracted patches to remove the redundant information and make the patches with zero mean and unit variance. It will also bring convenience for the following operations. Its result (a transformation matrix) will act on the input of nonlinear expansion.
(3) Graph construction. It is the core step in GSFA. We construct the graph with k-NN searching in parallel, in which the broadcast variables are used to improve the efficiency and the patches are divided into multiple RDDs (see Fig. 2) to satisfy the limitations of memory resources. FLANN[1] is used for k-NN searching locally (more details in Algorithm 1). The k-NN results and their source patches are used to compute the differences matrix after nonlinear expansion.
(4) Nonlinear expansion (Eq. (9)). Implement a quadratic expansion to the pre-processed patches data in parallel.

[1] http://www.cs.ubc.ca/research/flann/.

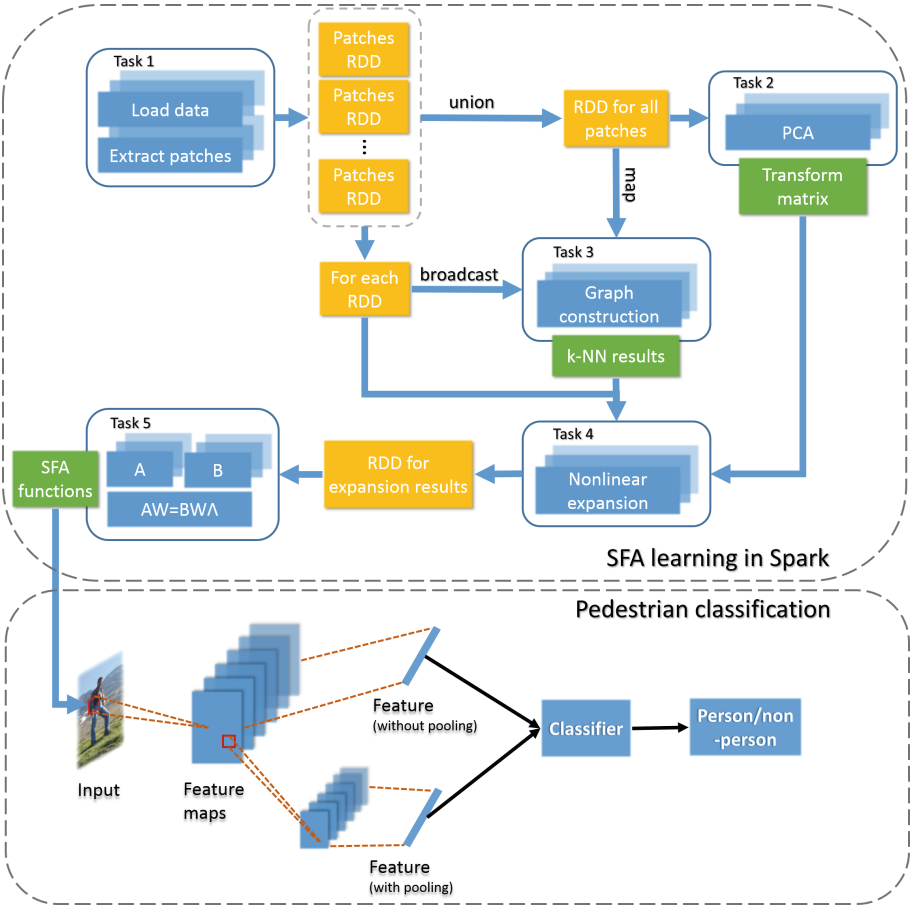

Fig. 2. Diagram of large-scale SFA in Spark for pedestrian classification. The upper part is the architecture for large-scale SFA in Spark. The lower part shows how to extract feature using slow feature functions in pedestrian classification. Two methods (with max pooling or not) are tested in experiment. (Color figure online)

(5) Solve the generalized eigenvalue problem. Firstly, compute the covariance matrices (Eqs. (7) and (8)) in parallel. Then collect the matrices from Spark RDD to the memory of master node to calculate the eigenvalues and eigenvectors.

2.3 Pedestrian Recognition Using Slow Feature Functions

Pedestrian recognition is one of the essential tasks in visual object recognition. Instead of detecting pedestrian over whole images where the performance depends not only on the feature itself, but also on the post-processing step, e.g.,

Algorithm 1. Graph construction: k-NN searching in Spark

Input: *sc*: the spark context which defines the entry of a spark application; *rdd_total*: a RDD contains all the extracted training patches; *rdd_query*: a RDD contains the query patches (its size is determined by the size of memory); *k*: the number of nearest neighbors to search.

Output: *rdd_knn*: a RDD contains the top k nearest neighbors of each query patch.

```
query_data_broadcast := sc.broadcast(rdd_query.collect());
rdd_knn_local := rdd_total.map(labmda element : {
    flann := FLANN();
    if isComingFromDifferentImages(element, query_data_broadcast)
        knn_local := flann.nn(element, query_data_broadcast, k, kdtree);
    end if
    return knn_local;
});
rdd_knn_local.cache();
knn_global := rdd_knn_local.treeAggregate();
rdd_knn_local.unpersist();
rdd_knn := sc.parallelize(knn_global);
return rdd_knn
```

Non-Maximum Suppression (NMS) [11], we only perform binary classifications over a set of candidate windows extracted from whole images, for a straight evaluation of the effectiveness of the slow features.

As shown in the lower part of Fig. 2, the learnt function can be seen as a filter (red block). For a candidate window (input), it is filtered using all the learnt functions with a specific stride that will generate a number of feature maps where each feature map corresponds to one slow feature function. Then, a nonlinear normalization function, e.g., the Sigmoid function used in this paper, is performed to clip the responses with large values. After that, we test two methods to generate the final feature vector: (1) concatenate the normalized feature maps directly; (2) execute max-pooling operation over a local region on the feature maps before the concatenation. To predict whether there is a person in the input window or not, a linear Support Vector Machine (SVM) is trained as the classifier.

3 Experimental Results

In this paper, we select the INRIA Person dataset [3] as the training and test dataset. It totally includes 2416 positive training windows and 1126 positive test windows, where each window corresponding to one person annotated in the dataset. To generate negative windows, we randomly extract ten windows per training negative image (12180 windows) and five windows per test negative image (2265 windows). The size of each window is 96×160 pixels.

We perform the slow feature learning with two strategies, i.e., *unsupervised SFA* and *supervised SFA*. For the supervised SFA, three kinds of methods are

implemented as shown in the lower part of Table 1. The key parameters in training SFA are chosen as follows: patch size is 16×16 pixels, k in k-NN searching is 3 and dense sampling stride is 4. When generate final features, we set filtering stride to 8 and window size to 64×128 pixels. For different numbers of selected functions (as shown in Table 1), the dimensions of final features are 2100, 2100, 13000 and 3150, respectively. Figure 3 shows the results of the visualization of the first 25 slow feature functions learnt by supervised and unsupervised learning strategies.

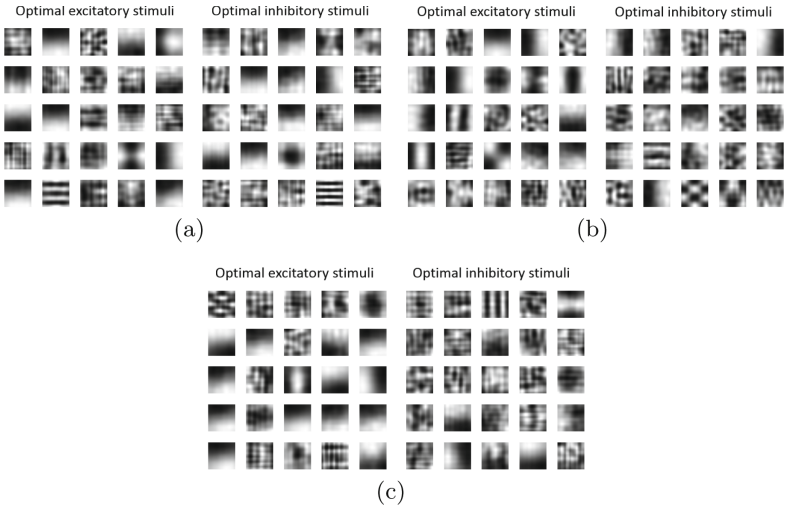

Fig. 3. Visualization of learnt slow feature functions.(a) The slow feature functions learnt by negative samples. (b) The slow feature functions learnt by positive samples. (c) The slow feature functions learnt by unsupervised SFA.

We study the relationship between the populations of training patches and the classification performance, under the case of unsupervised SFA. From Fig. 4, it clearly shows that the classification performance can be improved (precision is increased and miss rate is decreased) with the growth of training patches, which indicates that the large-scale training samples enhance the ability of the learnt slow features for pedestrian recognition. Meanwhile, compare the elapsed time of SFA learning between single-core machine and our proposed SFA+SPARK architecture, the speed-up ratio ranges from 3.56 to 14.67 when the number of training patches increases from 6k to 120k. It indicates that the efficiency has been improved significantly, especially when the data size is very large. And it can handle more than 10 million local patches within 140 h efficiently.

We compare both unsupervised and supervised SFA with the HOG feature and CNN feature. In experiment, we use the scikit-image[2] library to extract

[2] https://github.com/scikit-image/scikit-image.

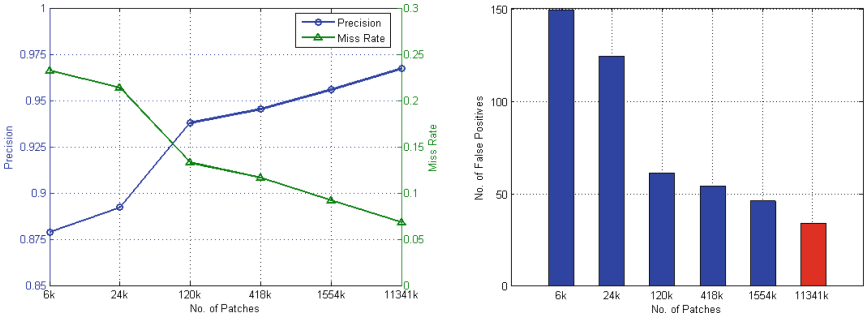

Fig. 4. Effect of the number of training samples on classification performance. The variations of precision and miss rate is shown on the left side, while the variation of the number of false positives located on the right.

Table 1. Comparison on classification performance with other features.

Methods	#Patches	#Slow feature functions	Precision	Miss rate	#False Positives (FP)
HOG [3]	-	-	0.9773	0.0521	18
CNN [8]	-	-	**0.9862**	**0.0415**	**0**
Unsupervised SFA	11 millions	20	0.9673	0.0680	34
Supervised SFA (positives w/o max pooling)	1.8 millions	20	0.9650	0.0777	31
Supervised SFA (positives with max pooling)	1.8 millions	200	0.9703	0.0583	35
Supervised SFA (positives & negtives)	11 millions	30	**0.9800**	**0.0477**	**14**

HOG feature, in which the cells per block is 2×2, pixels per cell is 8×8, number of orientations is 9 and the stride is 8. So the dimension of HOG feature is 3780. While the CNN feature is trained on the INRIA dataset using Caffe[3]. We select the output of fc7 as the final features which dimension is 4096.

The classification results of different methods are list in Table 1. The performance of our method trained with both the positive and negative samples under supervised learning strategy is superior to the performance of the HOG feature. However, it is worse than the CNN feature. It may because the CNN extracts features with multiple layers that can capture much high-level information. The hierarchical SFA [4] learning architecture may be introduced into our future work to improve the performance. From the table, we can also conclude that, the performance of slow functions trained with both the positive and negative samples is superior to that of the slow features trained only using the positive samples. Furthermore, along with the increasing of the number of slow feature functions, the max pooling may achieve more superior performance, compared to the feature representation obtained by directly concatenation. In summary, compared with the HOG feature and CNN feature, the comparable performance validates the effectiveness of our large-scale SFA+SPARK learning architecture.

[3] https://github.com/BVLC/caffe.

4 Conclusion

In this paper, we present a large-scale feature learning architecture based on SFA and Apache Spark. It is used to learn invariant visual features from more than 10 million of local patches under supervised and unsupervised learning strategies. Then, we extract features using the learnt slow feature functions to do pedestrian classification on the INRIA person dataset. The experiment results indicate that the efficiency of the SFA learning using large-scale data can be improved greatly with our proposed parallel computing architecture. It also demonstrates that the classification performance can be improved along with the increasing of the number of training patches. Furthermore, the performance of learned slow feature is comparable to the classic HOG feature and CNN feature in pedestrian classification.

Acknowledgments. This work is jointly supported by the National Key Research and Development Program of China (2016YFB1001005), the National Natural Science Foundation of China (Grant No. 61473290), the National High Technology Research and Development Program of China (863 Program) under Grant 2015AA042307.

References

1. Apache: Apache hadoop. https://hadoop.apache.org/
2. Apache: Apache spark. https://spark.apache.org/
3. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: CVPR 2005, vol. 1, pp. 886–893. IEEE (2005)
4. Escalante-B, A.N., Wiskott, L.: Heuristic evaluation of expansions for non-linear hierarchical slow feature analysis. In: Machine Learning and Applications and Workshops (ICMLA), vol. 1, pp. 133–138. IEEE (2011)
5. Hinton, G.E.: Connectionist learning procedures. Artif. Intell. **40**(1), 185–234 (1989)
6. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
7. Huang, Y., Zhao, J., Tian, M., Zou, Q., Luo, S.: Slow feature discriminant analysis and its application on handwritten digit recognition. In: IJCNN 2009, pp. 1294–1297. IEEE (2009)
8. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: convolutional architecture for fast feature embedding. In: Proceedings of the ACM International Conference on Multimedia, pp. 675–678. ACM (2014)
9. Lee, H., Battle, A., Raina, R., Ng, A.Y.: Efficient sparse coding algorithms. In: Advances in Neural Information Processing Systems, pp. 801–808 (2006)
10. Moritz, P., Nishihara, R., Stoica, I., Jordan, M.I.: SparkNet: training deep networks in spark. arXiv preprint arXiv:1511.06051 (2015)
11. Sermanet, P., Kavukcuoglu, K., Chintala, S., LeCun, Y.: Pedestrian detection with unsupervised multi-stage feature learning. CVPR **2013**, 3626–3633 (2013)
12. Sun, L., Jia, K., Chan, T.H., Fang, Y., Wang, G., Yan, S.: DL-SFA: deeply-learned slow feature analysis for action recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2625–2632 (2014)

13. Talwalkar, A., Kumar, S., Rowley, H.: Large-scale manifold learning. In: CVPR 2008, pp. 1–8. IEEE (2008)
14. Theriault, C., Thome, N., Cord, M.: Dynamic scene classification: learning motion descriptors with slow features analysis. In: CVPR 2013, pp. 2603–2610. IEEE (2013)
15. Wiskott, L., Sejnowski, T.J.: Slow feature analysis: unsupervised learning of invariances. Neural Comput. **14**(4), 715–770 (2002)
16. Wiskott, L., et al.: How to solve classification and regression problems on high-dimensional data with a supervised extension of slow feature analysis. J. Mach. Learn. Res. **14**(1), 3683–3719 (2013)
17. Yahoo: Caffe on spark (2016). https://github.com/yahoo/CaffeOnSpark
18. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. USENIX Association (2012)
19. Zhang, L.: Big data for visual recognition: potentials and challenges. Technical report, Microsoft Research Asia (2012)
20. Zhang, Z., Huang, K., Tan, T., Yang, P., Li, J.: RED-SFA: relation discovery based slow feature analysis for trajectory clustering. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 752–760 (2016)
21. Zhang, Z., Tao, D.: Slow feature analysis for human action recognition. IEEE Trans. Pattern Anal. Mach. Intell. **34**(3), 436–450 (2012)