2016

Improving Efficiency in Deep Learning for Large Scale Visual Recognition

Baoyuan Liu
University of Central Florida

IMPROVING EFFICIENCY IN DEEP LEARNING FOR LARGE SCALE VISUAL RECOGNITION

by

BAOYUAN LIU
B.S. Shanghai Jiao Tong University, 2010
M.S. University of Central Florida, 2013

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2016

Major Professor: Hassan Foroosh

ABSTRACT

The emerging recent large scale visual recognition methods, and in particular the deep Convolutional Neural Networks (CNN), are promising to revolutionize many computer vision based artificial intelligent applications, such as autonomous driving and online image retrieval systems. One of the main challenges in large scale visual recognition is the complexity of the corresponding algorithms. This is further exacerbated by the fact that in most real-world scenarios they need to run in real time and on platforms that have limited computational resources. This dissertation focuses on improving the efficiency of such large scale visual recognition algorithms from several perspectives.

First, to reduce the complexity of large scale classification to sub-linear with the number of classes, a probabilistic label tree framework is proposed. A test sample is classified by traversing the label tree from the root node. Each node in the tree is associated with a probabilistic estimation of all the labels. The tree is learned recursively with iterative maximum likelihood optimization. Comparing to the hard label partition proposed previously, the probabilistic framework performs classification more accurately with similar efficiency.

Second, we explore the redundancy of parameters in Convolutional Neural Networks (CNN) and employ sparse decomposition to significantly reduce both the amount of parameters and computational complexity. Both inter-channel and inner-channel redundancy is exploit to achieve more than 90% sparsity with approximately 1% drop of classification accuracy. We also propose a CPU based efficient sparse matrix multiplication algorithm to reduce the actual running time of CNN models with sparse convolutional kernels.

Third, we propose a multi-stage framework based on CNN to achieve better efficiency than a single traditional CNN model. With a combination of cascade model and the label tree framework, the

proposed method divides the input images in both the image space and the label space, and processes each image with CNN models that are most suitable and efficient. The average complexity of the framework is significantly reduced, while the overall accuracy remains the same as in the single complex model.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Hassan Foroosh, for his guidance, encouragement and patience. He has been always supportive with me and given me numerous great advices on research. Discussing research with him has always been enlightening and also fun. I'm deeply honored to be his PhD student.

I would also like to thank Dr. Marshall Tappen, who has been my advisor for one year in UCF and my manager while I was an intern in Amazon. Joining his group was the turning point of my Phd career and the best decision I've made these years. I cannot imagine how he can treat me more generously . I've learned so much from him both on research and on life.

This thesis is not possible without the love and support from my wife, Min Wang. The years that we spent together at UCF has been the best years in my life. Both research and life become much more delightful with her companion. We have been through all the happiness and sorrow together, and every detail in this Phd with her would be the best things to remember for my whole life.

Finally, I'm extremely grateful to my parents. Whenever I felt depressed, the weekly chat with them on Friday night always brought me hope and cheered me up. Every time I went back to China and stayed with them for a month, I became totally revived. This journey would be much harder without their unconditional support and sacrifice.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Visual recognition analyzes the information from an image and recognizes the type of objects in it. Visual recognition with sufficient accuracy and reliability is essential for a successful artificial intelligence system. For smart AI systems, such as autonomous driving, home assistant robot, and online image retrieval systems, the ability to recognize the visual objects is preliminary of performing more advanced intelligent behaviors.

Researchers have been working on the problem of visual recognition for decades. Due to limited resource of images and computing power, the visual recognition systems in the early days were trained and evaluated on small scale datasets, which typically include a few hundreds of categories and each category contains less than one hundred images. In recent years, with the dramatically increase of internet images and computing resources, training large scale systems that attempt to match human's intelligence becomes practical and attracts more and more attention.

The problem of large scale visual recognition is challenging for computers. First, the visual world is an extremely complicated space that is determined by many underlying physical factors. For objects within the same category, viewpoint, distance, illumination condition, occlusion, and background clutters can all effectively modify each pixel of its image. Second, the number of categories are naturally large for human vision system. Humans are able to recognize approximately 30 thousand different categories. An AI system that is designed to function as well as human, such as an autonomous driving agent, is required to possess comparable recognition ability. Third, many categories are easy to confuse with each other, which poses another level of challenge.

The difficulty of large scale visual recognition leads to the complexity of corresponding algorithms. Due to the complicated image space and the heavy confusion between similar categories, large amount of complex operations are necessary to transfer the images to a more separable space

in which machine learning algorithm can be utilized to train an effective classifier; In addition, the large number of categories also increases the complexity of the classifier, since each category requires certain amount of parameters to represent its distribution in the transformed feature space.

On the other hand, in most real world scenarios, the visual recognition systems are required to run with near real-time processing speed, but at platforms that possess limited amount of computing resource. For instance, in a home assistant robot or an auto-piloting drone, the AI systems need to response in typically less than 100ms to make sure it can work smoothly and handle emergency quickly. In contrast, due to limited space and battery life, the systems are most likely running on embedded System on Chip platforms, which have very limited computing power comparing to the servers and GPUs that are currently used to train large scale image classification models.

Therefore, the efficiency of current large scale recognition algorithms becomes a prohibiting bottleneck that limits their wider adoptions in many potential applications. In this dissertation, I focus on improving the efficiency of large scale visual recognition algorithms.

A standard pipeline of traditional image classification algorithm involves two fundamental stages. In the first stage, information is extracted from the raw pixels of each image, and is represented by a 1D feature vector. This stage is traditionally called feature extraction. A desirable feature extraction algorithm describes the categorical characteristics from the images, while is invariant to in-class diversity. In the second stage, a multi-class statistical classifier is trained and evaluated based on the generated feature vectors.

The computational complexity of an image classification system is the sum of these two stages. The feature extraction is performed once for each image regardless of the number of categories, while the complexity of the classifier is highly dependent on the scale of the problem. Therefore, when the size of the dataset scales up, the complexity of the classifier becomes the bottleneck of the whole framework.

For large scale problems, since both the number of images and the number of categories increase dramatically, the classifier should be chosen so that the complexity of both training and inference is limited to acceptable level. Linear 1-vs-all SVM or multi-class logistic regression model achieves similar inference complexity, which is linear to the length of feature vector and number of classes. The parameters for the popular linear SVM and multinomial logistic regression models contain one vector per possible class. Thus, if there are K possible classes, assigning a label to a new feature vector $\boldsymbol{x}$ will require the computation of K dot products between $\boldsymbol{x}$ and the vectors defining the classifier. For a relatively small number of categories, this computational cost is not significant enough to require attention.

However, as both the number of categories and need for fast recognition increases, this linear relationship between the complexity of recognition and the number of classes can become problematic. This issue is compounded if the classification process involves computations that are more complex than a dot product.

In [6], Bengio et al. introduce the label tree model for reducing the complexity of recognition in a problem with a larger number of classes. In the label tree model, a feature vector is assigned a label by traversing a tree. At each node visited, the classifier computes the dot product between the feature vector and a small number of vectors. This tree structure causes the classification complexity to grow logarithmically, rather than linearly, with the number of classes.

In Chapter 3, we present a novel, probabilistic approach to learning the parameters of a label tree. We show how a recursive process learns the tree parameters. The probabilistic approach provides a natural way of soft label partitioning, and is able to pass the label distribution as a prior information down to the leaf nodes. As the results in Section 3.6 will show, this approach produces significantly improved accuracies over previous results in [16]. The probabilistic model also makes it possible to tune accuracy versus efficiency without having to retrain the tree.

From a broader perspective, formulating the label tree in a probabilistic framework provides a straight-forward avenue for integrating more complex, accurate classification models into the label tree framework. With this probabilistic formulation, any classifier that can be expressed probabilistically can be integrated into the label tree.

In 2012, Krizhevsky et. al. [50] first propose to use Convolutional Neural Networks (CNN) to solve the large scale image classification problem, and significantly improves the state-of-the-art on ImageNet LSVRC [14] dataset, which includes one thousand categories and 1.2 million training images. Since then, the research of visual recognition has come to the deep learning era. Numerous improvement have been made on CNN, with the depth of the network increasing from 7 to hundreds, and the top-5 error on ImageNet LSVRC dataset reduced from 16% to as low as 3.5%.

CNN is a type of feed-forward artificial neural network that is specifically designed to recognize image signal. In CNN, the connections between neurons are organized in a convolutional fashion, so that each neuron receives information from a small local receptive field rather than all the bottom neurons. As in standard neural network, a non-linear neuron function is performed on the output of each neuron, making the network able to learn highly complex distributions in feature space. Spatial pooling layers are also occasionally used to reduce the spatial dimension of feature and gather information. At the end of CNN, a multi-class classifier, most commonly logistic regression, maps the transformed feature space to the space of semantic categories. Figure. 1.1 shows an example CNN model. In general, the spatial dimension of the feature map is gradually reduced by pooling or convolution with stride, while the number of channels increases with the depth of the network.

Due to the complexity of image space, large amount of convolutional layers are required to transfer the images into separable space. However, convolutional layers are computationally expensive.

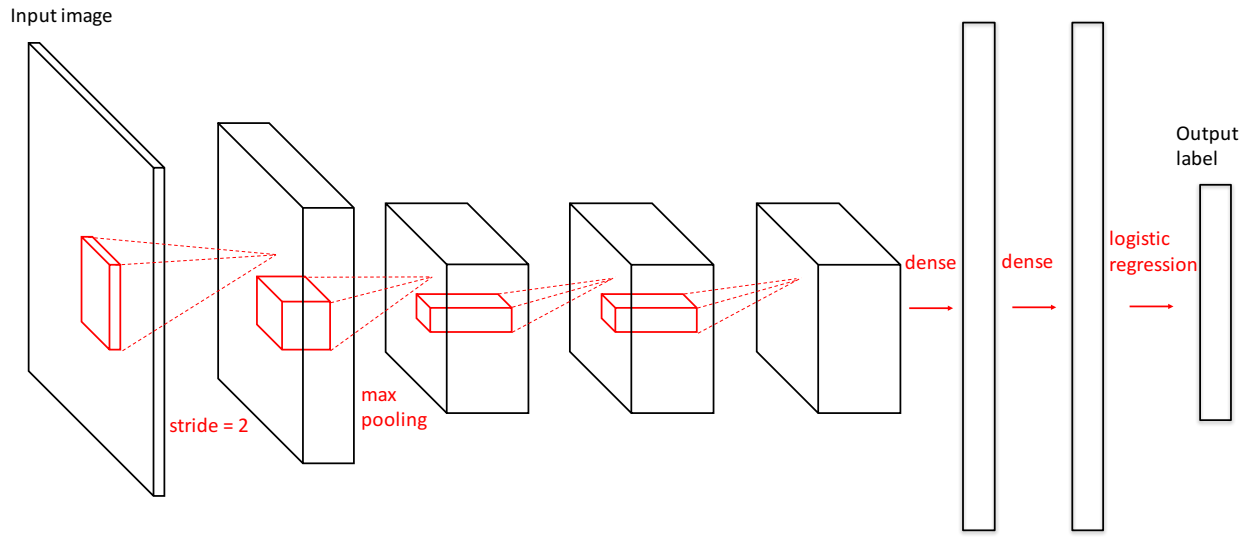


Figure 1.1: An example Convolutional Neural Network (CNN). The input of CNN is a color image and the its output is the probability of each label. The essential building block of CNN is the convolutional layer.

Each convolutional kernels involves convolving with all the input channels. and the number of input and output channels need to be large to retain the information in the image. Results of ImageNet LSVRC competitions in recent years have demonstrated a strong correlation between the network size and the classification accuracy. The ILSVRC 2014 submission from VGG [77] builds a network with up to 16 convolutional layers that reduces the top-5 classification error to 7.4%, at the expense of approximately one month of network training with 4 high-end GPUs.

The structure of CNNs makes it reasonable to conjecture that there exists heavy redundancy in these networks. Due to the highly non-convex property of neural networks, over-parameterization with random initialization is necessary to overcome the negative impact of local minimum in network training. Additionally, the fact that no independence constraint is imposed among the convolutional kernels for each layer in the training phase also indicates high potential for redundancy.

In Chapter 4, we show that this redundancy makes it possible to notably reduce the amount of computation required to process images, by sparse decomposition of the convolutional kernels. As

Figure 4.1 illustrate, two-stage decomposition's are applied to explore the inter-channel and intra-channel redundancy of convolution kernels. We first perform an initial decomposition based on the reconstruction error of kernel weights, then fine-tune the network while imposing the sparsity constraint. In the fine-tuning phase, we optimize the network training error, the sparsity of convolutional kernels, as well as the number of convolutional bases simultaneously, by minimizing a sparse group-lasso object function. Surprisingly high sparsity can be achieved in our model. We are able to zero out more than 90% of the convolutional kernel parameters of the network in [51] with relatively small number of bases while keeping the drop of accuracy to less than 1%.

In our Sparse Convolutional Neural Networks (SCNN) model, each sparse convolutional layer can be performed with a few convolution kernels followed by a sparse matrix multiplication. It could be assumed that the sparse matrix formulation naturally leads to highly efficient computation. However, computing sparse matrix multiplication can involve severe overhead that makes it difficult to actually achieve attractive acceleration. Thus, we also propose an efficient sparse matrix multiplication algorithm. Based on the fact that the sparse convolutional kernels are fixed after training, we avoid the necessity of indirect and discontinuous memory access by encoding the structure of the input sparse matrix into our program as the index of registers. Our CPU-based implementation demonstrates much higher efficiency than off-the-shelf sparse matrix libraries and a significant speedup over the original dense networks is realized. While convolutional network systems are dominated by GPU-based approaches, advances in CPU-based systems are useful because they can be deployed in commodity clusters that do not have specialized GPU nodes.

While the idea of label tree is able to significantly reduce the complexity of large scale multi-class classifier, it will make very little improvement of efficiency on the CNN framework. Different from the Bag of Visual Words (BoVW) model, most of the computation is consumed by the convolutional layers in CNN, which can be considered as a form of feature extractor, and need to be performed for images from all categories. The complexity of the convolutional layers is highly

dependent on the difficulty of the problems, namely, the number of categories and the level of confusion between them. When the number of categories is small and they are easy to distinguish, only a few number of convolutional layers are adequate separate them in feature space; While for more complicated dataset, a deeper stack of convolutional layers are necessary to achieve accuracy classification. Therefore, reducing the the original large scale problem to smaller scale, less complex problem will lead to a decrease of the complexity of convolutional layers, thus increasing the efficiency of the overall CNN framework.

In Chapter 5, we propose to decompose a single CNN model of high complexity into a multi-stage, tree structured framework of multiple efficient models. Instead of processing each image with a single computationally expensive network, coarse probabilistic estimates of the level of "difficulty" of the images are first estimated, and then assigned to models that are trained to recognize specific categories, with different levels of complexity. In this way, we trade the amount of memory space for less run time. The proposed framework is built on the idea of combining the cascade model [88] and the label tree algorithm [7].

CHAPTER 2: LITERATURE REVIEW

2.1 Image Classification

Bag of Visual Words (BoVW) framework has been the standard pipeline for image classification before CNN. For each input image, large amount of distinctive local patches are detected, and then represented with descriptors with certain types of invariance, such as SIFT [62] and LBP [93]. The descriptors for the whole training dataset are clustered into a "codebook", and each cluster center is called one "codeword". Each local patch is assigned to the nearest codeword. For each image, a histogram that describes the frequency of occurrence of each codeword is generated. Then a multi-class classifier is trained based on the histograms.

A few variants have been proposed to improve the performance of BoVW model. Spatial Pyramid Matching (SPM) model [55] provides a framework that takes advantage of the spatial information without sacrificing the robustness and invariance of BoVW.The input image is partitioned into increasing fine sub-regions, and a histogram of visual words are generated in each sub-region. The histograms are weighted and concatenated into one feature vector, which are used for training and predicting with multi-class classifier.

The histograms generated by standard BoVW framework lie in a non-linear high dimension space, and requires non-linear Mercer kernel SVM to achieve optimal performance. However, for non-linear SVMs, the training complexity is quadratic with number of training samples and the test complexity is linear with number of support vectors. This makes them impractical to be adopted for large scale image categorization problems, in which the number of training images can be as large as millions.

A few methods have been proposed to use linear SVM to achieve competitive or better result than

non-linear kernel SVMs. In [66] and [87], the histograms generated by BoVW are projected to linear space with specifically designed function, so that the kernel distances between the original histogram is approximately equal to the Euclidean distances between the projected feature vectors. Substituted the vector quantization process with sparse coding have been shown in [97] to achieve much better accuracy with linear SVM than the standard vector quantization with non-linear kernel SVM. In [92], Wang et al. propose Locality-constrained Linear Coding (LLC) scheme to further improve the accuracy of feature encoding. In LLC, locality constraint was imposed while encoding each feature, and they were combined in a max pooling fashion to form the feature vector of each image.

In [75], Perronnin et al. propose Fisher Vector representation, which is an alternative way of aggregating the local features to global representation based on the theory of Fisher Kernel [46]. The local features are first modeled with Gaussian Mixture Model (GMM). And each local feature is described by the gradient of its log-likelihood over the means and variances of GMM models. The representation of one image is obtained by aggregating the descriptor of all its local features. In BoVW, the k-means clustering and hard assignment can be considered as a vector quantization procedure, which only describes local distance information. In contrast, the Fisher Vector uses the gradients over parameters to provide much richer and more accurate information. Fisher Vector is shown to significantly outperforms BoVW in several standard datasets.

In [48], Jegou et al. propose Vector of Locally Aggregated Descriptors (VLAD), a simplified, non-probabilistic version of Fisher Vector. In VLAD, the local features are clustered with k-means and hard assigned to their nearest center as in BoVW. Then the difference between each feature and its nearest center is aggregated as the representation of the whole image. This representation can be considered as only describe the gradient over the mean of Fisher Vector model.

Comparing to BoVW, the only disadvantage of Fisher Vector is its dimensionality is much higher

than BoVW. To address this problem, in [76] the Fisher Vector representation is compressed with Product Quantization (PQ) technique. They show that the Fisher Vector can be significantly compressed with very little drop of accuracy.

2.2 Label Tree

For a multi-class classification problem with N classes, the inference complexity of traditional multi-class classifier is at least $O(N)$. Although the non-linear kernel SVM achieves best accuracy in general, its complexity is proportional to the number of support vectors for each class, therefore is not scalable for large scale problems. For linear multi-class SVM, since SVM is a binary classifier, multi-class problems need to be reduced to a series of binary classification problems to which SVM can be directly applied [43][19]. The complexity of multi-class SVM is dependent on the method of reduction. For one-vs-all method, in which a binary classifier is trained for each class against all the other classes, the complexity is $O(N)$; while for one-vs-one method, in which a binary classifier is trained for each pair of classes, the complexity is $O(N^2)$. For multi-class logistic regression model, which is naturally a multi-class classifier, the complexity is also $O(N)$.

To further reduce the inference complexity to sub-linear with N, researchers have been focusing on building a hierarchical tree structure of classifiers. In the tree structure, each node is associated with a set of classes. The root node includes all the classes. Each class of one node is assigned to one of its children, and a classifier is trained to distinguish the classes of its children nodes. In the traditional "flat" structure such as one-vs-all SVM and multi-class logistic regression model, each binary classifier is only able to distinguish one class, while in hierarchical tree structure, multiple classes can be separated (discarded) with one binary classifier. In this way, the overall complexity can be reduced to $O(log(N))$.

The essential challenge of building a hierarchical label tree model is how to learn the structure of the tree recursively, namely, for each node, how many children it should have and how to assign its classes to its children. This problem is equivalent to performing a hierarchical partitioning/clustering over the classes. In ideal scenario, the classes of one node should be partitioned in a way that the classes inside each child is relatively more confusing and the classes between children are more separable, so that least error is made at this node. However, there is no standard measurement for level of confusion between classes, and the different clustering algorithms can lead to different performance.

One class of methods use the distribution of feature vectors to cluster the categories. In [90] and [60], each class is represented by the mean of the feature vectors of its samples, and the classes are clustered with simple k-means algorithm. In [12], the distance between two classes are measured as the Kullback-Leibler distance [52] between their density functions. A max-cut algorithm is employed to obtain the partition of classes. In [101], a separability measurement based on support vector data description is adopted to represent the distances between classes, and the classes are clustered in a agglomerative fashion. Another class of methods measure the distance between categories by the confusion matrix of classification. In [35] and [7], a confusion matrix is obtained by cross-validation on the training set with traditional one-vs-all SVM, and the classes are partitioned with spectral clustering algorithm [70]. Comparing to feature distribution based clustering, the confusion matrix provides a better description of separability in a discriminative fashion, while at the cost of requiring expensive training with traditional SVM.

The most accurate way of performing the classes partition is to learn the partition and the classifier for each node simultaneously. Since the ultimate goal of partitioning is to reduce the probability of assigning the samples to the wrong child node, minimizing a loss function based on the classification error of node classifier achieves this goal directly. This strategy is adopted in both [16] and [28]. In both work, a max-margin based classifier is learned while the classes in each node are

optimized as latent variable.

In the hierarchical label tree framework, the number of classes in each node decreases as the tree grows deeper. For the root node, each of its children may contain hundreds of classes, while for the parents of leaf nodes, each of its children only contains one class. This lead to the problem that the nodes close to the root needs to solve more difficult classification problems since the feature distribution is more complex. With a hard partition of classes, the non-separable classes will inevitable suffer mis-classification. To alleviate this problem, a relaxed partition scheme with overlapping is proposed in [68] and also adopted in [28] and [16]. In the relaxation scheme, the classes that are not separable in current node are assigned to all of its children. In this way, the most separation of most confusing classes can be postponed to the deeper nodes, which have much less number of classes in its children. The level of relaxation, namely, the amount of overlapping, controls the balance between accuracy and speed. More overlapping makes the system less easy to make false classification, at the same time increases the number of classes in the children nodes.

The label embedding technique [7][26][2][32] can also be used to reduce the inference complexity from an orthogonal perspective. The original label space is mapped with a linear function to an embedded space, in which the distances between labels are better defined. The label embedding is commonly used to solve the multi-label classification problems. The inference complexity is only proportional to the dimension of the embedded space. Therefore, one can achieve lower complexity when the dimension of the embedded space is lower than the number of labels.

2.3 Convolutional Neural Networks

The famous paper by Krizhevsky et al. in 2012 [50] is a milestone for large scale visual recognition. They are the first one who successfully use deep neural networks to significantly outperform

the BoVW framework in visual recognition problems. They combined several clever techniques to train a powerful CNN model efficiently. First, they substitute Rectified Linear Unit (ReLU) function for traditional sigmoid function as the non-linear neuron, and significantly improve the network's converging speed. Second, to avoid overfitting, they adopt two effective strategies: (1) the input images are augmented by random cropping and color jittering; (2) In fully connected layers, the output of the neurons are randomly set to zeros. Third, they implement a GPU-based training framework that is several times more efficient than CPU counterpart.

Following their work, enormous effort has been made to improve the performance and efficiency of CNN [99] [81][78][38] [80] [45][39]. Table 2.1 lists the accuracies of recent deep CNN models and their depth. While the top-5 error rate has been reduced from 15.4% to as low as 3.1% in only a few years, the depth and complexity of the networks are consistently shown to be essential for better performance.

Table 2.1: Performance of Recent Deep CNN models.

Year	Method	Top-5 Error	Depth
2012	Alexnet [50]	15.4%	8
2013	Clarifai [99]	11.7%	8
2014	VGG [78]	7.3%	19
2014	GoogLeNet [81]	6.7%	22
2015	PReLU [39]	4.9%	22
2015	Batch Normalization [45]	4.8%	33
2015	Inception-v3 [82]	3.6%	46
2015	Residual Learning [38]	3.6%	152
2016	Inception-v4 [80]	3.1%	76

The structure of CNN models are highly flexible. The convolutional kernel sizes, the feature map sizes, the number of channels and the number of layers can all be configured. While the structure of Alexnet is just one possible configuration that is limited by the computing resources, researchers have adjusted the configuration of CNN models to achieve better balance of accuracy

and efficiency. Zeiler et al. [99] propose a visualization technique that shows the function of each intermediate layer, and adjust the kernel size and stride of lower convolutional layers so that they can retain much more useful information. In [58], the fully connected layers are substituted with global average pooling, which significantly reduce the memory required to store the model with almost no loss of accuracy. In [78], 3×3 convolutional kernels are exclusively adopted to build a 19 layers network. The network achieves state-of-the-art accuracy in ILSVRC 2014 competition, although the amount of computation is also extremely high. Within the same competition, Szegedy et al. introduce an architecture codenamed "Inception". In each "Inception" layer, convolutional layers with various kernel size and a max pooling layer are performed on the same input feature map, and the outputs are concatenated. To reduce the amount of computation, the number of input channels is first reduced by a 1×1 convolutional layer before fed to convolutional layer with larger kernel size. With carefully crafted design, their model called GooLeNet achieves better accuracy than [78] with several times less computation. In [38], a bottleneck structure, in which both the input and output channel dimensions of 3×3 convolutional layers are reduced, is proposed to further reduce the computation. In [82], Szegedy et al. use stacked $1 \times n$ and $n \times 1$ convolutional layers to replace $n \times n$ convolutional layers. This adjustment makes it practical to include kernels as large as 1×7 and 7×1 with acceptable complexity.

When the depth of CNN models increases, the networks get harder to converge. Initialization plays an important role at the training of deep CNN models. The parameters of each layer are randomly initialized with Gaussian distribution. Too large parameters lead an explosion in the last layer and make the network diverge, while too small parameters will lead to gradient vanishing and make the network converges extremely slow.

To solve this problem, Glorot et al. [31] analyze the behavior of the activation of each layer, and propose a normalized initialization method, so that in the forward pass, the variance of output remains the same as input, while in the backward pass, the variance of the input gradient remains

the same as the output gradient. He et al. [39] further improve the idea by considering the non-linear neuron function (ReLU)'s influence over the variance of the output.

In contrast, Szegady et al. [45] address this problem from a different perspective. Instead of making smart initialization, they explicitly normalize the input of each layer so that their statistical distribution remains the same. The statistics for each layer is calculated based on every mini-batch, and the normalization process can be considered as a part of the network so that it can be trained smoothly with back propagation. The mini-batch based normalization also adds random noise to the system and effectively prevents overfitting, so that dropout is not required anymore. With such a normalization mechanism, the learning rate of the SGD can be tuned much higher and lead to significantly faster converging speed.

For traditional CNN models, even though the problems of overfitting and vanishing/exploding of gradients are addressed by the techniques mentioned above, the increasing of depth cannot always convert to higher accuracy. This degradation is observed in [37][79] and thoroughly analyzed in [38]. To address this problem, He et al. propose a residual learning framework in [38]. For each convolutional layer with same input and output dimensions, the input is added to the output so that the convolutional layer only needs to learn the residual information. With residual learning, they successfully train a 152 layer CNN model that achieves state-of-the-art accuracy in ILSVRC 2015 competition.

2.4 Speeding-up CNNs

Most of the methods of accelerating CNNs are based on low rank decomposition. The convolutional kernel for each layer can be considered as a 4D tensor. The kernel can be approximated by multiplication and summation of tensors with lower-rank. Due to the various possible ways of per-

forming tensor decomposition, a few different methods have been proposed to decompose the convolutional layers. Denil et al. [17] treat the convolutional kernel as a matrix and decompose it into two lower-rank matrices. A few strategies are proposed to efficiently build one of the decomposed matrix to match the structure of the weight space, and the other matrix is learned by minimizing the loss of the whole network. In [47], two schemes of decomposing each convolutional kernel is proposed. In the first scheme, each kernel is decomposed into a pair of separable filter along spatial dimensions and one vector along the channel dimension; In the second scheme, each kernel is decomposed to one separable filter along one spatial dimension and one matrix along the other spatial dimension and the channel dimension. The reconstruction of the output data rather than the kernels are minimized. Experiments show that the second scheme achieves better speedups with similar reconstruction error. In [18], Denton et al. introduced two types of approximation methods: monochromatic approximation and bi-clustering approximation. In monochromatic approximation, the color input channels are projected to a group of grayscale channels, and the each output channel is only dependent on one grayscale channel. In bi-clustering approximation, both the input and the output channels are grouped into equal size clusters, and the kernels between each input group and each output group is approximated with low rank decomposition. In [56], Lebedev et al. used a classical low-rank CP-decomposition to decompose the 4D convolutional kernels to a sequence of 4 1D convolutional kernels, among which the first three are along the input channel dimension and two spatial dimensions, and the last one sums the output of rank-1 kernels. Such a straightforward decomposition makes the algorithm easy to implement and fine-tune. The network is fine-tuned after the original kernel is substituted with the decomposed 1D kernels. In [100], Zhang et al. consider the effect of non-linear neurons over decomposition and propose an optimization methods that minimize the reconstruction error of the nonlinear responses when performing low-rank decomposition.

Comparing to convolutional layers, the fully connected layers consume much more amount of

memory in early CNN models, but only require very low percentage of computations. A few methods have been proposed to reduce the memory consumption and computation of fully connected layers. Due to large amount of parameters, the fully connected layers possess more redundancy and can achieve much higher compression level than convolutional layers. In [98], Yang et al. propose to use adaptive fast food transformation to re-parameterize the fully connected layers. Their decomposition scheme does not require to pre-train the model and is end-to-end trainable. Cheng et al. [72] convert the dense matrix in fully connected layer to the Tensor Train format [74]. In [13], the dense matrix is substituted with a circulant matrix, whose number of parameters is significantly reduced. With circulant matrix, the fully connected layer can be implemented more efficiently with a FFT style algorithm. In [11], a method inspired by feature hashing is proposed to encode the fully connected layer. The weights in the dense matrix share a few fixed values and the assignment of values are determined by a hash function over the location of the weights. In [33], Gong et al. compare the compressing performance of matrix factorization, binarization, k-means, product quantization and residual quantization, and conclude that product quantization achieves the best overall compression.

In [36], Han et al. propose a framework that consists of small weights pruning, weight quantization and sharing, and Huffman coding. Such a framework achieve extremely high level of network compression and significant speedup for both convolutional layers and fully connected layers. Note that the weight pruning method in [36] is very similar to the sparse regularization in Chapter 4, although they do not perform any decomposition over the kernel, and their work was published later than our paper on which Chapter 4 is based on.

There are also several works that try to optimize the speed of CNN from other perspectives. Vanhoucke et al. [85] studies CPU based general neural network speed optimization. They discuss the usage of SIMD instructions, alignment of memory, as well as fixed point quantization of the network. Mathieu et al. [69] proposes to utilize FFT to perform convolution in Fourier domain. They

achieve 2x speedup on Alex net. Their method prefers a relatively large kernel size due to the overhead of FFT. Farabet et al. [24] implement a large scale CNN based on FPGA infrastructure that can perform embedded real-time recognition tasks. In [53], the authors improved the efficiency of convolutional layers by exploiting the algebraic structure of the convolution. Their method involved no approximation and can be used in any convolutional layers. They used minimal filtering algorithm to reduce the amount of multiplications for each convolutional kernels and improved the overall efficiency by a factor of 1.48.

2.4.1 Sparse Coding

Sparse coding approximates the input vector y by a sparse linear combination of items from an over-complete dictionary D. Due to the sparse natural of many computer vision problem, sparsity induced optimization has been consistently shown to perform exceptionally well in various computer vision problems, such as image denoising [21], image restoration [65], face recognition [95] and image classification with BoVW framework [97].

The sparsity penalty can be mathematically modeled as l_0 norm of the coefficient. However, minimizing the l_0 norm directly is an NP-hard problem and can only be solved approximately. Therefore, l_1 norm is proposed as a relaxation [83][10], which have been shown to also induce sparsity. With l_1 norm penalty, the corresponding optimization problem is convex and have a global optimal solution.

Large amount of methods have been proposed to solve the sparse decomposition problem with accuracy and efficiency. Greedy l_0 penalty based methods include matching pursuit [67], orthogonal matching pursuit [29], iterative hard-thresholding [8] et al.; l_1 penalty based methods include coordinate descent [27], iterative shrinkage thresholding [5], LARS [20] et al. In [3], Bach et al. give a good overview of the methods used for sparse decomposition.

The performance of sparse coding is highly dependent of the quality of the chosen dictionary. The optimal way of choosing dictionary is through learning from input data. Various methods have been proposed to learn the dictionary by updating the sparse coefficients and the dictionary iteratively [1][57][64].

Previous work on sparse matrix computation focus on the sparse matrix dense vector multiplication (SpMV) problem. The sparse matrix is stored with various formats, such as CSR [4] and ESB [61], for efficiency. Blocking is adopted in register [44] and cache [71] level to improve the spatial locality of sparse matrix. To further reduce bandwidth requirement, various techniques including matrix reordering [73], value and index compression [94] are proposed. We refer readers to [34] for a more comprehensive review.

CHAPTER 3: PROBABILISTIC LABEL TREE

3.1 Overview of a Label Tree

In this section, we briefly review how a label tree model operates and previous work on learning the tree's parameters. Following previous work in [6, 16], we will focus on a label tree with linear classifiers at each node of the tree.

Following the notation in [16], a label tree is a tree that has nodes V and edges E, such that the tree $T = (V, E)$. The children of one node r are contained in the set $\sigma(r)$. Every child node, c, is associated with a vector of weights w_c that are used to select which child node will be visited during the inference process. Each leaf node s is also assigned with a single label, $l(s)$, that specifies the label that is assigned to the example if s is reached as a leaf node.

Algorithm 1 Classifying a test example with the traditional label tree algorithm.

Require: Test example $\boldsymbol{x}$, label tree parameters T, σ, l

1: Initialize s to the root node
2: **while** $\sigma(s) \neq \emptyset$ **do**
3: $\quad s \leftarrow \underset{c \in \sigma(s)}{\operatorname{argmax}} \boldsymbol{w}_c^\top \boldsymbol{x}$
4: **end while**
5: Assign the label $l(s)$ to the test example

As shown in Algorithm 1, a new test example is classified by traversing the tree from the root node. Except for the leaf nodes, all the nodes are treated identically. At a non-leaf node s, classification scores are computed for every child node of s by multiplying the feature vector $\boldsymbol{x}$ of the input sample and the weights w_c for the child node c. The child node with the highest classification score is visited next. As described above, if a leaf node is arrived while traversing the tree, then the test example is assigned the label $l(s)$, which is associated with visited leaf node s.

3.2 Learning a Probabilistic Label Tree

We propose a probabilistic approach to learn the parameters of label tree. As will be shown in the experimental results in Section 3.6, this probabilistic model produces higher recognition accuracy than traditional label tree models.

In the probabilistic label tree, each node, s, is associated with a categorical distribution $p[y|S]$, where y denotes a valid label and S denotes the event that the classification process arrives at node s.

This categorical distribution is then combined with a probabilistic classifier that is defined by the classification weights at each node to compute the probability of assigning the test image to a particular label, given the feature vector $\boldsymbol{x}$.

3.2.1 Defining the Learning Criterion

Defining the label tree framework as a probabilistic model makes it natural to learn the parameters with a maximum likelihood approach. The first step is to define the probability of assigning label y to the example, beginning at the root node, r. This can be expressed as

$$p(y|\boldsymbol{x}) = \sum_{c \in \sigma(r)} p[y|S_c] P[S_c|\boldsymbol{x}] \tag{3.1}$$

where $\sigma(r)$ is the set of all children nodes of r, as in Algorithm 4, S_c denotes the event of choosing to visit the child node c next; $P[S_c|\boldsymbol{x}]$ denotes the probability visiting c given sample $\boldsymbol{x}$, and $p[y|S_c]$ denotes the probability of assigning label y given the system chooses node c.

To be consistent with our focus on linear classifiers, the conditional probability $P[S_c|\boldsymbol{x}]$ is defined

as a multinomial logistic regression model:

$$P[S_c|\boldsymbol{x}] = \frac{e^{\boldsymbol{w}_c^\top \boldsymbol{x}}}{\sum_{i \in \sigma(r)} e^{\boldsymbol{w}_i^\top \boldsymbol{x}}}. \tag{3.2}$$

It should be noted, though, that any conditional probability model could be applied here.

This model can be considered as breaking the classification into two steps. At the root node, the classification vectors are first applied to the feature vector $\boldsymbol{x}$ to choose which child node to visit. The example's final label is then determined by the categorical distribution of the labels in the child node.

3.2.2 Recursively Expanding the Label Tree Model

If there are K categories and each node has n children, then the tree needs at least $\log_n K$ levels of child nodes to guarantee that every possible category is associated with at least one leaf node. Having fewer leaf nodes than categories will lead to ambiguous results since some leaf nodes are forced to represent more than one classes.

One can add new levels to the distribution in Equation (3.1) by expanding each $p[y|S_c]$ term recursively. If c^1 represents the chosen child node at the first level and c^2 represents a child node of c^1, then a two levels model would have the form

$$p(y|\boldsymbol{x}) = \sum_{c^1 \in \sigma(r)} \sum_{c^2 \in \sigma(c^1)} p[y|S_{c^2}] P[S_{c^2}|S_{c^1}, \boldsymbol{x}] P[S_{c^1}|\boldsymbol{x}]. \tag{3.3}$$

$P[S_{c^2}|S_{c^1}, \boldsymbol{x}]$ has the same multinomial logistic regression form as Equation (3.2). Additional levels can be further added in a similar fashion.

3.2.3 Building the Tree Stagewise

If given unlimited computing resources, the label tree parameters could be learned by recursively expanding Equation (3.3) to the desired number of levels and maximizing the log of Equation (3.3) with a continuous optimization method. However, as the number of levels increases, the number of parameters as well as complexity of training grow exponentially. Therefore, for practical reasons, it is necessary to be able to independently train the linear classifier at each node.

Returning to the probability of the test sample at the root node in Equation (3.1), once the classifier weights and the parameters of the categorical distributions $p[y|S_c]$ have been found, one can apply Jensen's Inequality to the log of Equation (3.1) to calculate a lower bound on $p(y|\boldsymbol{x})$.

$$\begin{aligned} \log p(y|\boldsymbol{x}) &= \log \left[\textstyle\sum_{c \in \sigma(r)} p(y|S_c) P[S_c|\boldsymbol{x}] \right] \\ &\geq \sum_{c \in \sigma(r)} P[S_c|\boldsymbol{x}] \log \left[p(y|S_c) \right] \end{aligned} \tag{3.4}$$

Defining this lower bound as L,

$$L = \sum_{c \in \sigma(r)} P[S_c|\boldsymbol{x}] \log \left[p(y|S_c) \right], \tag{3.5}$$

the categorical distribution at each of the child nodes can be recursively expanded to a new linear classifier and corresponding new set of child nodes. Using this bound, the log probability of the two level expansion shown in Equation (3.3) becomes

$$L = \sum_{c^1 \in \sigma(r)} P[S_{c^1}|\boldsymbol{x}] \log \left[\sum_{c^2 \in \sigma(c^1)} p(y|S_{c^2}) P[S_{c^2}|S_{c^1}, \boldsymbol{x}] \right] \tag{3.6}$$

for each training example.

In this equation, each term inside the log function corresponds to one child node. Since each of the terms in the summation in the right hand side of Equation (3.6) has its own set of parameters, L will be maximized by maximizing each of the terms individually. This decouples loss function of the child nodes during training and makes it possible to learn the parameters for each child node independently.

This also shows that learning the parameters of the probabilistic label tree model can be considered as a stage-wise lower-bound maximization of the log likelihood function for the linear classifiers.

3.3 Final Algorithm for Learning a Probabilistic Label Tree

The process of learning the probabilistic label tree can be viewed as a recursive expansion of nodes. Starting at the root node, each non-leaf node is expanded into a set of children nodes by iteratively performing: (1)Learning the weights of the multinomial logistic regression model with maximum likelihood based on the categorical distribution of each child node (2) Updating the categorical distribution of each child node with the logistic regression model fixed.

3.3.1 Learning Parameters for an Expanded Node

For a training set with N training samples, $(y_i, \boldsymbol{x}_i)$, this is equivalent to optimizing the log of Equation (3.1) over all examples, for an arbitrary node.

The overall training process is more easily specified by defining a general loss for expanding a node s, given N training samples of the form $(y_i, \boldsymbol{x}_i)$, expressed as:

$$L = \sum_{i=1}^{N} \alpha_i \log \left[\sum_{c \in \sigma(s)} p(y_i|S_c) P[S_c|\boldsymbol{x}_i] \right] \tag{3.7}$$

This log likelihood can be maximized with two alternating convex optimizations. In the first step, $p(y_i|S_c)$ is held constant and the loss function is similar to training a multinomial logistic regression model, or softmax classifier. Then, $P[S_c|\boldsymbol{x}_i]$ is held constant, and the optimization is essentially equivalent to maximum-likelihood estimation of the categorical distribution parameters. We have found that running this optimization for a fixed number of iterations, typically under 10, works well. In our implementation, we use a weighted k-means algorithm to generate an initial set of clusters that can be used to initialize the categorical distributions.

Algorithm 2 Algorithm for Learning Node Parameters

Require: N training pairs $(y_i, \boldsymbol{x}_i)$, maximum number of levels, L, branching factor B, weight α
Test example $\boldsymbol{x}$, label tree parameters T, σ, l

for $l = 0 \ldots L-1$ **do**
  for all nodes s in $\lambda(l)$ **do**
    Create B child nodes, except at final level (see Sec. 3.5.2)
    $\alpha_i \leftarrow 1 \cdot \prod_{t \in \psi(l)} P[S_t|\boldsymbol{x}_i], \forall i \in \{1, \ldots, N\}$
    for all nodes c in $\sigma(s)$ **do**
      for M iterations **do**
        Fix the parameters for $P[y|S_c]$, maximize Equation (3.7) over classifier parameters.
        Fix the parameters for classifier parameters $\boldsymbol{w}$, maximize Equation (3.7) over the parameters of $P[y|S_c]$.
      end for
    end for
  end for
end for

3.3.2 Formal Specification of the Algorithm

Our formal specification of the algorithm for constructing the label tree will involve two sets of functions. First, we will define $\psi(s)$ to be the set of nodes that must be traversed before arriving at node s, or the path to s. Second, we will define $\lambda()$ to be the set of nodes at a given level of the tree. This will start at 0, with $\lambda(0)$ only containing the root node, $\lambda(1)$ containing the next level of

branch nodes, and so on. Algorithm 2 shows the recursive process for expanding the branch nodes.

3.4 Understanding the Expansion Process

Figure 3.1 shows examples of how the expansion process operates on the ILSVRC2010 database discussed in Section 3.6. Figure 3.1(a) shows the categorical distribution at a branch node in the second level of a T6,4 tree that has four levels, not counting the root node. Figure 3.1(b) - 3.1(d) show several of the six child nodes of this branch node. In this next level, the classes with the highest probability at the branch node are distributed among the children. The final row in these tables also shows that probability has become more concentrated in the most likely classes.

3.4.1 Balanced Trees and Efficiency

Increasing classification efficiency is the primary motivation behind the label tree model. For the models based on linear classifiers, this efficiency is best measured by the average number of dot products needed to produce a final label. The number of dot products depends on the number of leaf nodes, which is dependent on how balanced the tree is. A perfect balancing of the probabilities for each class across the nodes of the tree would minimize the number of leaf nodes needed.

This raises the question of whether the maximum-likelihood approach proposed here will find a balanced tree. A maximum-likelihood approach attempts to learn a model which induces a label distribution similar to the one in the data. Depending on the underlying distribution, this will not always produce a balanced label tree, and one can certainly construct artificial counter-examples. However, in most natural applications, it is reasonable to assume that the label distribution can be well approximated by a reasonably-balanced label tree, where the leaf nodes distribution concentrate on individual classes.

(a) Distribution at Parent Node before Expansion

Class	Prob.
chickpea	0.019
hazelnut	0.019
lentil	0.017
kidney bean	0.016
brussels sprouts	0.016
coffee bean	0.015
clam	0.015
peanut	0.015
plum	0.015
mashed potato	0.014
soy	0.014
orange	0.014
cashew	0.014
lemon	0.014
walnut	0.014
Rem. 985 classes	0.771

(b) Distribution at a Child Node after Expansion]

Class	Prob.
hazelnut	0.075
lentil	0.067
kidney bean	0.064
peanut	0.062
cashew	0.057
soy	0.056
coffee bean	0.055
peanut	0.055
green pea	0.055
pumpkin seed	0.054
pistachio	0.051
walnut	0.040
pea	0.038
corn	0.032
bean	0.028
Rem. 985 classes	0.210

(c) Distribution at a Child Node after Expansion]

Class	Prob.
chickpea	0.107
mashed potato	0.084
clam	0.081
brussels sprouts	0.074
shrimp	0.071
okra	0.038
french fries	0.035
acorn squash	0.035
broccoli	0.034
cucumber	0.031
spaghetti squash	0.028
celery	0.025
shiitake	0.025
black olive	0.020
bok choy	0.019
Rem. 985 classes	0.292

(d) Distribution at a Child Node after Expansion]

Class	Prob.
lemon	0.061
orange	0.060
plum	0.056
persimmon	0.056
guava	0.055
mango	0.055
shallot	0.052
kumquat	0.052
Granny Smith	0.049
turnip	0.047
quince	0.046
bell pepper	0.045
butternut squash	0.032
fig	0.025
spaghetti squash	0.023
Rem. 985 classes	0.285

Figure 3.1: This figure visualizes the expansion process for one part of the tree. The table in (a) shows a portion of the categorical distribution at a branch node in the second level of a T6,4 tree that has four levels, excluding the root node. The tables in (b), (c), and (d) show the categorical distributions learned for three of the child nodes. The probability in these child nodes is more concentrated on a subset of the classes than in the parent node. In this result, the tree has not been trained with the pruning techniques in Section.3.5.1, so the distributions accurately represent the behavior of the maximum-likelihood criterion.

A simple way to demonstrate this is to show that if the data distribution indeed corresponds to a balanced label tree, then the maximum-likelihood approach would learn a similar balanced tree. Note that this is not altogether trivial: There might be a very unbalanced label tree, which induces the exact same conditional distribution $p(y|\boldsymbol{x})$ as the balanced label tree, so a maximum-likelihood approach might learn the unbalanced tree instead. To show that this isn't the case, we prove below that our model is *identifiable* - namely, that under mild conditions, given enough data from a distribution induced by a given label tree, then our algorithm would learn the exact same tree.

Theorem 1. *Suppose the training data is sampled i.i.d. from a distribution, such that $p(y|\boldsymbol{x})$ is generated by some label tree, for which $\boldsymbol{w}_c \neq w_{c'}$ for any two sibling nodes c, c'. Also, suppose that the support of $p(\boldsymbol{x})$ is continuous in some part of the domain. Then as the dataset size increases, the structure and weights of the label tree learned by our algorithm converges to those of the true label tree*

Proof. It is enough to show that we can perfectly reconstruct the root node of the tree - the reconstruction of its child nodes and other nodes in the tree would follow in a similar way by induction. In the limit of infinite data, this boils down to showing that the label distribution $p(y|\boldsymbol{x})$, which can be written as

$$\sum_{c\in\sigma(r)} p(y|S_c)P[S_c|\boldsymbol{x}] = \sum_{c\in\sigma(r)} p(y|S_c)\frac{e^{\boldsymbol{w}_c^\top \boldsymbol{x}}}{\sum_{i\in\sigma(r)} e^{\boldsymbol{w}_i^\top \boldsymbol{x}}}$$

can be induced only by a single choice of the parameters $\{\boldsymbol{w}_c, p(y|S_c)\}_{c\in\sigma(r)}$. Let us assume on the contrary that there exist some other set of parameters $\{\boldsymbol{w}'_{c'}, p'(y|S_{c'})\}_{c'\in\sigma'(r)}$ (possibly corresponding to a different number of child nodes), which induce the same distribution, namely

$$\sum_{c\in\sigma(r)} p(y|S_c)\frac{e^{\boldsymbol{w}_c^\top \boldsymbol{x}}}{\sum_{i\in\sigma(r)} e^{\boldsymbol{w}_i^\top \boldsymbol{x}}}$$
$$= \sum_{c'\in\sigma'(r)} p'(y|S_{c'})\frac{e^{\boldsymbol{w}_{c'}'^\top \boldsymbol{x}}}{\sum_{i'\in\sigma'(r)} e^{\boldsymbol{w}_{i'}'^\top \boldsymbol{x}}}$$

for any $\boldsymbol{x}$ in the support of $p(\boldsymbol{x})$. Taking a common denominator, and switching sides, this is equivalent to requiring

$$\frac{\sum_{c\in\sigma(r),c'\in\sigma'(r)}(p(y|S_c)-p'(y|S_{c'}))e^{(\boldsymbol{w}_c+\boldsymbol{w}'_{c'})^\top\boldsymbol{x}}}{\sum_{c\in\sigma(r),c'\in\sigma'(r)}e^{(\boldsymbol{w}_c+\boldsymbol{w}'_{c'})^\top\boldsymbol{x}}}=0.$$

on the support. Since the denominator is always positive, this is equivalent to

$$\sum_{c\in\sigma(r),c'\in\sigma'(r)}(p(y|S_c)-p'(y|S_{c'}))e^{(\boldsymbol{w}_c+\boldsymbol{w}'_{c'})^\top\boldsymbol{x}}=0.$$

The left hand side is identically zero if $|\sigma(r)| = |\sigma'(r)|$ (namely, there are the same number of child nodes), $\boldsymbol{w}_c = \boldsymbol{w}'_{c'}$, and $p(y|S_c) = p'(y|S_{c'})$ for all c, c' (up to permutation of the c, c' labels). If this is not the case, then the equation above can be rewritten as

$$\sum_{i=1}^{N} a_i e^{b_i^\top \boldsymbol{x}} = 0,$$

for some finite N, distinct $\{b_i\}$, and $\{a_i\}$ such that $a_i \neq 0$. However, if the support of $p(\boldsymbol{x})$ is dense in some neighborhood, and the equation holds in that domain, then the left hand side can be shown to equal 0 for *all* $\boldsymbol{x}$ in Euclidean space, which is easily seen to be impossible. Therefore, the parameters that we will learn indeed correspond to the actual label tree. □

3.5 Implementation

As described in Section 3.3.1, learning the label tree parameters at each node consists of two alternating steps: learning the weights of logistic regression model, and learning the parameters of categorical distribution. The classification weights were optimized using the limited memory BFGS (L-BFGS) algorithm. We also experimented with the stochastic meta-descent (SMD) algo-

rithm [9], which has been shown to perform better than traditional stochastic gradient descent [89]. We found that using L-BFGS converged faster and to better values for the training criteria.

3.5.1 Eliminating Samples During Training

In [16], each child node is assigned only a specific subset of categories during the training process. An advantage of this approach is that only the training examples that belongs to those categories need to be processed when learning the parameters of that node and all its offspring nodes.

In contrast, in the probabilistic model proposed here, the learning criterion depends on the probability that each example arrives at the node where the parameters are being learned. In practice, this probability for many examples is often quite small, but still non-zero, so the learning could require processing all of the training examples at every node in the label tree. To increase the speed of training, the training examples used at a node are pruned to eliminate examples that have a very low probability of reaching the node. This pruning is done independently for each node.

3.5.2 Fixing the Number of Leaf Nodes

Following [16], the label tree is constructed with a fixed number of levels. Ideally, the learning system would be able to find a perfectly balanced tree and each leaf node would correspond to one class. In practice, it is difficult to find a perfectly balanced tree due to the complex distribution of the categories, so the number of leaf nodes must be determined individually for each branch in the next to last level of the tree. A number of training examples could be assigned to a leaf node with very low probability, so a natural criterion is to assign one leaf per class for the set of classes that account for some percentage, such as 90%, of the probability in the categorical distribution.

For comparison or performance evaluation, it is also useful to have more direct control over the

number of leaf nodes. While this threshold could be adjusted to achieve a desired number of leaf-nodes, we found it easier to directly control this number by imposing a hard cap on the number of leaf nodes per branch node. In our experiments, this greatly simplified controlling how many dot products were necessary and performed well.

3.6 Results

We evaluated our algorithms on both the ILSVRC2010 and ImageNet10k datasets used in [16]. In ILSVRC2010, there are 1.2M images from 1K classes for training, 50K images for validation, and 150K images for test. Due to limitation of system memory, we randomly picked 300 images of each category for training. In ImageNet10k, there are 9M images from 10184 classes. We randomly picked 100 images from each category for training and 50 for testing. This is fewer than used in [16], but demonstrates that our method can scale to large problems.

We generated the feature vector for each image in the same fashion as [16] using the LLC coding strategy introduced in [91]. For each image in ILSVRC2010, we used the VLFeat toolbox [86], which was also used in [16] to extract dense SIFT features from the image. The features were encoded with a codebook with 10k entries generated with k-means and the image was encoded using a two-level spatial pyramid[54] with 1×1 and 2×2 grids. This resulted in a feature vector with approximately 50,000 dimensions. In the experiments with the ImageNet10K database, images were represented with vectors encoded using LLC, but without a spatial pyramid. The number of classes in ImageNet10K makes reducing the number of dot-products beneficial, even considering the time needed to generate features. In our experiments computing the classification scores dominated the computation time. The average time for generating features from one image was approximately 0.59 seconds, while computing the one-versus all linear classifier required approximately 2.09 seconds.

In our experiments, we did not make an effort to strictly control the computational complexity in the training phase. Training a label tree with our method on ILSVRC2010 took less than a day on a multi-core machine with 48GB of memory.

3.6.1 Comparison with Previous Work

Table 3.1 summarizes the accuracy of our system, compared with the trees trained using the method in [16]. The various columns of Table 3.1 represent different tree structures. The tree denoted by Tm, n has m children per node when branching and n levels, not including the root node.

Table 3.1: Comparison of our method and [16] with different tree configuration on ILSVRC 2010 dataset.Tm, n denotes the tree that has m children per node when branching and n levels. We show the classification accuracy(Acc) and the test time speedup S_{te}. The first row shows the result using our ML based method. The second row is the result using hard label partition and our probabilistic framework as explained in Section 3.6.2. Our method significantly outperforms [16]. The accuracy of maximum likelihood (ML) is consistently better than the hard label partition with similar speedup. For reference, the last row shows the accuracy produced by a multi-class SVM trained with LIBLINEAR.

Method	Flat		T32,2		T10,3		T6,4		T4,5	
	Acc%	S_{te}	Acc%	S_{te}	Acc%	S_{te}	Acc%	S_{te}	Acc%	S_{te}
Trained with ML	-	-	21.38	10.42	20.54	17.85	17.02	31.25	14.98	41.67
Trained with Hard Partition	-	-	20.70	10.37	19.15	17.90	15.76	31.27	14.85	33.84
Results in [16]	-	-	11.9	10.3	8.92	18.2	5.62	31.3	-	
LIBLINEAR	24.84	1	-							

To make a fair comparison, we tuned the number of leaf nodes, using the approach described in Section 3.5.2, so that the trees trained using our approach used a similar average number of dot-products to classify each example. As Table 3.1 shows, the accuracy rate is significantly higher for the trees that are trained using our approach. Depending on the depth of the tree, our approach classifies images with an accuracy rate that is nearly double or triple the accuracy rates produced using the method from [16]. Our approach is also superior at learning deeper trees. Comparing the accuracies for the $T32, 2$ and $T6, 4$ trees, the reduction in accuracy is less significant in the trees

trained using the probabilistic approach.

For reference, the last row of Table 3.1 also reports the classification accuracy of a multi-class one-against-all SVM classifier trained using LIBLINEAR [23]. The trees trained using maximum likelihood produce competitive results while requiring 18 to 30 times less dot products at test time.

We show the results of our method on the ImageNet10K dataset in Table 3.2. While the accuracies cannot be compared directly because of differences in the test set, these results show that this approach to learning trees can scale to larger problems and that our method produces reasonable performance.

Table 3.2: Result on Imagenet 10K. While the accuracy numbers cannot be compared directly because of differences in the test set(see text), these results show that our method can scale to large numbers of classes and performs reasonably.

Method	T101,2		T10,4	
	Acc%	S_{te}	Acc%	S_{te}
Trained with ML	4.77	33.22	3.08	204.54
Results reported in [16]	3.4	32.40	-	

3.6.2 Evaluating Hard Label Partitioning

One of the most significant differences between the proposed method and the method in [16] lies in how the labels are assigned to the child nodes. In our approach, the probability of each sample reaching a particular node is maintained while optimizing that node's parameters. In contrast, the learning approach in [16] adopts a binary partition matrix so that all examples from a class either reach the node or do not. We refer to this as *hard label partitioning* because the examples of every class are assigned a hard binary label describing whether they can reach a node. This partition matrix is found by optimizing a measure of accuracy with hard constraints on the average number of categories in the children nodes.

To explore whether this style of hard label partitioning can produce improved results, we evaluated the combination of our probabilistic framework with the partitioning strategy used in [16] to measure whether hard label partitioning is superior for learning the label tree. This was implemented by replacing the categorical distributions in our model with distributions derived from the partition matrix computed using OP3' in [16]. This binary partition matrix can be used to calculate a new distribution $p(y|S_c)$ in which every class assigned to a specific node is equally likely. For numerical stability reasons, classes not assigned to a node are given a very small probability. This results in a two-stage alternating algorithm that consists of learning a linear classifier in the form of a multinomial logistic regression classifier, and using the classifications from that classifier to learn the partition matrix.

The ambiguity constraints during learning were manually tuned so that the trees used a comparable average number of dot products to classify examples. Below, Section 3.6.3 will discuss performance across various parameter settings. As the second row of Table 3.1 shows, training the tree just using the maximum likelihood criterion consistently outperforms this approach. In addition, as the trees become deeper, the advantage of the maximum likelihood approach increases.

This result also provides insight into the significant performance differences in Table 3.1. Given that using OP3' from [16] reduced accuracy, but not dramatically, the combination of the multinomial logistic regression models with an L-BFGS optimization system probably accounts for most of the increase in performance over the results in [16]. The system in [16] uses several iterations of parallel Stochastic Gradient Descent [102] to learn the classifiers. In our experiments, we also found that using a stochastic gradient descent algorithm led to far worse classification performance than using the quasi-Newton style L-BFGS algorithm. It should be noted, though, that the max-margin criterion in OP2 in [16] is non-differentiable at points, due to the use of a max operator. So the framework in [16] cannot be trivially modified to use L-BFGS instead of stochastic gradient optimization.

3.6.3 Exploring the Accuracy-Efficiency Trade-off

In a traditional label tree model, one weakness of the classification procedure is that the classifier must make a hard choice to decide which branch to follow. In cases where there is ambiguity in the correct branch, this hard choice will degrade recognition accuracy if the wrong branch is chosen. An advantage of the probabilistic formulation is that it facilitates performing a limited search over multiple branches of the label tree.

In this limited search, the classifier does not just follow the best-scoring branch. Instead, it makes a decision at each branch whether to follow just the highest-scoring branch or both the highest-scoring and the second highest-scoring branches. This decision is made by comparing the difference in scores between the two branches. If this difference is below a threshold, both branches are followed. The final classification is found by adding together the categorical distributions at each of the leaf nodes that the classification algorithms reaches.

As this threshold varies, the average number of dot-products needed to produce a classification also varies. The green curves in Figures 3.2 and 3.3 show the relationship in different tree models between the average number of dot-products needed to classify a sample and the resulting accuracy. Notice that as the classification process explores more branches, the accuracy rises.

For comparison, the blue curves in these figures are computed by varying the ambiguity limits of the hybrid systems explained in Section 3.6.2. These curves are shorter because that as the ambiguity limits grows, the number of classes at deeper levels of the tree increases dramatically and the time required to train the system becomes untenable. In both tree configurations, the tree learned with our probabilistic approach provides higher accuracy with similar average number of dot products.

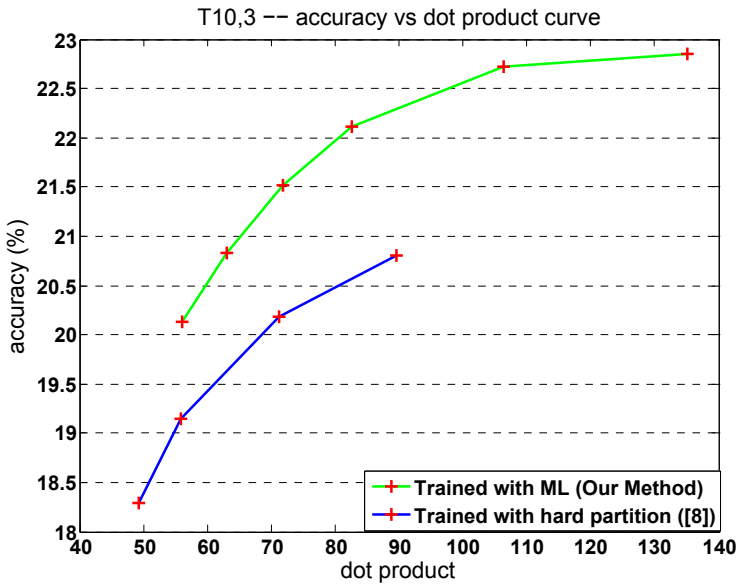


Figure 3.2: Accuracy vs dot product curve for our T10,3 tree. The green curve shows our method using maximum likelihood with multinomial estimation. The blue curve shows the result using hard label partition[16] with our framework. The maximum likelihood method achieves consistent higher classification accuracy with similar average dot products needed at test time

.

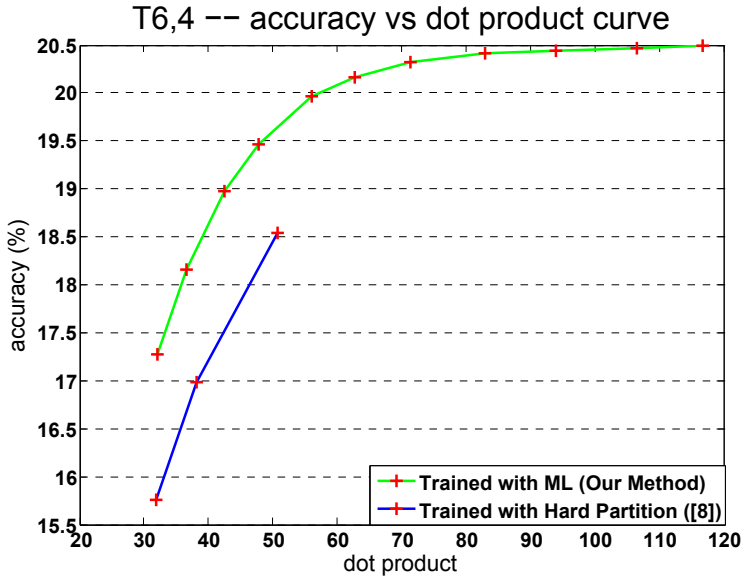


Figure 3.3: Accuracy vs dot product curve for our T6,4 tree. Again, the tree trained with the maximum likelihood approach has consistently higher accuracy.

It is important to note that with our model, creating this curve does not require retraining of the tree parameters. This approach can be used to adjust the accuracy/efficiency trade-off at inference time on an existing tree. In contrast, using the optimization with ambiguity constraints in [16] requires the tree to be re-trained for different parameters.

3.7 Summary

In this chapter, we propose a probabilistic label tree model to improve the efficiency of large scale classification problems. Since our method is purely probabilistic and optimized with maximum likelihood, it can be easily adapted to different types of probabilistic classifiers. Our experimental results demonstrate that the proposed probabilistic label tree is able to achieve higher recognition accuracy with comparable complexity to previous work.

CHAPTER 4: SPARSE CONVOLUTIONAL NEURAL NETWORKS

4.1 Method

4.1.1 Sparse Convolutional Neural Networks

Given the input feature maps $\mathbf{I}$ in $\mathbb{R}^{h\times w\times m}$, where h, w and m are the height, width and number of channels of the input, and the convolutional kernel $\mathbf{K}$ in $\mathbb{R}^{s\times s\times m\times n}$, where s is size of the convolutional kernel and n is the number of output channels. We assume the convolution is performed with no zero padding and stride equal to 1. Then, the output feature maps of a traditional convolutional layer $\mathbf{O}\in\mathbb{R}^{(h-s+1)\times(w-s+1)\times n}=\mathbf{K}*\mathbf{I}$ are computed by

$$\mathbf{O}(y,x,j)=\sum_{i=1}^{m}\sum_{u,v=1}^{s}\mathbf{K}(u,v,i,j)\mathbf{I}(y+u-1,x+v-1,i) \tag{4.1}$$

Our objective is to substitute the computationally expensive convolutional operation $\mathbf{O}=\mathbf{K}*\mathbf{I}$ in formula (4.1) with a more efficient sparsified version that can be implemented with multiplication of sparse matrices.

For this purpose, we first transform the input tensor $\mathbf{I}$ to $\mathbf{J}\in\mathbb{R}^{h\times w\times m}$ and the convolutional kernel $\mathbf{K}$ to $\mathbf{R}\in\mathbb{R}^{s\times s\times m\times n}$ using a matrix $\mathbf{P}\in\mathbb{R}^{m\times m}$, so that $\mathbf{O}\approx\mathbf{R}*\mathbf{J}$, and

$$\begin{aligned}\mathbf{K}(u,v,i,j)&\approx\sum_{k=1}^{m}\mathbf{R}(u,v,k,j)\mathbf{P}(k,i)\\ \mathbf{J}(y,x,i)&=\sum_{k=1}^{m}\mathbf{P}(i,k)\mathbf{I}(y,x,k)\end{aligned} \tag{4.2}$$

Next, for every input channel $i=1,\cdots,m$, we decompose the convolutional kernels that cor-

respond to this channel $\mathbf{R}(\cdot,\cdot,i,\cdot) \in \mathbb{R}^{s\times s\times n}$ into the product of matrix $\mathcal{S}_i \in \mathbb{R}^{q_i\times n}$ and tensor $\mathcal{Q}_i \in \mathbb{R}^{s\times s\times q_i}$, where q_i is the number of bases:

$$\begin{aligned}\mathbf{R}(u,v,i,j) &\approx \sum_{k=1}^{q_i} \mathcal{S}_i(k,j)\mathcal{Q}_i(u,v,k)\\ \mathcal{T}_i(y,x,k) &= \sum_{u,v=1}^{s} \mathcal{Q}_i(u,v,k)\mathbf{J}(y+u-1,x+v-1,i).\end{aligned} \tag{4.3}$$

so that

$$\mathbf{O}(y,x,j) \approx \sum_{i=1}^{m}\sum_{k=1}^{q_i} \mathcal{S}_i(k,j)\mathcal{T}_i(y,x,k) \tag{4.4}$$

Note that if we represent the tensor $\mathbf{O}$ and $\mathcal{T}_i$ as matrices by combining the first two dimensions, and concatenate both $\mathcal{S}_i$ and $\mathcal{T}_i$ along the dimension q_i, formula (4.4) can be implemented by a single matrix multiplication.

Here, we shall search for matrices $\mathbf{P}$, $\mathcal{Q}_i$ and $\mathcal{S}_i$, $i = 1,\cdots,m$, such that q_i are much smaller than s^2, matrices $\mathcal{S}_i$ have a large number of zero elements and columns, while our new sparse convolutional kernel $\mathbf{R}$ provides output that is close to the one obtained with the original kernel $\mathbf{K}$.

4.1.2 Computational Complexity

The theoretical complexity of our method is measured by the number of multiplications. For traditional convolutional layers, the required number of multiplication is:

$$m\,n\,s^2(h-s+1)(w-s+1) \tag{4.5}$$

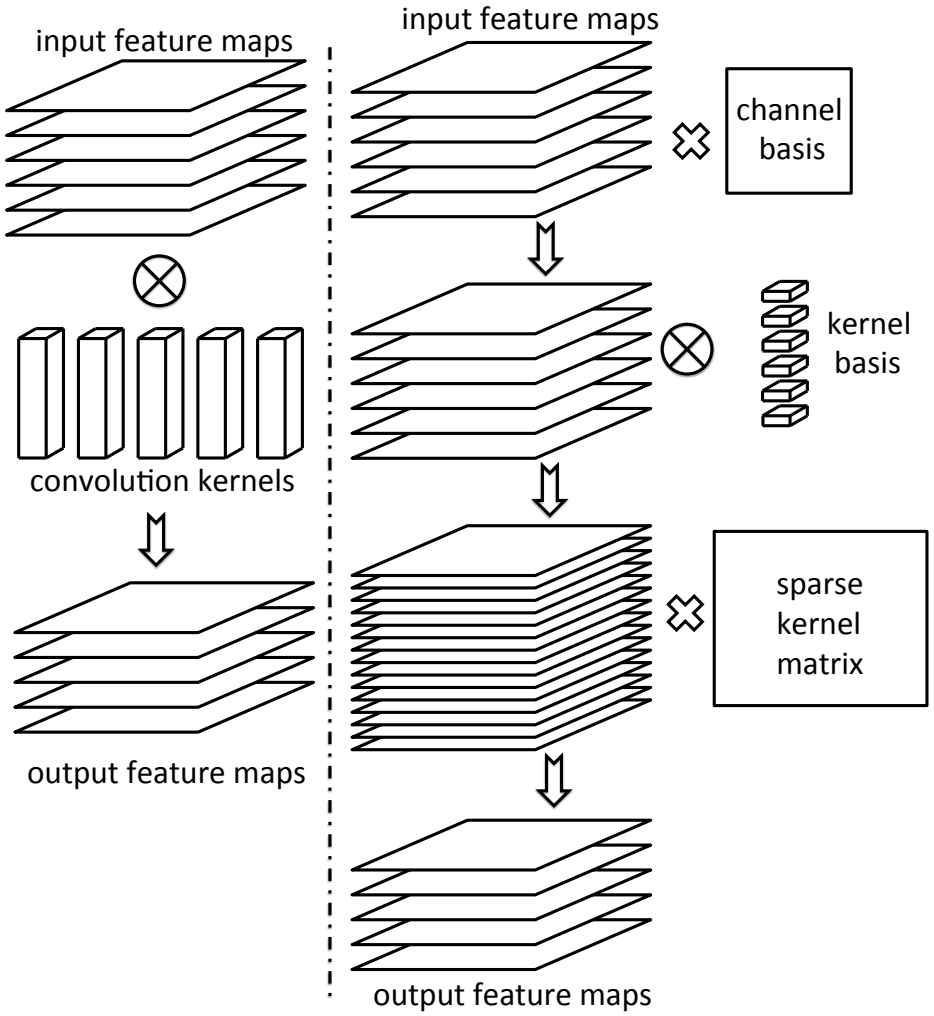


Figure 4.1: Overview of our sparse convolutional neural network. Left: the operation of convolution layer for classical CNN, which convolves large amount of convolutional kernels with the input feature maps. Right: our proposed SCNN model. We apply two-stage decompositions over the channels and the convolutional kernels, obtaining a remarkably(more than 90%) sparse kernel matrix and converting the operation of convolutional layer to spare matrix multiplication.

Our method significantly reduces the complexity by sparsifying the the convolutional kernel, while introducing only very small overhead of two matrix decompositions:

$$\left(\gamma mn + \sum_{i=1}^{m} q_i\right) s^2(h-s+1)(w-s+1) + m^2hw \tag{4.6}$$

where γ is the ratio between the number of non-zero elements in the sparse matrix and the total number of elements in the original dense kernel. The decomposition overhead is small when (1) average of q_i is much smaller than γn and (2) m is much smaller than $\gamma n s^2$.

4.1.3 Learning Parameters

We adopt a two-phase strategy to learn the sparse kernels of Sparse Convolutional Neural Networks: initial decomposition and fine-tuning. The factorization in both Equation (4.2) and Equation (4.3) can be considered as classical matrix factorization problem if we reshape the convolutional kernels from a 4-D tensor to a 2-D matrix by collapsing the decomposition dimension. Therefore, it is a natural choice to use sparse matrix decomposition algorithm as initialization. However, there are a couple of concerns with regard to current sparse dictionary learning algorithms: First, the learning algorithms are not convex and therefore cannot guarantee to converge to a global optimal solution. Second, the learning process balances the sparsity of coefficients and the reconstruction error with a hyper-parameter, hence cannot provide exact reconstruction. Due to these concerns, we experiment with the following initial decomposition methods and compare their performance in Section 4.4.4:

- Decompose $\mathbf{K}$ and $\mathbf{R}$ using the sparse dictionary learning algorithm in [63], setting $\mathbf{P}$, $\mathcal{Q}_i$ as the learned dictionaries

- Decompose $\mathbf{K}$ and $\mathbf{R}$ using Principal Component Analysis (PCA), setting $\mathbf{P}$, $\mathcal{Q}_i$ as the principal components
- Initialize $\mathbf{P}$, $\mathcal{Q}_i$ as identity matrices, while keeping K and R unchanged. In this way, there is practically no initial decomposition and the sparsity is achieved solely by fine-tuning.

The initial decompositions with the above methods can only obtain moderate sparsity, but provide meaningful starting points with very low or no reconstruction error. Further fine-tuning is critical for learning highly sparse kernels. In the fine-tuning phase, we train the whole network with back-propagation while imposing sparsity constraints over the sparse kernels. The underlying reason of why fine-tuning can increase sparsity is that the original network is trained without any sparsity constraint. Since the network is known to be over-parameterized to avoid the sensitivity to initialization, it should have some freedom of modification without compromising the classification accuracy. The initialization phase can only obtain sparsity based on the reconstruction error, while the fine-tuning process can fully exploit the sparsity potential of our model by adopting the network loss as the direct error measurement. Our experiments justify this point by showing a significant increase in sparsity due to fine-tuning.

Formally, we minimize the following objective function in the fine-tuning phase

$$\begin{aligned} \underset{\mathbf{P},\mathcal{Q}_i,\mathcal{S}_i}{\text{minimize}} \quad & \mathbf{L}_{net} + \lambda_1 \sum_{i=1}^{m} \|\mathcal{S}_i\|_1 + \lambda_2 \sum_{i=1}^{m} \sum_{j=1}^{q_i} \|\mathcal{S}_i(j,\cdot)\|_2 \\ \text{s.t.} \quad & \|\mathbf{P}(\cdot,j)\|_2 \leq 1, j = 1,\ldots,m \\ & \|\mathcal{Q}_i(\cdot,\cdot,k)\|_2 \leq 1, i = 1,\ldots,m, k = 1,\ldots,q_i \end{aligned} \tag{4.7}$$

where $\mathbf{L}_{net}$ stands for the logistic loss function of the output layer of the network[51], and $\|\cdot\|_1$ and $\|\cdot\|_2$ denote the element-wise l_1 and l_2 norms of a matrix. The second term in (4.7) imposes lasso constraint over each of the elements of $\mathcal{S}_i$, and the third term treats each row of $\mathcal{S}_i$ as a group variable in a spirit of group-lasso formulation. The effective columns of each $\mathcal{Q}_i$ that needs to be

calculated is equal to the number of non-zero rows of corresponding $\mathcal{S}_i$. In this way, we can further reduce the overhead of convolving $\mathcal{Q}_i$ over the feature maps.

4.1.4 Comparison between our method and low-rank decomposition

Like [47] [18], our model uses a low-rank decomposition, but goes beyond previous work by encouraging sparsity in the weights used to express the filter to further reduce the redundancy in convolutional layers. We impose both sparse constraints and low-rank constraints with a combination of an l_1 norm and group lasso penalty.

Consider the decomposing matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ as the multiplication of $\mathbf{S} \in \mathbb{R}^{m \times n}$ and $\mathbf{P} \in \mathbb{R}^{n \times n}$. Using a sparse decomposition, the speedup is proportional to the percentage of non-zero elements in S, while the reduction in complexity using a decomposition is proportional to the percentage of non-zero columns. The sparsity constraint is able to target specific entries in S, while a low-rank constraint must eliminate entire columns. This makes it possible for our approach to target redundancy more precisely. As the results in Section 6.2 will show, this leads to remarkable performance improvement using the multiplication algorithm.

4.2 Sparse Matrix Multiplication Algorithm

While the sparsity penalties can target redundancy more precisely, the performance benefits can only be realized with a sparse matrix multiplication algorithm that does not incur overhead that overwhelms the benefit of the sparse matrix. In this section, we show how the multiplication can be implemented efficiently.

4.2.1 Motivation

When storing and manipulating matrices with high sparsity, to minimize the memory consumption, the sparse matrices are traditionally stored as the value of the non-zero elements and their indexes. This will inevitably lead to indirect and jumping memory access when multiplying two sparse matrices. Due to the pipeline and memory cache design of current CPU and GPU architecture, such jumping memory access is much slower than the direct and continuous access used in the dense case in practice. In addition, the irregular pattern of input matrices also makes it almost impossible to fully utilize the capacity of Single Instruction Multiple Data (SIMD) technology, which is essential to achieve high performance in compute-intensive algorithms like dense matrix multiplication.

To alleviate this problem, we introduce an efficient, sparse-dense matrix multiplication algorithm to improve the speed of our sparse convolutional kernels. The proposed method is based on the following two observations:

- Once the CNN model has been trained, the sparse convolutional kernels stay constant while the input feature maps vary with the input images. Therefore, the location of non-zero elements are fixed and pre-known, and they can be encoded directly into the code of matrix multiplication.
- Only the convolutional kernels are considered as sparse matrices. Although the input feature maps may possess moderate level of sparsity for some layers due to the effect of ReLU function, they will be treated as a dense matrix.

Our method is implemented on x86_64 CPU micro-architecture with the Advanced Vector Extension (AVX), which is an extension of SIMD and available on both Intel and AMD's CPUs after 2011. We expect that this approach could also be easily extended to GPU architectures. Our

method is implemented based on OpenBLAS [96], an open source linear algebra library mainly written in C and assembly. In the following sections, we will first briefly describe the traditional dense matrix multiplication algorithm in OpenBLAS, then introduce our method on multiplying sparse matrices.

4.2.2 Dense Matrix Multiplication in OpenBLAS

There are two main considerations on designing an efficient matrix multiplication algorithm: (a) Taking advantage of SIMD instructions for higher computing throughput; (b) Maximally utilizing the caches to reduce memory access latency. In the OpenBLAS library, matrix multiplication is implemented with the AVX instruction sets, in which 8 single precision floating point numbers can be stored in one 256bit AVX register and manipulated simultaneously. Namely, in one cycle of each CPU core, 8 pairs of single precision numbers can be multiplied and added simultaneously.

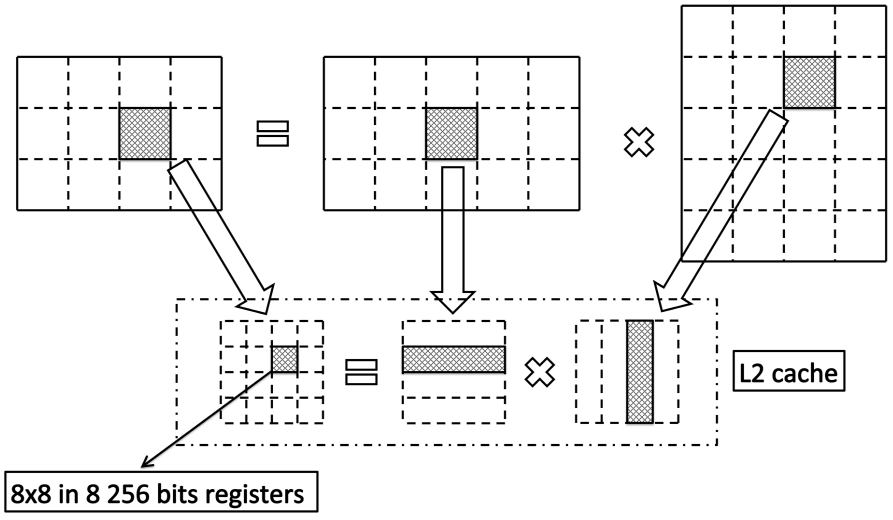


Figure 4.2: Matrix Multiplication Algorithm in OpenBLAS. The input matrices are first divided into blocks which can fit in L2 cache. Each block is then divided into 8 element wide strips. The multiplication outputs of two strips are held in 8 AVX registers during calculation.

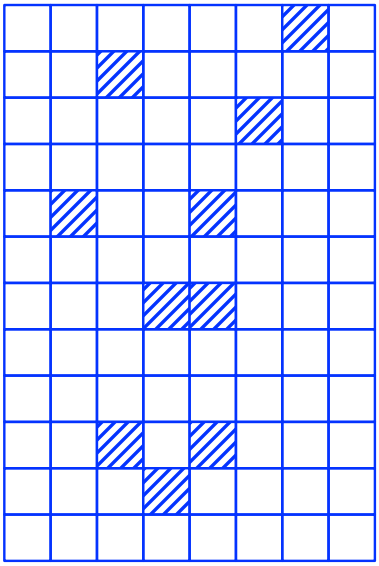
To reduce the memory access latency, the input matrices are divided into small blocks that are able to fit in the Level-2 cache of each CPU core. Then the blocks in the two input matrices are multiplied in the following way: The first block is divided into row strips that are 8-elements wide and the second block is divided to column strips that are also 8-elements wide. Then every two strips are multiplied into an 8×8 tiny square matrix that can be cached within 8 AVX registers. Figure 4.2 provides a graphical visualization of the matrix multiplication algorithm adopted in OpenBLAS.

4.2.3 Sparse Matrix Multiplication

Consider the sparse-dense matrix multiplication problem $\mathbf{C} = \mathbf{A} \times \mathbf{B}$. $\mathbf{A} \in \mathbb{R}^{m \times k}$ is a dense matrix and $\mathbf{B} \in \mathbb{R}^{k \times n}$ is a fixed and known sparse matrix. $\mathbf{A}$ and $\mathbf{B}$ are divided into blocks and strips in the same way as in OpenBLAS described above. Now, we consider the multiplication of one row strip and one column strip $\bar{\mathbf{C}} = \bar{\mathbf{A}} \times \bar{\mathbf{B}}, \bar{\mathbf{A}} \in \mathbb{R}^{8 \times k}, \bar{\mathbf{B}} \in \mathbb{R}^{k \times 8}, \bar{\mathbf{C}} \in \mathbb{R}^{8 \times 8}$. For any matrix $\mathbf{M}$, let $\mathbf{m}_{i,*}$ be the i^{th} row of $\mathbf{M}$ and $\mathbf{m}_{*,j}$ be the j^{th} column of $\mathbf{M}$. The matrix multiplication can be represented as

$$\bar{\mathbf{c}}_{*,j} = \sum_{i=1}^{k} \bar{\mathbf{a}}_{*,i} \bar{b}_{i,j}, 1 \leq j \leq 8 \tag{4.8}$$

In which every $\bar{\mathbf{c}}_{*,j}$ and $\bar{\mathbf{a}}_{*,i}$ are held in one AVX vector. Since $\bar{\mathbf{B}}$ is sparse in our case, we need to have the knowledge of the locations of non-zero elements and skip the zeros. To avoid indirect memory access, we propose to encode the structure of $\bar{\mathbf{B}}$ into the program. For each non-zero value $\bar{b}_{i,j}$, i indicates which $\bar{\mathbf{a}}_{*,i}$ to multiply with and j indicates which $\bar{\mathbf{c}}_{*,j}$ to save to. Since they both correspond to single AVX registers, we can simply encode i and j into our program as the index of registers. Figure 4.3 shows a toy example of how our method generates code from given sparse matrix.



(a) An example sparse matrix $\boldsymbol{B}$. The shadowed squares represent non-zero elements and the blank squares represent zero elements.

Input:

$\mathbf{A}$: 8×12 dense matrix

$\mathbf{B}$: 12×8 sparse matrix

Output:

$\mathbf{C} = \mathbf{A} \times \mathbf{B}$

Operations:

$\mathbf{c}_7 += \mathbf{a}_1 \times b_{1,7}$

$\mathbf{c}_3 += \mathbf{a}_2 \times b_{2,3}$

$\mathbf{c}_6 += \mathbf{a}_3 \times b_{3,6}$

$\mathbf{c}_2 += \mathbf{a}_5 \times b_{5,2}$

$\mathbf{c}_5 += \mathbf{a}_5 \times b_{5,5}$

$\mathbf{c}_4 += \mathbf{a}_7 \times b_{7,4}$

$\mathbf{c}_5 += \mathbf{a}_7 \times b_{7,5}$

$\mathbf{c}_3 += \mathbf{a}_{10} \times b_{10,3}$

$\mathbf{c}_5 += \mathbf{a}_{10} \times b_{10,5}$

$\mathbf{c}_4 += \mathbf{a}_{11} \times b_{11,4}$

(b) Generated Pseudo code for calculating $\boldsymbol{C} = \boldsymbol{A} \times \boldsymbol{B}$. $\boldsymbol{c}_i$ is the i^{th} column of $\boldsymbol{C}$ and $\boldsymbol{a}_j$ is the j^{th} column of $\boldsymbol{A}$. $b_{i,j}$ is the element of $\boldsymbol{B}$ at i^{th} column and j^{th} row

Figure 4.3: An example that illustrates how our algorithm generates code for multiplying a dense matrix and a sparse matrix

4.3 Application to Object Detection

We now apply SCNN to the object detection problem. Girshick et al. [30] first proposed to use CNN to solve the object detection problem. They warp each candidate window generated by the Selective Search[84] method to a fixed size and use CNN to generate high level discriminative features, with which linear SVM with hard negative mining is adopted to train object detection model. He et al. [40] significantly improves the speed of [30] by utilizing a Spatial Pyramid Pooling (SPP) scheme, in which the convolutional layers only need to be performed once (single scale) or several times (multiple scales) on the whole image and only the fully connected layers are

performed on each candidate window. Due to large amount of candidate windows in each image ($\sim$2000), the total running time of fully connected layers in [40] is comparable to or dominant over the one of convolutional layers.

We apply SCNN to accelerate the convolutional layers of SPP model. Although our method can accelerate the convolutional layers by a high factor, the total running time would not reduce much due to the relatively time consuming fully connected layers. In this section, we propose two schemes to reduce the complexity of fully connected layers to achieve an overall high efficiency.

First, we propose to apply a cascade scheme over the network in a similar fashion to [25]. Since the output of the last convolutional layer is already highly discriminative, directly applying a linear SVM classifier over it achieves good performance [30][40]. Therefore, we can use the last convolutional layer as the first stage of our cascade model to prune large portion of candidate windows, and then use the output of the final fully connected layer as a second stage classifier to generate the final detection results. The decision of detection threshold of first stage is a trade-off between efficiency and accuracy. A lower threshold retains high recall so that the overall accuracy is not affected while a higher threshold removes more candidates to achieve higher efficiency. In our case, we found that a threshold with a corresponding precision equals 0.05 is a balanced trade-off. Previous work on cascade models were applied only to single class detection. Multiple class detection, in general, achieve less gain, since the number of candidates rapidly increases with the number of classes. However in our model we can still remove considerably large portion of candidates with multiple classes.

Second, we decompose the fully connected layer in a similar fashion as we did with the convolutional layers. The operation of fully connected layer can be represented by matrix-vector multiplication followed by the neuron function. We decompose the weight matrix into the product of one sparse matrix and one dense matrix as $\mathbf{M} \approx \mathbf{PS}$, where $\mathbf{M} \in \mathbb{R}^{m \times n}$, $\mathbf{P} \in \mathbb{R}^{m \times k}$ and $\mathbf{S} \in \mathbb{R}^{k \times n}$.

We choose to enforce sparsity on $\mathbf{P}$ if $m > n$ and on $\mathbf{S}$ otherwise, so that the dense matrix has a smaller dimension. We impose both l_1 norm and group lasso cost function so that k is also reduced.

4.4 Experimental Results

4.4.1 Setup

We trained our model on the ImageNet LSVRC 2012 [14] dataset. We start from a pre-trained Caffe[49] reference CNN model, which is almost identical to the model described in [51]. The model consists of 5 convolutional layers and two fully connected layers, interlaced with subsampling layers, local normalizing layers, max pooling layers, rectified linear unit layers and dropout layers. The first convolutional layer has relatively large 11×11 kernels and only 3 input channels; the second convolutional layer has 5×5 kernels; The third, fourth and fifth convolutional layers have very small 3×3 kernels. The difference of kernel sizes as well as the number of input kernels affects the possible sparsity that can be achieved.

All 5 convolutional layers are optimized simultaneously according to Equation 4.7 using stochastic gradient decent with momentum. The base learning rate is initially set to 0.001, while sparsifying the network parameters. To stabilize the training process, we adopt a thresholding function that sets parameters smaller than $1e^{-4}$ to zero during training. Once the training process converges, we remove the sparsity constraint, but keep the thresholding function. Finally, we gradually decrease the base learning rate to fine-tune the network for best accuracy.

4.4.2 Results on ILSVRC12

Table 4.1 shows the results on ILSVRC12. For all 5 convolutional layers, we obtain more than 90% sparsity. The average number of bases in each layer is significantly smaller than s^2(square of kernel size), which corresponds to the full rank decomposition. The theoretical speedup are the ratio between the running time of our SCNN layer, and the original convolutional layer. The final column of Table 4.1 shows the actual acceleration factor achieved. Because of the overhead in sparse matrix multiplication, it is expected that actual performance improvements will not match theoretical results.

Table 4.1: Sparsity, Average number of bases ,theoretical and actual speedup corresponding to each convolutional layer for our SCNN model. q_i is the average number of bases in each channel. Results demonstrates that our highly sparse model could lead to remarkably acceleration for computation in both theory and practice.

layer	conv1	conv2	conv3	conv4	conv5
kernel size	11	5	3	3	3
input channels	3	96	256	384	384
output channels	96	256	384	384	256
complexity%	15.8	33.6	22.5	16.8	11.2
sparsity%	0.927	0.950	0.951	0.942	0.938
average q_i	29	7.91	5.23	4.32	3.95
theoretical speedup	2.61	7.14	16.12	12.42	10.77
Actual speedup	2.47	4.52	6.88	5.18	3.92

The speedup factor in conv1 is not as significant as other layers due to limited redundancy that is caused by large kernel size and the small number of input and output channels. The number of bases in conv1, although substantially reduced, still accounts for a large portion of running time because of small number of output channels. This issue is also present in previous low-rank approaches [47]. We experimented with decomposing the basis filters with combination of separable filters, but the bases showed too much variation to be expressed with separable filters.

4.4.3 Comparison with Only Using Low-Rank Approximations

To more directly compare with previous work, we also trained a modified version of the model used in [40], which is similar to that used in [18]. The primary difference lies in a spatial pyramid layer that will not affect the sparsity properties of the convolutional layers. Table 4.2 shows the comparison of theoretical speedup factors between [18] and our method trained based on [40]. We have a similar speedup as [18] in conv1 due to the above mentioned reason, while obtaining much higher speedup for conv2. It should be noted that [18] did not attempt to accelerate the later layers, likely due to the tiny size of filters (3×3), while our method is able to achieve significant speedup.

Table 4.2: Comparison between a model, similar to [40], trained with sparsity and the speedup factors reported in [18].

layer	conv1	conv2	conv3	conv4	conv5
kernel size	7	5	3	3	3
sparsity%	0.840	0.956	0.893	0.904	0.890
average q_i	21	9.06	6.76	6.86	6.98
theoretical speedup	2.62	7.06	8.03	8.78	7.29
low-rank [18]	2.4	2.5	-	-	-

4.4.4 Comparison of Initialization Methods

Figure 4.4 shows a comparison of the performance of different initialization methods that we adopted. Both PCA and sparse coding obtain more than 90% sparsity, while random initialization shows inferior sparsity by a large margin. Notably, all the methods obtain very limited sparsity with only initialization. These results demonstrate the importance of both initialization and fine-tuning. All of the methods show very small amount of accuracy drop during the fine-tuning process. The accuracy numbers shown in Figure 4.4 are generally lower than final accuracy since the learning rate we set during sparsifying the network is higher than final learning rate for faster convergence.

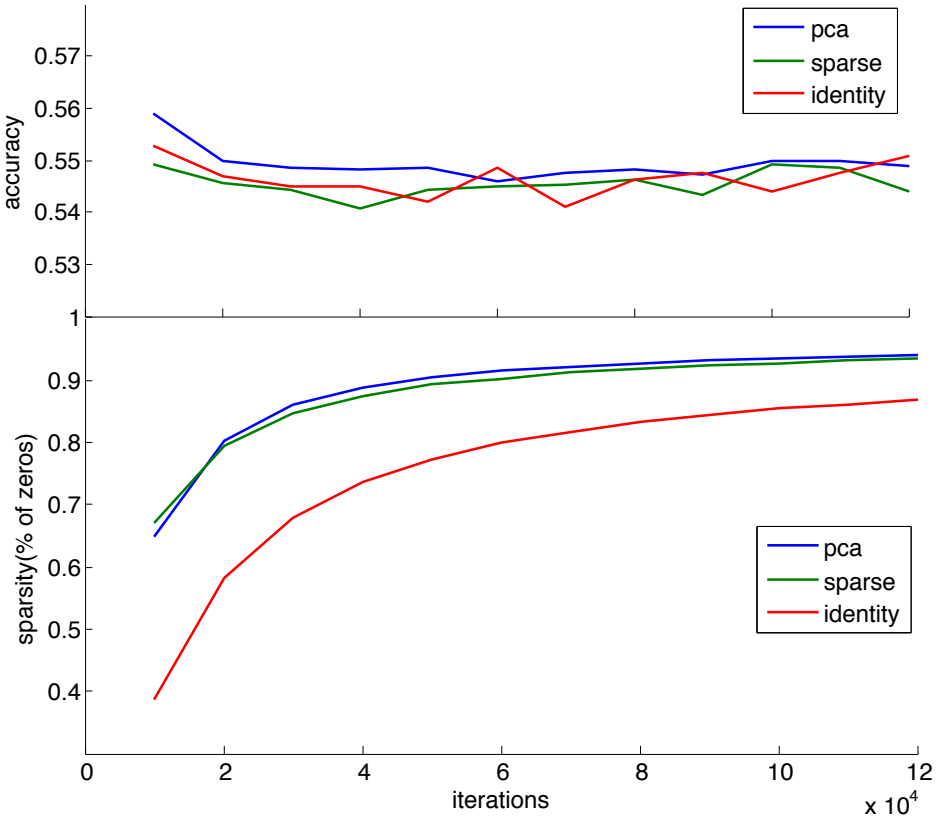


Figure 4.4: Comparison of initial decomposition methods. We show the variation of both the accuracy and the average sparsity of our sparse CNN during the training process.

Among all the three methods, the accuracy of PCA is slightly better than the others. The sparse coding method, although seems to make more sense theoretically, is inferior practically. We argue that the main reason is its non-convexness and non-accurate reconstruction. Although PCA is also a non-convex problem, global unique optimum can be obtained with SVD.

4.4.5 Bases Visualization

Figure 4.5 shows the average number of non-zero elements in our sparse convolution kernels corresponding to each basis of our decomposition over both input channels and kernels inside each channel. The bases are sorted according their eigenvalues of PCA initialization. High correlation

can be found between the sparseness and the eigenvalues of PCA, which justifies the importance of PCA initialization. For a significant portion of the bases, the sparse kernels are almost all zero. The all zero bases are equivalent to the ones that can be eliminated by low-rank decomposition. For relatively large-size kernels, like conv1 and conv2, the percentage of all zero bases is significant enough to achieve moderate level of speedup. However, for small-size kernels like conv3, conv4 and conv5, the percentage of all zero bases is very limited. Even a speedup factor of 2 will significantly sacrifice the accuracy with low-rank decomposition. The advantages of our method are clearly shown by the sparsity obtained for the non-zero bases, shown in Figure 4.5.

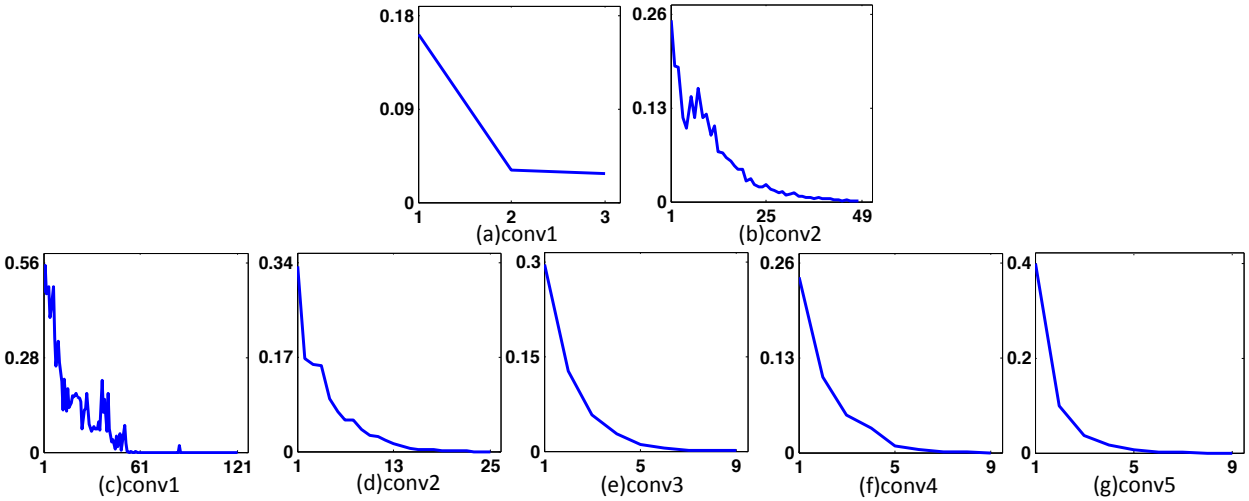


Figure 4.5: Average ratio of non-zero elements in our sparse convolution kernels corresponding to the bases of decompositions over channels and filters. (a) (b) show the bases over channels and (c) to (g) show the bases over filters.The bases in each figure are sorted in descending order of their eigenvalues in PCA initialization.

To justify the necessity of fine-tuning, we compare the convolution kernels in the original model and the ones that are reconstructed from our fine-tuned sparse model in Figure 4.6. We only show the kernels of conv1, conv2 and conv3 layers due to limited space. We ignore the kernels that are all zeros, which is very common from conv3 to conv5. We also measure the average similarity between the original and reconstructed kernels by first deduct the mean values from both kernels, and then calculate the cosine similarity measurement, which is defined as $\text{Sim}_{\cos}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$. From

Figure 4.6, we can see that the average similarity decreases rapidly from conv1 to conv3. The kernels in conv3 look very different from the original ones. Considering how little the accuracy drops in our model, this justifies our argument that the original network is extremely over-parameterized and significant regularization can be imposed without affecting the performance. Thus, one can hardly exploit the full sparsity potential of the network by only attempting to approximate the original kernel instead of network loss based fine-tuning adopted in our method.

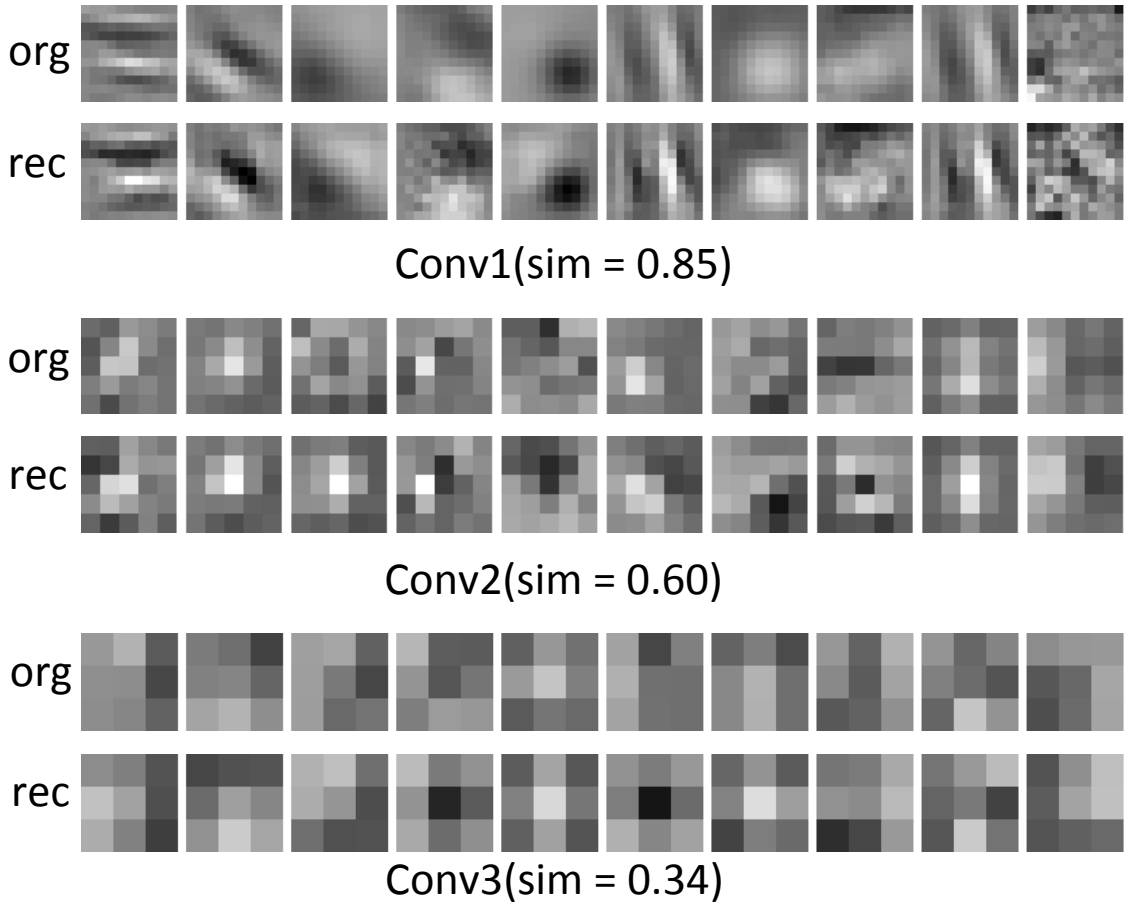


Figure 4.6: Comparison between the original convolution kernels and the convolution kernels reconstructed from our sparse kernels. Here we show randomly sampled kernels conv1, conv2 and conv3 layers. conv4 and conv5 are very similar to conv3. For each layer, the first row shows the original kernels and the second row shows the reconstructed ones. The average cosine similarity between them are displayed under.

4.4.6 *Evaluation of Sparse Matrix Multiplication Algorithm*

We first analyze the performance of our sparse dense matrix multiplication algorithm with a randomly generated matrix. We randomly generate one 1024×1024 dense matrix and one sparse 1024×1024 matrix, and measure the running time of multiplying them with our algorithm. Our experiments are performed on an Intel i7-3930k CPU. We measure the single-thread performance in this case for simplicity. The multi-thread performance should be consistent. Figure 4.7 shows the result of our evaluation.

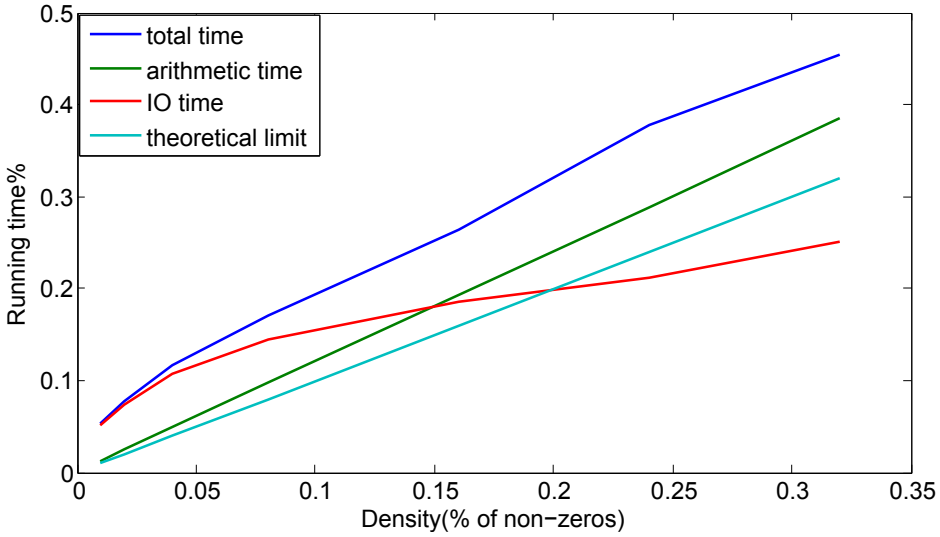


Figure 4.7: Running time analysis of our sparse-dense matrix multiplication algorithm. The horizontal axis stands for the percentage of non-zero elements in the input sparse matrix, and the vertical axis is the relative running time comparing to the dense matrix multiplication code in OpenBLAS. The arithmetic time is the running time of multiplication and addition, and the I/O time includes loading the input matrix from memory to cache, loading from cache to CPU and writing result to memory. The theoretical time is the best possible speedup one can achieve, which is identical to the density of the input matrix.

The following conclusions can be drawn from this figure: (a)The arithmetic time is strictly proportional to the density of the input matrix, and very close to the theoretical limit; (b) The I/O time decreases with the increase in sparsity, but in a sub-linear speed. The time of loading from memory to cache and storing results are constant regardless of the variation of sparsity, and the time of loading the dense matrix from cache to CPU depends on the percentage of consecutive 8

zeros in the sparse matrix; (c) The I/O time is increasingly dominant when the sparsity increases, when the density is less than 10%, the I/O operations takes more than 80% of the total running time. (d) Note that the sum of arithmetic time and I/O time is significantly higher than the total time. This is due to the parallelism introduced by the pipeline strategy of CPU. For this reason, the arithmetic operations and I/Os can be executed simultaneously, thus providing a significantly better efficiency.

4.4.7 Running Time Analysis

Table 4.3 shows the actual speed of our code as well as the proportion of each component. Significant speedups are achieved for all 5 layers. The sparse matrix multiplication time is dominant for the last three layers while the basis convolution time takes a large portion for the first two layers, which is consistent with theoretical analysis. The gap between actual speedup and theoretical one comes mainly from two factors: (1) Overhead of sparse matrix multiplication; (2) Low efficiency of basis convolution. Caffe implements convolution as matrix multiplication, which is relatively inefficient for small number of filters as in our case. We implement a faster version but is still not fully optimized. In addition, joint cache optimization with both convolution and sparse matrix multiplication will further improve the efficiency.

Table 4.3: Running time analysis and comparison with original dense networks. All the numbers for each layer are normalized with the layer's total running time with our method. The last row is the speed-up factor of our method.

layer	conv1	conv2	conv3	conv4	conv5
Channel Decomp	0.06	0.14	-	-	-
Basis Convolution	0.78	0.41	0.21	0.30	0.44
Matrix Mult	0.16	0.45	0.79	0.70	0.56
Original	2.47	4.52	6.88	5.18	3.92

4.4.8 Results on Object Detection

We used the fine-tuned model in Table 4.2 to perform object detection on PASCAL VOC2007 [22] dataset. The accuracy of our method compared with the original one in [40] is shown in Table 4.4. We are not able to reproduce the accuracy using the code published by [40], so we put the number we get instead. Our method is approximately 2% worse than the original SPP model, while obtaining several times faster speed. We speculate that the higher accuracy drop than classification problem is probably due to the fact that the convolutional layer is trained on ILSVRC dataset, while only the fully connected layer are fine-tuned to adapt to the PASCAL data as in [40]. Thus, although we are sparsifying the convolutional layers with fine-tuning, the loss of the network is not equivalent to the loss of the detection problem. A whole network based fine-tuning should further reduce the accuracy drop of our method.

Table 4.4: Mean average precision of object detection with our method compared with [40]. "bb" stands for bounding box regression, "1s" means 1 scale and "5s" means 5 scales. Our sparse model is inferior to [40] by approximately 2%.

	fc7(1s)	fc7bb(1s)	fc7(5s)	fc7bb(5s)
spp[40]	52.47	54.19	54.75	57.19
ours	50.16	52.64	52.58	55.13

In our cascade model, we set the thresholds of first stage so that the precision for each class equals 0.05. Approximately 80% of candidate windows are pruned for each image, thus bringing approximately 5$\times$ speedup of fully connected layers with almost no drop in accuracy. In addition, 85% and 68% sparsity are achieved for the first and second fully connected layer, further providing over 2$\times$ speedup. The running time of fully connected layer is thus reduced to be much smaller than the convolutional layers.

4.5 Summary

In this chapter, we propose to apply sparse decomposition to the convolutional layers to improve their efficiency. We apply both inter-channel and inner channel decomposition to fully exploit the redundancy underlying the convolutional kernels. We learn the sparsity with an initial decomposition and further fine-tuning. To convert the obtained sparsity to actual lower runtime, we proposed a CPU based sparse matrix multiplication algorithm that takes advantage of the fact that the sparse kernels are known before the actually computation. In addition, we also apply our method to the object detection framework and achieve similar speedups.

CHAPTER 5: A MULTI-STAGE CONVOLUTIONAL NEURAL NETWORK FRAMEWORK

5.1 Review of Cascade and Label Tree Models

5.1.1 Cascade Models

Cascade models are commonly used to improve the efficiency of object detection algorithms. Based on the fact that the negative samples are statistically dominant and most of them are relatively easy to distinguish from the positive examples, a cascade of classifiers with increasing complexity and accuracy are sequentially performed on the test samples. At each stage of cascade, a fraction of the negative samples are pruned. Algorithm. 3 shows how the cascade model works for a test image. The underlying idea of the cascade model is to divide the feature space and process samples in different areas with different models.

Algorithm 3 Cascade Model

Require: Test example $\boldsymbol{x}$, cascade of binary classifiers and corresponding thresholds $(C_i, t_i), i = 1, 2, ..., n$

```
label x as positive;
for i = 1 to n do
    if C_i(x) < t_i then
        label x as negative;
        break;
    end if
end for
```

5.1.2 Label Tree Model

The label tree model was proposed by [7] for efficient large scale classification. A label tree is a tree $T = (V, E)$, which has a set of nodes V and edges E. Each node is associated with a cluster of labels. The root node contains all the labels, and each node's labels are a subset of its parent's labels. The leaf nodes have only one label. Each non-leaf node has a decision function that assigns the input sample to one of its children. The test image is categorized by traversing the tree from the root, as is described in Algorithm. 4.

Algorithm 4 Label Tree Algorithm

Require: Test example $\boldsymbol{x}$, label tree parameters:
$\sigma(s)$: children nodes of s
$l(s)$: labels of s
$\boldsymbol{w}_s$: classifier weights associated with s

1: Initialize pointer s to the root node
2: **while** $\sigma(s) \neq \emptyset$ **do**
3: $s \leftarrow \underset{c \in \sigma(s)}{\operatorname{argmax}} \boldsymbol{w}_c^\top \boldsymbol{x}$
4: **end while**
5: Assign the label $l(s)$ to the test example

To optimize the accuracy of the label tree model, the labels in the same node are more confusing than the ones in different nodes. The label tree model reduces the complexity by recursively dividing the label space into small label clusters within which recognition can be performed efficiently.

5.2 Proposed Method

5.2.1 Challenges

In this chapter, we build a multi-stage framework by applying the ideas of cascade and the label tree model to the Convolutional Neural Networks. Each test image is processed differently based

on its feature property. One important assumption for the cascade model to achieve significant speedup is that there is a large portion of "easy" negative samples that can be efficiently eliminated with a simple model. Cascade model for image classification is naturally more difficult than object detection since we need to provide an accurate categorization for each test image. Despite this, we argue that there exists high diversity in the level of "difficulty" of images, and therefore using recognition models with different level of complexity can lead to higher efficiency.

In previous work [7][15][59], the label tree method has been exclusively applied to linear classifiers, of which the complexity is proportional to the number of categories. While this assumption is necessary to achieve logarithmic complexity with the label tree model, it cannot be made when we use the CNN models as classifier. In CNN, the convolutional layers behave as feature extractors and the final logistic regression layer behaves as a linear classifier. Since most of the computation is consumed by the convolutional layers, the complexity of the whole CNN model is sub-linear with the number of categories. In other words, a network that classifies 100 classes will not be 10 times faster than a network that classifies 1000 classes with similar level of accuracy. Due to this reason, we need to carefully design our framework to achieve optimal efficiency. For a typical CNN model, most of the running time is spent on the convolutional layers rather than the classifier layer. Our hypothesis is that, classifying smaller number of classes would require less number of convolutional filters than classifying large number of classes with similar level of accuracy. On the other hand, the dominance of the convolutional layers in complexity leaves us more freedom to design a more complicated tree branching model based on the extracted feature from the CNN models, without worrying about weakening the overall efficiency.

5.2.2 *Intuition: Cascading label clusters*

In this section, we briefly introduce the intuition behind our idea. Consider a two-level label tree framework, in which each node is associated with a CNN model. The model in the root node assigns the input to one of its child nodes, each of which contains a subset of labels. Since the root node model cannot perfectly distinguish all categories, the categories that are more confusing with each other tend to be assigned to the same child node. There exists a trade-off between the accuracy of the root node model assigning samples to its children node, and the overlapping of labels between the children nodes, as is described in [15]. To maintain a high accuracy, the level of label overlapping needs to remain at a high level, which leads to large number of labels for each child node, hence increasing the complexity of CNN models in the children nodes.

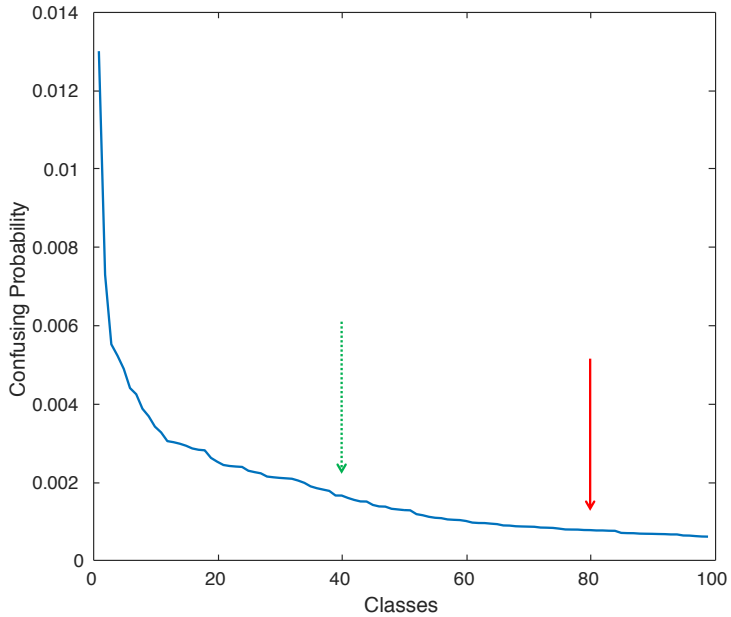


Figure 5.1: The probability of one class confusing with other classes. The shown class is relatively heavily confused with a few classes, while slightly confused with a large number of classes. This "long tail" property of confusion leads us to the idea of cascading label tree model.

To alleviate this problem, we consider cascading two levels of children nodes that have different level of label overlapping. Figure. 5.1 shows the probability of one typical class confusing with

other classes sorted in a descending order. This curve shows a "long tail" property. There is a relatively high probability for this class to confuse with a small number of classes, while low probability to confuse with a large number of classes. We attempt to build a two-stage cascade of label clusters. In the first stage, the child node that includes the chosen class also includes all the classes in the figure until the green arrow, while in the second stage it contains the classes until the red arrow. The first stage, despite its lower overall accuracy, can correctly process a large portion of input images due to the long tail property of the curve. Due to less number of clusters in each node, the first stage also achieves higher efficiency than the second stage. If we design a "stage predictor" that assign the "easy" images to the first stage, and "hard" images to the second stage, then we obtain a model that can be as accurate as the second stage but with higher average efficiency.

5.2.3 Multi-stage framework

We propose a three-stage framework, as is illustrated in Figure. 5.2. Both accuracy and complexity of the stages increase monotonically. While the first stage is an efficient CNN model, the second and third stages are composed of a two-level label tree model, and each node of the tree is associated with a CNN model. A stage predictor, which is based on the output of the first stage, selects which stage should be chosen to predict the label of the input image. In the following sections, we will explain each stage in detail.

5.2.3.1 First Stage

The first stage of our framework consists of an efficient CNN model with relatively low accuracy. This stage processes **every** test image and generates the following two outputs:

- $P(y_i|\boldsymbol{x}, s_1)$: Probability of each output label using this model
- $f_{s1}(\boldsymbol{x})$: Feature vector that will be used by the following stages

5.2.3.2 Stage Predictor

Given $f_{s1}(\boldsymbol{x})$ for each image, the stage predictor determines whether directly using the label probability estimation of the 1st stage $P(y_i|\boldsymbol{x}, s_1)$ as output, or assigning the image to the 2nd stage or the 3rd stage for more accurate categorization.

The stage predictor first predicts whether to directly use the 1st stage as the output, then predict whether to use the 2nd stage or the 3rd stage, as is modeled probabilistically in the following equation:

$$\begin{aligned} p(y|\boldsymbol{x}) = \; & p(y|s_1, \boldsymbol{x})p(s_1|\boldsymbol{x}) + \\ & p(y|s_2, \boldsymbol{x})p(s_2|\boldsymbol{x})(1 - p(s_1|\boldsymbol{x})) + \\ & p(y|s_3, \boldsymbol{x})p(s_3|\boldsymbol{x})(1 - p(s_1|\boldsymbol{x})) \end{aligned} \tag{5.1}$$

Both $p(s_1|\boldsymbol{x})$ and $p(s_2|\boldsymbol{x})$ are modeled as a binary logistic regression model, whose input feature vector is transformed from $f_{s1}(\boldsymbol{x})$, as will be discussed in Sec. 5.3.3.

5.2.3.3 Second Stage

The 2nd stage is composed of a two-level label tree model. A linear model that takes $f_{s1}(x)$ as feature assigns the test image to one of the children nodes. Each child node is associated with one

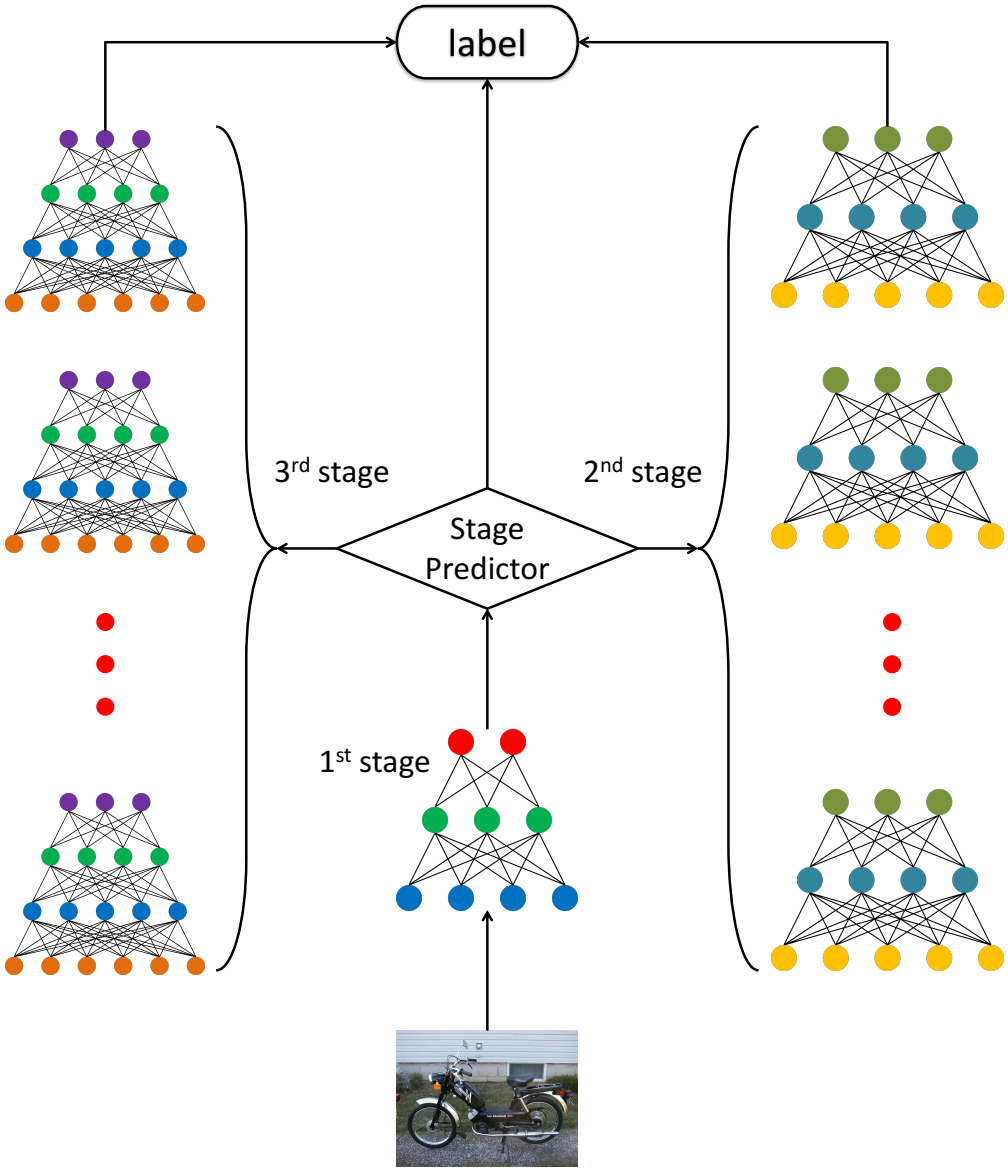


Figure 5.2: Flow chart of our multi-stage framework. All the input images are processed by the first stage. The stage predictor determines whether to directly output label predicted by first stage, or assign the image to the label tree model in 2nd or 3rd stage for more accurate prediction. The 3rd stage label tree model has heavier label overlapping, and is more accurate and complex than 2nd stage.

CNN model that assigns a label to the image in the node. If one test image is assigned to this stage, then its probability of each label is calculated with the following equation:

$$p(y_i|\boldsymbol{x}, s_2) = p(y_i|\boldsymbol{x}, c_j), \;\; j = \arg\max_{c_k \in s_2} p(c_k|f_{s1}(x)) \tag{5.2}$$

where c_j is the jth cluster of the label tree model.

The overall accuracy of the 2nd stage is better than the first stage, with relatively higher complexity.

5.2.3.4 Third Stage

The 3rd stage is also a label tree model similar to the second stage, except that it has different number of clusters and each cluster has different number of classes. There are higher level of label overlapping between children nodes, so that the children nodes have more labels on average, and the probability of assigning a test image to a wrong child node is lower. The images that are not assigned to the first two stages are assigned to this stage. The probability of each label is calculated as follows:

$$p(y_i|\boldsymbol{x}, s_3) = p(y_i|\boldsymbol{x}, c_j), \;\; j = \arg\max_{c_k \in s_3} p(c_k|f_{s1}(x)) \tag{5.3}$$

Due to large number of classes for each cluster, the complexity of the CNN models in this stage is higher than previous 2 stages.

5.2.4 Learning Label Tree Clusters

The label tree clusters are learned with the probabilistic formulation proposed in Chapter. 3. Within the multi-stage framework, we need to accurately control the balance between accuracy and the

Algorithm 5 Proposed Framework

Require: Test example $\boldsymbol{x}$, first stage model S_1, second stage model S_2, third stage model S_3

1: Calculate probability of x belonging to each label with S_1, $P(y_i|x, S_1)$
2: Calculate $P(c_i|x, s_2)$, $P(c_i|x, s_3)$
3: Calculate $P(s_1|x)$, $P(s_2|x)$, $P(s_3|x)$
4: **while** $\sigma(s) \neq \emptyset$ **do**
5: $\quad s \leftarrow \arg\max_{c \in \sigma(s)} \boldsymbol{w}_c^\top \boldsymbol{x}$
6: **end while**
7: Assign the label $l(s)$ to the test example

number of classes for each cluster in the 2nd and 3rd stage. Instead of using the trivial methods of determining the number of leaf node based on the categorical distribution described in Section. 3.5.2, we apply the linear programming algorithm proposed in [15] to obtain a hard label partition according to an desired accuracy. We'll formally describe this method in this section.

Given the input feature vectors and their corresponding labels $\{\boldsymbol{x}_i, y_i\}, i = 1, 2, ...n$, the number of classes K, and the number of clusters Q, the probability of assigning label y to $\boldsymbol{x}$ is defined as

$$p(y|\boldsymbol{x}) = \sum_{k=1}^{K} p(y|C_k, \boldsymbol{x}) p(C_k|\boldsymbol{x}) \tag{5.4}$$

Where $p(C_k|\boldsymbol{x})$ denotes the probability of assigning sample $\boldsymbol{x}$ to cluster k, and $p(y|C_k, \boldsymbol{x})$ denotes the probability of cluster k assigning label y to $\boldsymbol{x}$. $p(y|C_k, \boldsymbol{x})$ is defined as a multinomial distribution, and $p(C_k|\boldsymbol{x})$ is defined as a multi-class logistic regression model:

$$p(C_k|\boldsymbol{x}) = \frac{exp(\boldsymbol{w}_k^T \boldsymbol{x})}{\sum_{i=1}^{K} exp(\boldsymbol{w}_i^T \boldsymbol{x})} \tag{5.5}$$

We optimize $p(y|C_k, \boldsymbol{x})$ and $p(C_k|\boldsymbol{x})$ in an iterative fashion. When $p(y|C_k, \boldsymbol{x})$ is fixed, optimizing $p(C_k|\boldsymbol{x})$ is equivalent to learning a weighted logistic regression model. When $p(C_k|\boldsymbol{x})$ is fixed, we update $p(y|C_k, \boldsymbol{x})$ with the following method.

We define a binary partition matrix $P \in \{0, 1\}^{Q \times K}$. $P_{qk} = 1$ if class k belongs to cluster q. Note

that the clusters can be overlapping, namely, there can be multiple 1s on each column of P. The overlapping reduces the system's chance of assigning to wrong clusters. For each sample $(\boldsymbol{x}_i, y_i)$, q_i is the cluster assigned by the classifier by maximum likelihood, and the following optimization problem is solved:

$$\begin{aligned} \underset{w,P}{\text{minimize}} \ & \frac{1}{m}\sum_{i=1}^{m} A(q_i, P) \\ \text{subject to} \ & \frac{1}{m}\sum_{i=1}^{m} L(q_i, y_i, P) \leq \epsilon \\ & P \in \{0,1\}^{Q\times K} \end{aligned} \tag{5.6}$$

where

$$\begin{aligned} A(q_i, P) &= \frac{1}{K}\sum_{k=1}^{K} P(q_i, k) \\ L(q_i, y_i, P) &= 1 - P(q_i, y_i) \\ q_i &= \underset{j}{\text{argmax}} \ \boldsymbol{w}_j^T \boldsymbol{x}_i \end{aligned} \tag{5.7}$$

$A(q_i, P)$ is defined as the ambiguity term, which represents the average number of classes in each cluster, and $L(q_i, y_i, P)$ stands for the error of the assigning the images to clusters. When q_i is fixed, optimal P can be approximately found with linear programming according to [15].

Given P , $p(y|C_k, \boldsymbol{x})$ is updated with the following equation:

$$p(y|C_k, \boldsymbol{x}) = \frac{\sum_{i=1}^{N} \boldsymbol{I}(q_i = k)\boldsymbol{I}(y_i = y)\boldsymbol{I}(P(k, y_i) = 1)}{\sum_{i=1}^{N} \boldsymbol{I}(q_i = k)\boldsymbol{I}(P(k, y_i) = 1)} \tag{5.8}$$

Where $\boldsymbol{I}(\cdot)$ is a binary indicator function.

5.2.5 Learning Stage Predictor

The objective of stage predictor is to assign each image to the right stage, so that the overall accuracy is maintained, with minimal amount of **average** computation. While the complexity of each stage is fixed, the average computation is determined by the ratio of selecting stages with lower complexity. Avoiding the more complex stages will lead to less overall computation. Therefore, we learn the stage predictor by maximize the log likelihood of ground truth label of input image, while also maximizing the average chance of selecting the first 2 stages, as defined in the following equation:

$$L = -\sum_{i=1}^{N} (\log p(y|\boldsymbol{x}) + \alpha p(s_1|\boldsymbol{x}) + \beta p(s_2|\boldsymbol{x})) \tag{5.9}$$

Where α and β are positive hyper-parameters that control the ratio of selecting 1st and 2nd stage. Both $p(s_1|\boldsymbol{x})$ and $p(s_2|\boldsymbol{x})$ are represented as binary logistic regression models.

5.3 Experiments

We evaluate our multi-stage framework on the ImageNet LSVRC 2012 dataset [14]. The dataset includes 1000 categories, with 1.2M training images, 50K validation images, and 100K test im-

ages. We use Caffe[49] to train the CNN models in our framework. In this work, we focus on optimizing the top-1 test accuracy for a single center crop, since efficiency is our main objective.

Table 5.1: Configuration of the CNN models used in our framework. They vary in input size, number of convolutional layers, resulting in diverse complexity and accuracy.

Stage	Input Size	# Conv Layers	# Multiplications	Accuracy
1st	150×150	7	148M	53.2%
2nd	221×221	13	450M	59.5%
3rd	221×221	13	769M	62.2%

The structure of our CNN models is designed by modifying from [37]. We split the convolutional filters into two groups as in [50], and change the input image size as well as the number of filters and layers to make a balance between accuracy and complexity for each stage of our framework. For similarity and universality of model training, we follow the standard training process of Caffe, omitting some techniques adopted by [37], such as SPP [40] and different image cropping method [42]. Table. 5.1 briefly summarizes the complexity of our CNN models and their accuracy on ILSVRC 2012 dataset. Table. 5.2 and Table. 5.3 list their detailed configurations.

5.3.1 Evaluation of Learning Label Tree

We first evaluate the performance of label tree learning algorithm based on the feature $f_{s1}(\boldsymbol{x})$ generated by the first stage CNN model. We experiment with different form of $f_{s1}(\boldsymbol{x})$, and come to the conclusion that setting $f_{s1}(\boldsymbol{x}) = P(y_i|\boldsymbol{x}, s_1)$, namely using the category probabilities as the feature, achieves the best performance due to its outstanding generalization ability. The category probabilities provide direct information about the confusing potential between categories, which is the essential information needed for generating label tree.

We compare the relationship between ambiguity and accuracy, as defined in equation. 5.7, in the

Table 5.2: Configuration of 1st stage model

type	kernel	stride	pad	group	output
input					3×150^2
conv	7	2	0	1	64×72^2
max pooling	3	3	0	0	64×24^2
conv	2	1	0	2	128×23^2
			1		128×24^2
max pooling	2	2	0	0	128×12^2
conv	2	1	0	2	256×11^2
			1		256×12^2
max pooling	2	2	0	0	256×6^2
conv	2	1	0	2	512×5^2
			1		512×6^2
	1	1	0	1	1024×6^2
average pooling	6	6	0	0	1024
linear	-	-	-	-	1000

setting of different number of clusters, as shown in Figure. 5.3. The ambiguity increases rapidly with the accuracy. When the accuracy is set to 70%, the ambiguity is approximately $\frac{\#\ classes}{\#\ clusters}$, which means there is almost no overlapping between clusters. Whereas when we increase the accuracy threshold, the increase in ambiguity is not quite dependent on the number of clusters, but more related to the level of confusion like in Figure. 5.1.

Based on Figure. 5.3, we select the 64 clusters model with accuracy equals 75% as our second stage model, and the 32 clusters model with accuracy equals 90% as our third stage model.

5.3.2 Performance of CNN Models in 2nd and 3rd Stages

In the 2nd stage and 3rd stage, the structure of the CNN model for each cluster is the same. We first train the CNN model using the whole 1K categories ILSVRC 2012 dataset, then fine-tune the model for each cluster using the images whose labels are in the cluster. To further improve the

Table 5.3: Configuration of 2nd and 3rd stage models.

type	kernel	stride	pad	group	output 2nd(3rd) stage
input					3×221^2
conv	7	2	0	1	64×108^2
max pooling	3	3	0	0	64×36^2
conv	2	1	0	2	$128(180) \times 35^2$
			1		$128(180) \times 36^2$
			0		$128(180) \times 35^2$
			1		$128(180) \times 36^2$
max pooling	2	2	0	0	128×18^2
conv	2	1	0	2	$256(360) \times 17^2$
			1		$256(360) \times 18^2$
			0		$256(360) \times 17^2$
			1		$256(360) \times 18^2$
max pooling	2	2	0	0	256×6^2
conv	1	1	0	1	$512(720) \times 6^2$
	2		0	2	$512(720) \times 5^2$
	2		1	2	$512(720) \times 6^2$
	1		0	1	$1024(1440) \times 6^2$
average pooling	6	6	0	0	$1024(1440)$
linear	-	-	-	-	1000

accuracy, we adopt the distilling method proposed recently in [41]. We generate the probability output of all the training images from the latest CNN model in [38] with temperature equal to 2. Then for each cluster, we extract the probabilities of labels that are inside this cluster, and fine-tune the CNN model by minimizing the cross entropy between the probability of source model and target model.

In this section, we verify our assumption that with the same network structure, training one model for each cluster can achieve more accurate recognition than training one single model for all the classes. For each image in the validation set, we first manually assign it to the cluster that contains its ground truth label and has minimum number of classes, namely, we assume our first stage makes

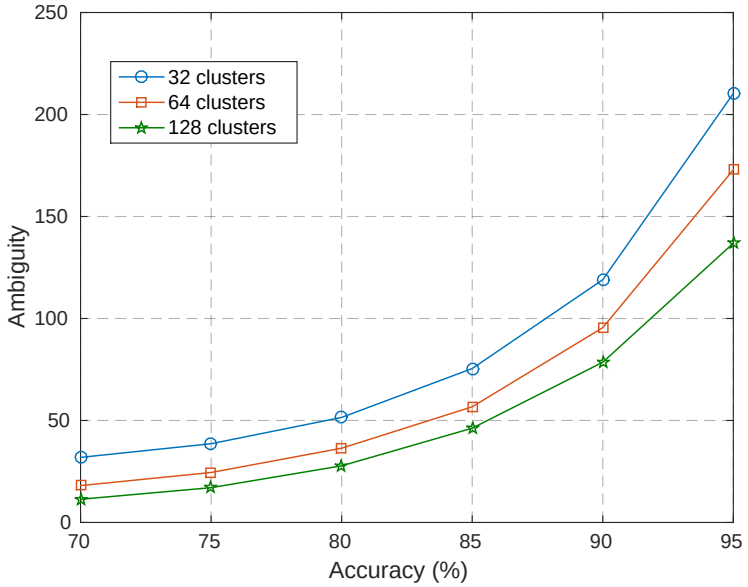
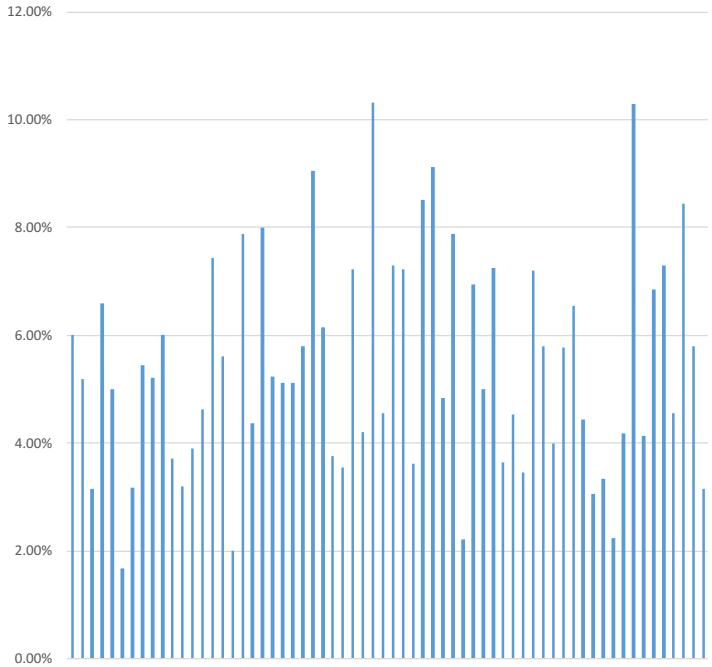


Figure 5.3: Relationship between the ambiguity and accuracy for label clustering algorithm with different number of clusters

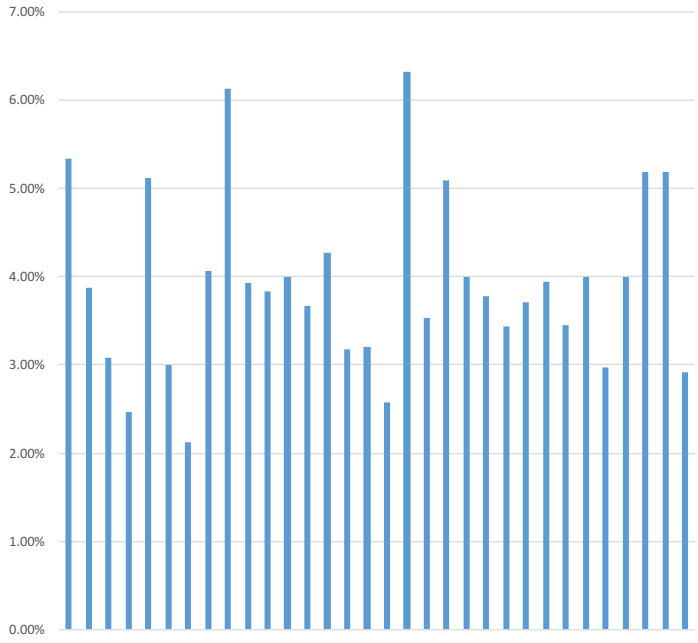
no mistake. Then we predict its label using either the models that are fine-tuned for each cluster, or the single model that is trained using the whole dataset, and compare their performance. In this way, we guarantee that they are compared in a fair condition and the result is not affected by the performance of our first stage model. Figure. 5.4 shows the improvement over the single model of the 2nd stage and the 3rd stage. For both 2nd stage and 3rd stage, all of the clusters demonstrate significant improvement by fine-tuning on each cluster. For some clusters, the improvement is more than 10%. The average improvement of 2nd stage is higher than the 3rd stage, owing to smaller number of classes for each cluster.

5.3.3 Evaluation of Stage Predictor

We train $p(s_1|\boldsymbol{x})$ and $p(s_2|\bar{s_1}, \boldsymbol{x})$ in equation 5.1 in a sequential fashion. First, we set $p(s_2|\bar{s_1}, \boldsymbol{x}) = 0$, assuming no image is assigned to the 2nd stage, and train $p(s_1|\boldsymbol{x})$; Then we fix $p(s_1|\boldsymbol{x})$, and train $p(s_2|\bar{s_1}, \boldsymbol{x})$. The reason why we utilize a two-step training strategy rather than training both



(a) 2nd stage model



(b) 3rd stage model

Figure 5.4: Comparison of accuracies between fine-tuning separate CNN models for each cluster and using one single CNN model for all the categories, assuming that the 1st stage is optimal. For both 2nd and 3rd stage, all the clusters show significant improvement.

predictors jointly, is that the two stage predictors require different type of features to achieve optimal performance. We experimented with various types of features, and found that the following feature generalizes best: For learning $p(s_1|\boldsymbol{x})$, we sort the category probability estimation of the 1st stage model in a descending order, and take the top 4 probability values as the feature vector; For learning $p(s_2|\bar{s_1}, \boldsymbol{x})$ we sort the probability of assigning the input image to each cluster in the 2nd stage in descending order, and take the top 8 probability values as feature vector.

By tuning the parameters α and β in Equation. 5.9, we obtain various ratios of using the 1st and the 2nd stage model, and their corresponding overall accuracies, which are shown in Figure. 5.5. The accuracies decrease very slowly when the ratios of 1st stage and 2nd stage are approximately less than 35%, and then decrease faster afterwards. This verifies that a significant portion of images can be processes with only the first two stages almost without hurting the accuracy.

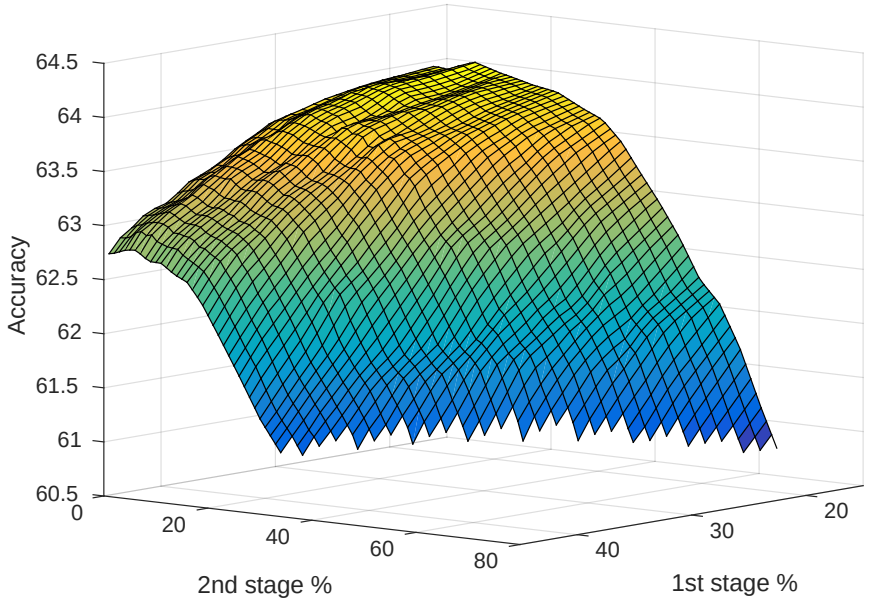


Figure 5.5: Overall accuracy as a function of the ratio of selecting 1st stage model and 2nd stage model by the stage predictor. When the ratios of 1st stage and 2nd stage are approximately less than 35%, the accuracy decreases very slowly.

5.3.4 *Comparison with Other CNN Models*

Finally, we compare the overall performance of our multi-stage framework with other CNN models as well as the models we used in our framework. We analyze the tradeoff between accuracy and complexity for each model, as is shown in Figure. 5.6. For our framework, we obtain a series of models by analyzing Figure. 5.5 and taking the models with best accuracy complexity ratio, as is shown by the blue curve. The orange triangle and the yellow squares represent our 2nd stage and 3rd stage CNN models that are trained and tested as a single model using the whole ILSVRC dataset. Due to our careful design, they achieve better accuracy and efficiency than the Caffe reference model and Zeiler & Fergus model. Comparing to the 3rd stage model, our multi-stage framework is 2% better with less complexity, or 60% more efficient with similar accuracy.

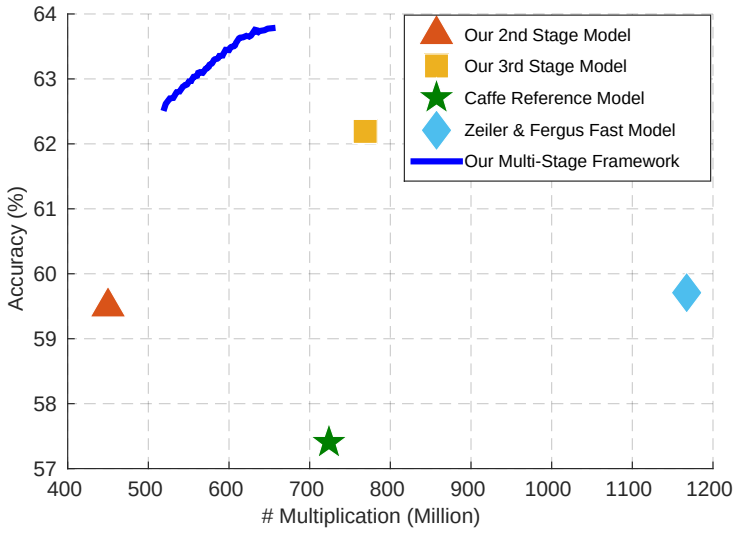


Figure 5.6: Comparison of our framework with other classical CNN models and our 2nd and 3rd stage model. Our framework achieves significant improvement on accuracy/complexity ratio over single CNN models.

5.4 Summary

In this chapter, we proposed a multi-stage CNN framework that consumes less average computation than single CNN model with comparable accuracy. With a combination of label tree and cascade model, we divide both the sample space and the label space, so that each image is processed with least amount of computation. Experiments on ImageNet dataset demonstrate that our methods achieve better accuracy/complexity ratio than popular CNN models.

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 Summary of Contributions

This dissertation presents several methods of improving the efficiency of large scale visual recognition from different perspectives. By exploring the hierarchical structure of label space, the diverse distribution of the image samples, and the redundancy in the convolutional kernels, we attempt to reduce the necessary amount of computation while maintaining accuracies as good as traditional recognition systems.

In chapter. 3, a probabilistic label tree model is presented to preform large scale multi-class classification with sub-linear complexity. The label tree framework arranges the label space in a tree structure so that the prediction of one sample can be implemented by traversing the tree from root. Different from the hard label partition in traditional label tree model, the proposed probabilistic framework maintains a categorical distribution of all the labels at each non-leaf node. This brings several advantages including preservation of prior information, and more natural relaxation. The tree is learned stage-wise in a recursive fashion, and the classifier in each node is trained with an iterative maximum likelihood optimization scheme. We show that the proposed probabilistic framework achieves significantly higher accuracy with similar speedup to previous work.

In chapter. 4, we propose a sparse decomposition based method to reduce the complexity of Convolutional Neural Networks (CNN). By decomposing the convolutional kernels along both the input channel dimension and the spatial dimensions, we are able to fully exploit the redundancy underlying the traditional kernels. We learn the sparsity with initial decomposition while minimizing reconstruction error, and further whole network level fine-tuning. Both of these two steps are shown by our experiments to be essential to achieve high sparsity. While maintaining similar

accuracy, the proposed method is able to learn a model with more than 90% sparsity. To improve the practical running speed of our sparse CNN model, we also propose a CPU based efficient sparse matrix multiplication algorithm. The proposed method encodes the location of the non-zeros elements in the sparse kernel into the program, such that the jumping memory access can be effectively avoided.

In chapter. 5, we build a multi-stage CNN framework by combining the idea of label tree and cascade model. While the label tree framework divides the label space, the cascade model divides the sample space. Instead of using one accurate but complex CNN model to process all images, we use a much more efficient model to classify the images that are relatively easy to distinguish, and assign the images that are more difficult to classify to one of the label tree clusters. Such a multi-stage CNN framework is able to achieve similar accuracy to a single complex model, while the average amount of computation is significantly reduced since each CNN model only need to handle a much simpler problem than the original model.

6.2 Future Work

6.2.1 Sparse Convolutional Neural Networks

An efficient sparse matrix multiplication algorithm that is specifically designed for current GPU architecture will enable the proposed sparse model to be applied more widely. The discontinuous memory access problem for sparse matrix will be greatly alleviated if the sparse matrix possesses certain type of structure that is friendly to system cache and pipeline. A structured sparsity loss function can be designed to learn such type of structure. The proposed algorithm is also limited to fine-tuning from a pre-trained model and cannot accelerate the training process. Learning low-rank and sparsity from the scratch would be able to improve both the training and inference process,

although will be also more challenging due to lack of decomposition.

6.2.2 Multi-stage CNN framework

The proposed framework uses each CNN model as a basic building block. While such a framework is able to provide decent speedup, the CNN models have no interaction with each other, and are probably learning large amount of shared information. A more natural solution would be to train one single model that has the mechanism of automatic branching and early termination. Such a model requires the integration of a widespread high-level gating function that is only turned on when necessary. This would require significant modification to the traditional CNN model, and careful design of regularization to prevent overfitting problem.

LIST OF REFERENCES

[1] M. Aharon, M. Elad, and A. Bruckstein. k -svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, Nov 2006.

[2] Zeynep Akata, Florent Perronnin, Zaid Harchaoui, and Cordelia Schmid. Label-embedding for attribute-based classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 819–826, 2013.

[3] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*, 4(1):1–106, 2012.

[4] Richard Barrett, Michael W Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*, volume 43. Siam, 1994.

[5] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[6] Samy Bengio, Jason Weston, and David Grangier. Label embedding trees for large multi-class tasks. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *Advances in Neural Information Processing Systems*, pages 163–171, 2010.

[7] Samy Bengio, Jason Weston, and David Grangier. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems*, pages 163–171, 2010.

[8] Thomas Blumensath and Mike E Davies. Iterative hard thresholding for compressed sensing. *Applied and Computational Harmonic Analysis*, 27(3):265–274, 2009.

[9] M. Bray, E. Koller-Meier, P. Müller, L. Van Gool, and N.N. Schraudolph. 3d hand tracking by rapid stochastic gradient descent using a skinning model. In *In 1st European Conference on Visual Media Production (CVMP*. Citeseer, 2004.

[10] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.

[11] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. *CoRR, abs/1504.04788*, 2015.

[12] Yangchi Chen, Melba M Crawford, and Joydeep Ghosh. Integrating support vector machines in a hierarchical output space decomposition framework. In *Geoscience and Remote Sensing Symposium, 2004. IGARSS'04. Proceedings. 2004 IEEE International*, volume 2, pages 949–952. IEEE, 2004.

[13] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865, 2015.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[15] Jia Deng, Sanjeev Satheesh, Alexander C Berg, and Fei Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In *Advances in Neural Information Processing Systems*, pages 567–575, 2011.

[16] Jia Deng, Sanjeev Satheesh, Alexander C. Berg, and Fei-Fei Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In *Advances in Neural Information Processing Systems*, pages 567–575, 2011.

[17] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

[18] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, 2014.

[19] Kai-Bo Duan and S Sathiya Keerthi. Which is the best multiclass svm method? an empirical study. In *International Workshop on Multiple Classifier Systems*, pages 278–285. Springer, 2005.

[20] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[21] Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image processing*, 15(12):3736–3745, 2006.

[22] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[23] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[24] Clément Farabet, Yann LeCun, Koray Kavukcuoglu, Eugenio Culurciello, Berin Martini, Polina Akselrod, and Selcuk Talay. Large-scale fpga-based convolutional networks. *Machine Learning on Very Large Data Sets*, 2011.

[25] Pedro F Felzenszwalb, Ross B Girshick, and David McAllester. Cascade object detection with deformable part models. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 2241–2248. IEEE, 2010.

[26] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *Advances in neural information processing systems*, pages 2121–2129, 2013.

[27] Wenjiang J Fu. Penalized regressions: the bridge versus the lasso. *Journal of computational and graphical statistics*, 7(3):397–416, 1998.

[28] Tianshi Gao and Daphne Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2072–2079. IEEE, 2011.

[29] Mohammad Gharavi-Alkhansari and Thomas S Huang. A fast orthogonal matching pursuit algorithm. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 3, pages 1389–1392. IEEE, 1998.

[30] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[31] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[32] Yunchao Gong, Qifa Ke, Michael Isard, and Svetlana Lazebnik. A multi-view embedding space for modeling internet images, tags, and their semantics. *International journal of computer vision*, 106(2):210–233, 2014.

[33] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[34] Georgios Goumas, Kornilios Kourtis, Nikos Anastopoulos, Vasileios Karakasis, and Nectarios Koziris. Performance evaluation of the sparse matrix-vector multiplication on modern architectures. *The Journal of Supercomputing*, 50(1):36–77, 2009.

[35] Gregory Griffin and Pietro Perona. Learning and using taxonomies for fast visual categorization. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[36] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015.

[37] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5353–5360, 2015.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 37(9):1904–1916, 2015.

[41] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[42] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.

[43] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002.

[44] Eun-Jin Im, Katherine Yelick, and Richard Vuduc. Sparsity: Optimization framework for sparse matrix kernels. *International Journal of High Performance Computing Applications*, 18(1):135–158, 2004.

[45] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[46] Tommi S Jaakkola, David Haussler, et al. Exploiting generative models in discriminative classifiers. *Advances in neural information processing systems*, pages 487–493, 1999.

[47] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proc. BMVC*, 2014.

[48] Herve Jegou, Florent Perronnin, Matthijs Douze, Jorge Sánchez, Patrick Perez, and Cordelia Schmid. Aggregating local image descriptors into compact codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1704–1716, 2012.

[49] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast

feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[52] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[53] Andrew Lavin and Gray Scott. Linear spatial pyramid matching using sparse coding for image classification. In *Computer Vision and Pattern Recognition, 2016. CVPR 2016. IEEE Conference on*. IEEE, 2016.

[54] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[55] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[56] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[57] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pages 801–808, 2006.

[58] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[59] Baoyuan Liu, Fereshteh Sadeghi, Marshall Tappen, Ohad Shamir, and Ce Liu. Probabilistic label trees for efficient large scale image classification. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 843–850. IEEE, 2013.

[60] Song Liu, Haoran Yi, L-T Chia, and Deepu Rajan. Adaptive hierarchical multi-class svm classifier for texture-based image classification. In *2005 IEEE International Conference on Multimedia and Expo*, pages 4–pp. IEEE, 2005.

[61] Xing Liu, Mikhail Smelyanskiy, Edmond Chow, and Pradeep Dubey. Efficient sparse matrix-vector multiplication on x86-based many-core processors. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*, pages 273–282. ACM, 2013.

[62] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[63] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 689–696. ACM, 2009.

[64] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(Jan):19–60, 2010.

[65] Julien Mairal, Michael Elad, and Guillermo Sapiro. Sparse representation for color image restoration. *IEEE Transactions on image processing*, 17(1):53–69, 2008.

[66] Subhransu Maji and Alexander C Berg. Max-margin additive classifiers for detection. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 40–47. IEEE, 2009.

[67] Stéphane G Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12):3397–3415, 1993.

[68] Marcin Marszałek and Cordelia Schmid. Constructing category hierarchies for visual recognition. In *European Conference on Computer Vision*, pages 479–491. Springer, 2008.

[69] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. In *International Conference on Learning Representations (ICLR2014)*. CBLS, April 2014.

[70] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[71] Rajesh Nishtala, Richard W Vuduc, James W Demmel, and Katherine A Yelick. When cache blocking of sparse matrix vector multiply works and why. *Applicable Algebra in Engineering, Communication and Computing*, 18(3):297–311, 2007.

[72] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

[73] Leonid Oliker, Xiaoye Li, Parry Husbands, and Rupak Biswas. Effects of ordering strategies and programming paradigms on sparse matrix computations. *Siam Review*, 44(3):373–393, 2002.

[74] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[75] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In *Computer Vision–ECCV 2010*, pages 143–156. Springer, 2010.

[76] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1665–1672. IEEE, 2011.

[77] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[78] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[79] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[80] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016.

[81] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[82] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[83] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[84] Jasper RR Uijlings, Koen EA van de Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[85] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

[86] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

[87] Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(3):480–492, 2012.

[88] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[89] S Vishwanathan, N Schraudolph, M. Schmidt, and K. Murphy. Accelerated training of conditional random fields with stochastic meta-descent. In *International Conference on Machine Learning (ICML '06)*, 2006.

[90] Volkan Vural and Jennifer G Dy. A hierarchical method for multi-class support vector machines. In *Proceedings of the twenty-first international conference on Machine learning*, page 105. ACM, 2004.

[91] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3360–3367. IEEE, 2010.

[92] Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3360–3367. IEEE, 2010.

[93] Li Wang and Dong-Chen He. Texture classification using texture spectrum. *Pattern Recognition*, 23(8):905–910, 1990.

[94] Jeremiah Willcock and Andrew Lumsdaine. Accelerating sparse matrix computations via data compression. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 307–316. ACM, 2006.

[95] John Wright, Allen Y Yang, Arvind Ganesh, S Shankar Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE transactions on pattern analysis and machine intelligence*, 31(2):210–227, 2009.

[96] Zhang Xianyi, Wang Qian, and Zaheer Chothia. Openblas. *URL: http://xianyi. github. io/OpenBLAS*, 2012.

[97] Jianchao Yang, Kai Yu, Yihong Gong, and Tingwen Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1794–1801. IEEE, 2009.

[98] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1476–1483, 2015.

[99] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer vision–ECCV 2014*, pages 818–833. Springer, 2014.

[100] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1984–1992, 2015.

[101] Liu Zhigang, Shi Wenzhong, Qin Qianqing, Li Xiaowen, and Xie Donghui. Hierarchical support vector machines. In *Proceedings. 2005 IEEE International Geoscience and Remote Sensing Symposium, 2005. IGARSS'05.*, volume 1, pages 4–pp. Ieee, 2005.

[102] M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. *Advances in Neural Information Processing Systems*, 23(23):1–9, 2010.