SCALABLE OPTIMIZATION METHODS WITH SIDE INFORMATION IN IMAGE UNDERSTANDING

by

Maxwell D. Collins

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2017

Date of final oral examination: October 20, 2017

For my family.

ACKNOWLEDGMENTS

The road to writing this thesis has followed a roundabout path at times. It never would have been completed if not for the incredible support of family, friends, and colleagues. I benefited more from their encouragement and patience than I ever could have asked. I would especially like to thank an incredible set of people who went out of their way to mentor and guide me towards where I am now as a computer scientist. Richard McGehee, Robert Miner, Volkan Isler, and finally my advisor Vikas Singh all generously gave their time.

I also never would have succeeded through the key phase of my maturation as an Artificial Intelligence researcher without the support of a wonderful community of friends and co-workers in my graduate lab, and in the AI group across the University of Wisconsin-Madison. Christopher Hinrichs, Brandon Smith, Jia Xu, Deepti Pachauri, Kamiya Motwani, Won Hwa Kim, and Hyunwoo Kim were all a joy to get to know. As colleagues, each of them set a stellar example of what a computer scientist should be, displaying skill, dedication, and perseverance beyond anything I have seen elsewhere.

My family also provided a ton of encouragement and support throughout my education. One could mark the start of my career when my sister Nicole arranged my first opportunity to visit a real AI lab, at the University of Minnesota, when I was a high school student. Since then, every member of my close extended family have always been incredibly and unconditionally encouraging and positive.

Finally, none of the work in this dissertation could have been done without having incredibly talented collaborators. Vikas Singh and Pushmeet Kohli lent me their incredible insight, by setting before me some awesome problems to solve. I learned most of what I know from working with them and Leo Grady, Ji Liu, Lopamudra Mukherjee, and Jia Xu.

TABLE OF CONTENTS

Appendix

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The field of Computer Vision includes a highly varied collection of technologies for using Artificial Intelligence to understand and process images. A common thread throughout Computer Vision is mathematical optimization, frequently used as a tool to model and solve these problems. A key advantage of optimization formulations is that a model of any basic Computer Vision problem can be extended to include additional information and requirements. Prior knowledge, side information, and application-specific restrictions can be expressed within the objective or constraints. This provides a general way to formally pose modifications to common vision algorithms. It is not entirely sufficient, however, to only describe an optimization model that includes the desired side information. The wide array of problems that can be formulated this way is accompanied by a similarly wide range of computational difficulty. While the optimization problem for a standard Computer Vision problem may be solved by a simple and efficient algorithm, with included side information the extended problem can be fundamentally harder and require more complex solvers.

The focus of this dissertation is a set of vision applications in image segmentation, clustering, and classification that include side information. For the extended problems, I describe scalable and distributed algorithms that allow even the harder optimizations to be solved efficiently. In the case of segmentation, the inference is extended to consider multiple images related by the presence of a common foreground, with an interactive implementation that can parallelize across the computational units of a Graphics Processing Unit (GPU). Then, a distributed image clustering algorithm that can incorporate side constraints is presented. The final problem that is considered is the use of side constraints in neural network training to build image classifiers with reduced memory requirements. This dissertation shows that

modeling Computer Vision problems as an optimization effectively provides a way both to reason about the kinds application-specific extensions presented in these examples and to make finding a solution fast and efficient.

Chapter 1

Introduction

A broad range of problems in Computer Vision are posed as numerical optimizations of an objective function that acts as a surrogate for a perceptual model of image interpretation and understanding. In the ideal case, the decision variables at the maximum or minimum of the optimization correspond to the inference that a human would draw from the image. For example, in the "optical flow" problem, the goal is to identify the transformation between a source and a target image that describes the perceptual correspondence between pixels in each image [155]. In the stereo problem, the optimal choice of depths of individual objects in the 3D scene will be those that could most likely produce the given pair of images of the scene being captured [125]. The segmentation problem is often posed as an energy minimization over the object or class labels, where labels for each pixel identify the extent of the foreground and background regions. Within a popular class of modern approaches that are often referred to as "Deep Learning," the process of building a learned model is to "train" it by optimizing its weights to fit so that its response on the training data matches the desired human response. To summarize, the overwhelming majority of current approaches in Computer Vision involve some form of minimization or maximization over a set of optimization variables. Progress on the theoretical and implementation sides of such algorithms and their deployment as mature software libraries has been a key driver of progress in the Computer Vision field throughout most of the past two decades.

In practice, constructing an optimization model for a Computer Vision problem involves a sequence of interrelated steps. First, the notion of what constitutes a good solution to the problem is described in terms of an objective function. For the unknown desired solution, we want to infer from the image, or the set of images, the objective function that will determine what is the “best” choice of these unknown variables. The model must frequently also include constraints on what is a valid solution. For example, often the unknown variables must lie between 0 and 1 to facilitate interpretation as output for the target problem. The optimization model is formed from the combination of this objective, the constraints, and the variables that they act upon. This model can frequently be intractable or very hard to optimize, but researchers usually seek formulations for which efficient and well-characterized algorithms are known. Depending on whether our decision variables are discrete or continuous, two major classes of methods can be applied, and correspond to discrete/combinatorial and continuous optimization models, respectively. Within the discrete optimization field, techniques commonly used in Computer Vision include graph cuts [20], dynamic programming [52], submodular methods [94], branch and bound [144], and even greedy schemes [15]. In continuous optimization, it is common to use gradient descent, including stochastic gradient descent and conjugate and accelerated gradients, and Newton's method. With constraints, techniques such as the simplex method, penalty and barrier methods, or primal-dual methods are used. Many of these methods are either accessed natively through Computer Vision libraries such as OpenCV [148] or VLFeat [186], or within vision-focused optimization and training libraries such as Tensorflow [1], or they are implemented within optimization toolboxes such as IPOPT, CPLEX, or CVX [62].

Historically, the core focus in Computer Vision has been on tasks that are defined to work on a single image and make a single specific inference about that image. In other words, the input corresponds to one image on which the inference produces one output or solution. Part of this is because an image in isolation is the minimal possible input that one may seek to “understand” using Computer Vision methods. The preference for this strategy can also be attributed to the fact that many of the optimization models are difficult to optimize, and until recently, the hardware could not cope with an image containing millions of pixels and at least as many decision variables. However, the setting of one image in, and one decision out, does not match how images usually appear in the real world, and a person's understanding of the image in one context may be very different from their understanding of that same image in another setting. For example, some of the semantic meaning of an image in an album may only be apparent by looking at the full set. This is part of why it has proven useful to include "side information" in vision tasks. Side information can be defined as any secondary information that facilitates the target task but explicitly does not appear in the image itself. Conceptually, this can be thought of as a prior, but often may also involve information *not* available when building the initial model. This side information can also take many different forms depending on the application of interest, including the context or contexts in which the image appears; other similar images, tags, or annotations about that image; or even active involvement of a user. More often than not, leveraging such side information, while highly meaningful to the application, poses numerous challenges. In particular, when a model is modified to take into account new types of information, it may no longer be solvable by standard mature optimization solvers. Typically, "template" versions of the optimization problem for a simpler formulation of the problem can treat the solver as a black box with strong preexisting theoretical guarantees. However, the theory as well as the expected empirical performance of the black box solver do not always generalize to the augmented model, as will be demonstrated shortly with a concrete example. A central focus of this dissertation is to study several different instantiations of this challenge in standard Computer Vision settings and describe effective ways to perform the resulting optimizations.

We also want to include this side information in a *scalable* way. Improvements in sensor technology have led to a rapid increase in the resolution of the images captured by typical cameras. For inference that must ideally be based in a fine-grained way on the content of full-resolution images, this requires working with much larger inputs to the underlying optimization model. Further, people worldwide have taken up more and more prolific capturing and sharing of photos, producing a deluge of image data that one would want to understand and utilize. While the hardware that can be brought to bear on these larger problems has also grown in its power, this has not been enough, by itself, to cope with the growth in the necessary computations. First, the growth in hardware has not been uniform. The most rapid growth has been in hardware that executes large numbers of operations in parallel, such as GPUs, or improved techniques for building clusters of commodity hardware. Naïve implementations of the same algorithms that worked for smaller problems will not always scale out of the box when simply run on a newer processor. Further, even to the extent that both hardware and the size of the problems have scaled, users increasingly demand higher accuracy on more complex vision problems, with less user intervention, and this requires solving computationally harder problems. A central focus of this dissertation is producing models, including those with side information, that better *scale* to big datasets and better leverage the available computational resources and specialized architectures (e.g., GPUs) that are dominating the landscape of Computer Vision research today.

1.1 Graph Cut Segmentation with Side Information

As a practical example of this modelling approach with "side information" and what it involves, this section starts with the classical graph cut segmentation approach [19]. Recall that the term "segmentation" describes a general class of vision problems that seek to assign some semantically meaningful label to each pixel in an image. This group describes such related problems as motion segmentation [63], semantic segmentation [49], and what we consider in this example: figure/ground segmentation [19], [92], [128].

One of the most widely used models for segmentation is Markov Random Fields (MRFs) [117]. An MRF is a probabilistic model defined over *cliques* of random variables. In segmentation, each random variable is a segmentation label or class for a single pixel. In the figure-ground segmentation problem, these are binary variables that take a value of 1 if the pixel p belongs to the foreground and a value of 0 if the pixel p belongs to the background. Each clique is over a set of variables, e.g., $\{p, q, r\}$, that are mutually dependent. For example, two adjacent pixels p and q will share an edge (a clique of size two) with a probability distribution modelling the expectation that two adjacent similar pixels both belong to the foreground or both belong to the background. This is frequently called a "smoothness" prior, in that it is used in models that expect the labels to be piecewise-smooth. For a binary figure-ground segmentation, this means that the foreground and background pixels will tend to occur as part of contiguous regions. The model will also include *unary* components for each variable that assign a probability, based on the appearance of just one pixel, to whether that pixel belongs to the foreground as considered independently of any other pixels.

To build an optimization problem, it is common to pose the MRF in the form of an equivalent Gibbs energy function [22], where each term is a potential function on the labels of one clique. The set of labels that minimize this Gibbs energy will also be the maximum a posteriori (MAP) solution of the corresponding MRF, giving the most likely set of labels under that probability distribution. The authors in [20] describe this energy function with the formula:

$$E(f) = \sum_{\{p,q\}\in\mathcal{N}} V_{p,q}(f_p, f_q) + \sum_{p\in\mathcal{P}} D_p(f_p). \tag{1.1}$$

In this notation, each f_p is the output label for pixel p, and $\mathcal{N}$ is a set of neighborhood relations between pairs of pixels. D_p and $V_{p,q}$ are clique potentials, as described above. Let us use potentials constructed as follows. First, the pairwise potential can use the *Potts energy* that penalizes, with a weight, each pair of similar pixels that are assigned different labels:

$$V_{p,q}(f_p, f_q) = \begin{cases} B_{p,q} & \text{if } f_p \neq f_q \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

where $B_{p,q}$ is some measure of the similarity between pixels p and q. This quantity can be based on, for example, the intensity difference or distance between the RGB vectors, and it will have a higher value for more similar pixels. The unary potentials $D_p(f_p)$ are defined based on an appearance model of the foreground and background—for instance a Gaussian Mixture Model (GMM) over the RGB values [169] for a target class—or learned from interactive seed points. Specifically, parameterizing the color distribution of the foreground with a GMM model of K components, the probability of a pixel p having color x_p is given by:

$$p(x_p|f_p = l) = \sum_{k=1}^{K} \pi_{lk}\mathcal{N}(x_p|\bar{x}_k, \Sigma_k) \tag{1.3}$$

for per-class component weights π_{lk}. Take the mean and covariance $\bar{x}_k, \Sigma_k$ of each component's normal distributions to be shared between classes. The energy can then have the form:

$$D_p(f_p) = \log \sum_k \pi_{lk} p(k|x_p). \tag{1.4}$$

This problem, when posed over binary labels, can be solved using the Graph Cuts algorithm [19]. This uses a classical "min-cut" or "max-flow" algorithm that solves the problem of finding the set of edges in a graph that separate a source and sink node and have minimum total weight. One can construct a specialized network such that pixels are nodes in the graph, to which source and sink nodes that are connected to the pixel nodes are added. The smoothness terms are encoded in the edges between the pixel nodes. The unary potentials are further included as the weights between the pixel nodes and the source and sink nodes. Pixels on the same side of the minimum cut as the source node are assigned to foreground and the rest to background.

Extended sets of labels, including an integer number of ordered labels with an appropriate smoothness function, can also be used in an MRF while still allowing for a guaranteed globally optimal solution [83]. Each of these extensions makes the problem harder to solve as it grows along these dimensions. The method in [83], for instance, uses a graph construction with a number of nodes that grows with the product of the number of pixels and the number of possible labels that can be assigned to them. It is applicable to MRFs, like those above, that have only unary and pairwise cliques. Formulations based on the min-cut algorithm can also exactly solve for the MAP of an MRF for figure-ground segmentation, as long as every projection of the energy function onto two variables is *regular* as defined in [97].

In general, MRFs can also be solved by the *belief propagation* algorithm [147], [193], even when none of the above conditions hold. Belief propagation is a message-passing algorithm where, after initializing each node to zero, it incrementally updates each node from its neighbors' messages with:

$$m^t_{p\to q}(f_q) = \min_{f_p} V(f_p, f_q) + D_p(f_p) + \sum m^{t-1}_{s\to p}(f_p), \tag{1.5}$$

as in [51]. These updates can be guaranteed to converge to an optimal solution in the case that there are no loops in the message-passing. For MRFs that do produce loops, finding the MAP solution is NP-hard, and in practical cases one can produce approximate solutions at best. However, in the case that there is a loop, it has been shown empirically that "loopy" belief propagation will produce good approximate solutions on tasks like denoising and stereo [51], [133].

1.1.1 Examples of Incorporating Side Information

Example 1. The additional complexity that can arise when using side information is demonstrated by the "cosegmentation" model in [156]. That work takes, as a starting point, the MRF figure-ground segmentation described above. They take a problem where they combine two images, each containing the same foreground object, as the input to a common segmentation task for which they want to label the foreground object in both. The additional prior knowledge that the foreground is present in both images, especially in cases where the background is different, can significantly improve the quality of the output segmentation. In the work in [156], the MRF segmentation model was extended to include a term that a histogram over the foreground pixels be roughly consistent between the pair of images in the cosegmentation input. This will, for instance, exclude background elements that are not common between the images, because including them would increase the mismatch between these histograms and make the labelling suboptimal. The original MRF segmentation on each individual image can be solved by graph cuts, as described above. The histogram consistency term, though, is a *supermodular* function of the input labels. Within an energy minimization framework, the sum of the submodular MRF objective and the supermodular histogram consistency term is NP-hard to solve exactly. Empirically, however, the authors find that a trust region method that locally approximates the supermodular term with a linear function tends to converge on good segmentations.

Example 2. The work in [187] imposes an additional *connectivity* constraint on the segmentation output. This constraint especially helps in producing good segmentations of elongated or thin objects. The “shrinking bias” normally imposed by graph cut segmentation, caused by the smoothness term penalizing long boundaries relative to small regions of foreground, can undesirably cut out these regions. The formulation in [187] accepts as an additional input some set of points in the image that both belong to the foreground and belong to a single connected component in the pixel adjacency graph. The additional constraint is that for a pair of these points, there should be a path on an adjacency graph that connects these points and for which each pixel in the path is labeled as part of the foreground. The authors of [187] show, however, that the result in [207] has as a corollary that this constraint makes the segmentation problem NP-hard. Vicente et al. therefore propose a heuristic method called DijkstraGC as a solver for these constrained segmentations. DijkstraGC finds multiple MRF segmentations as a subroutine within finding shortest paths that lie in the resulting foregrounds.

Example 3. The authors of [107] present an MRF-based semantic segmentation technique that incorporates additional prior terms on the co-occurrence of class labels. These terms include some common-sense notions of what objects and classes of things are likely to appear together in the same image. It is unsurprising to label one pixel as belonging to a cow and an adjacent pixel as being grass; it is less common to see a television in the same scene. Very unlikely juxtapositions of labels are more probably a segmentation error, so a labelling that is consistent with other parts of the model but uses only labels that are likely to co-occur may be preferable. Prior formulations [57], [149], [180] of co-occurrence required the instantiation of a fully connected graph over the label variables. This was potentially very expensive, as the number of edges in this graph grows quadratically with the number of pixels in the input image. Ladicky et al. [107] instead define an energy function that is directly a function of the set of labels that are present in the segmentation, regardless of which pixels or exactly how many pixels take that label. This means it is not distinguishing between the case that one pixel has a particular label, or one hundred pixels do. Their resulting energy function is posed as an integer program, which is then relaxed to a linear program that is solved with a graph-cut-based move-making algorithm. Again, the model cannot be optimally solved in polynomial time, and few theoretical guarantees are available.

Example 4. Toshev et al., in [179], develop a segmentation method that incorporates an objective term based on the foreground object's shape. Much like the previous examples, the original baseline segmentation model, without this additional shape prior, is tractable. In this case, the baseline segmentation can be computed with a quadratic programming problem over the labels assigned to superpixels that are computed with NCuts [168]. Their shape prior is modeled as a *chordiogram*, a descriptor of the expected 2D shape of the object. It is constructed by taking a histogram over chord lengths, giving the distribution of distances over pairs of points on the boundary of the segmented object. The resulting optimization, with the chordiogram matching term, is an integer quadratic programming problem, which is NP-hard to solve exactly. The authors present a Semi-definite Programming (SDP) relaxation that can be solved with standard tools for this class of problems, with their empirical results computed using SeDuMi [173].

1.2 Problems Studied in this Thesis

This dissertation studies, from the theoretical and implementation points of view, the efficiency and algorithmic issues in incorporating and leveraging side information within optimization methods for Computer Vision problems. The following sections describe the problems that are discussed in the chapters of this thesis at a high level and highlight the core contributions.

1.2.1 Random Walker Cosegmentation

The basic goal of cosegmentation is to segment a common salient foreground object from two or more images, as shown in Figure 1.1. It uses the additional information that a set

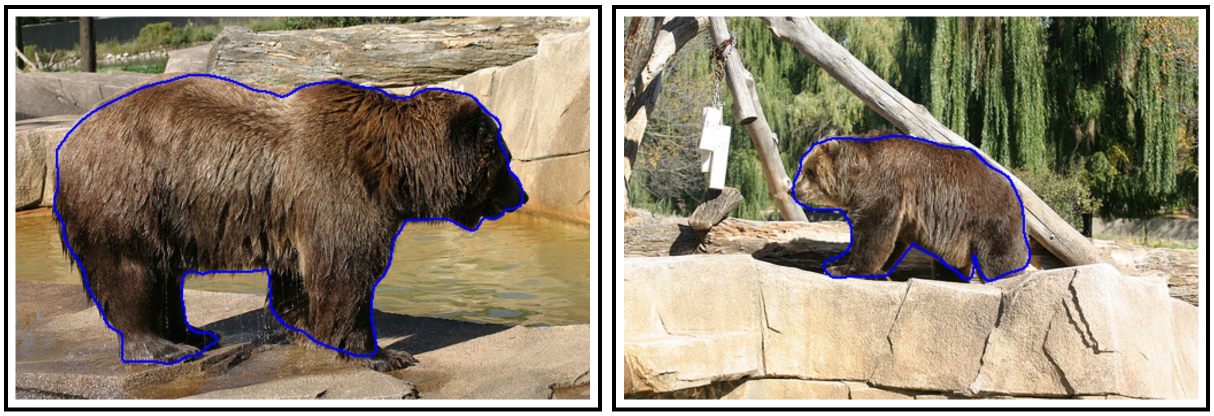


Figure 1.1: A representative application for cosegmentation. The same foreground, the bear, appears in both images. The goal is to find the segmentations outlined in blue.

of images contain the same object, and that some properties of the object's appearance are shared between the instances. This is useful when one wants to analyze multiple images and extract and understand their commonality. The information that multiple images share in common can help improve the segmentation model. Within, for example, an interactive cosegmentation, it's possible that only a subset of images need labels at all, with the remainder of the images being segmented using only the histogram prior. Here, consistency between the labelled object regions is accomplished by imposing a *global* constraint that penalizes variations between the object's respective histograms or appearance models. The idea has been adopted for obtaining concise measures of image similarity [156], discovering common appearance patterns in image sets [113], medical imaging [75], and building 3D models from community photo collections [5], [99]. However, most previous work provided insights on cosegmentation only in the context of the traditional Markov Random Field (MRF)-based segmentation objective [19]. Adding such a global constraint adds a fair bit of overhead to some existing MRF methods. Consider the simple toy example in Figure 1.2, where the segmentation should identify the common blue circle in distinct backgrounds. MRF approaches for cosegmentation with a global model require that an auxiliary node or variable be introduced into the graph on which the network flow will be executed whenever

Figure 1.2: Toy example for cosegmentation demonstrating a uniform foreground that we would like to segment.

two pixels share the same bin in the histogram [75], [129]. While the segmentation of the blue circles by itself is easy, the cost of introducing an auxiliary node for perceptually similar pixels can grow *very* quickly – counting, for example, the foreground pixels for a 196×196 image pair. As a result, these previous models are limited to feasibly cosegmenting only those image pairs that have a relatively high entropy distribution: a distribution where each bin is shared by only a few pixels.

Chapter 3 provides a solution to this problem. It describes a cosegmentation model with the Random Walker segmentation at its core. The model reduces to a convex program with box constraints. Based on this structure, I propose a specialized and efficient gradient projection-based procedure that finds a global real-value optimum of the model and preserves many advantages of Random Walker [61]. The model allows for a nonparametric representation of the foregrounds—for example using distributions over texture words—but one which permits any distribution of features without incurring additional computational costs. This provides a substantial advantage over the existing nonparametric cosegmentation approaches, which are limited only to regions described by a high entropy model in which object features must be spread evenly across bins. I also extend this model to a scale-independent penalty. The treatment presents a novel optimization method for a class of objectives based on quasiconvex functions. I later prove correctness and demonstrate it for model-based image segmentation. These theoretical results are of independent interest. Interestingly, our optimization consists of linear algebra operations (BLAS Level 1, 2) on sparse matrices, which has significant benefits. For example, the algorithm is easy to execute on a GPU architecture, as I show using extensive experiments.

1.2.2 Parallel Spectral Clustering with Nonsmooth Side Information

Clustering serves as an important exploratory tool for categorizing sets of images into semantically meaningful concepts. To facilitate such inference tasks, the image is first expressed in terms of its response to a large set of specialized filters pertaining to texture, distinct object categories, and appearance, among others. This information may be further complemented by textual cues that co-occur with the images, image tags, or hyperlinks to the image as "side information." With these representations in hand, the goal is to leverage all views simultaneously and derive a solution that best groups the given set of examples. Once this is done image understanding can proceed cluster-by-cluster—using a cosegmentation method, for instance.

One of the most general ways to extract clusters from this kind of highly complex data is Spectral Clustering [25], due to its ability to find the structure of arbitrarily shaped clusters. Despite its advantages, Spectral Clustering is expensive for larger datasets. The most common optimization is to sparsify [29] or subsample [55], [116] the matrix of similarities between examples. Even with these optimizations, solving the Spectral Clustering problem for very large datasets is expensive—mainly due to the need for an eigendecomposition of big matrices. These issues clearly intensify when operating with multiple views of the data and when incorporating side information such as tags. To address these limitations, there is significant recent interest in making Spectral Clustering efficient in the large dataset setting—with user interaction [30], "activization" schemes [100], and parallelized versions of the Lanczos solver [29]. These solutions are highly effective for the standard Spectral Clustering objective and several also come with nice guarantees. However, they are not necessarily straightforward to adapt to run in a distributed manner over multiple threads. More importantly, incorporating weak or distant supervision, user interaction, and/or auxiliary domain-specific priors beyond must-link/cannot-link constraints is challenging. Such side information is ubiquitous in most real-world datasets and seems like a desirable feature for a system deployed in practice, producing the need to overcome these computational difficulties.

My primary goal in Chapter 4 is to develop stochastic and distributed optimization methods for this class of problems. To effectively utilize the side information, I focus on on Spectral Clustering for both single and multi-view data. The algorithms in that chapter satisfy some key theoretical and practical criteria that make them applicable to very large problems of this form. First, thanks to using a distributed stochastic descent method, the clustering method can scale to very large datasets, on the order of 100 million examples. To the extent the problem grows, or finding the optimal clustering becomes harder, the problem can still complete in reasonable time, as a larger number of machines can be put to work on a solution. It also shows provably good convergence behavior, using results that are presented in that chapter. It finally offers the ability to incorporate high-level prior knowledge and probability distributions—for example that the images share 'tags' or some element of user interaction—as regularization terms or additional soft constraints.

1.2.3 Sparsity-Inducing Regularizers for Convolutional Neural Nets (CNNs)

Over the last few years, high capacity models such as deep Convolutional Neural Networks (CNNs) have been used to produce state-of-the-art results for a wide variety of fundamental vision problems, including image classification and object detection. This success has been in part attributed to the feasibility of training models with large number of parameters and the availability of large training datasets. These techniques, however, require careful tuning of the optimization and initialization hyperparameters to ensure that the training procedure arrives at a reasonable model [102]. While the capacity of such deep models allows them to learn sophisticated mappings, it also introduces the need for good regularization. The first generation of useful deep models [102], [120], [175] showed a reasonable degree of generalization, in part because of recent advances in regularization methods like DropOut [102].

However, these and other approaches [103] proposed in the literature fail to address the issue of high memory cost, which is particularly problematic in models that rely on multiple fully connected layers. The high memory cost becomes especially important when deploying in applications in computationally constrained architectures like mobile devices.

Satisfying these constraints requires models that come with a side constraint specifying memory limits or other bounds on resources and model capacity. I propose as one promising solution the use of sparsity-inducing regularizers for Convolution Neural Networks (CNNs). These regularizers encourage fewer connections in the convolution and fully connected layers to take non-zero values and in effect result in sparse connectivity between hidden units in the deep network. In doing so, the regularizers not only restrict the model capacity but also reduce the runtime memory cost involved in deploying the learned CNNs. Chapter 5 introduces a set of sparsity-inducing regularization functions, including group-sparse constraints, that are effective at reducing model complexity with no or minimal reduction in accuracy. It also describes updates for these regularizations that are easily implemented within standard existing stochastic-gradient-based deep network training algorithms. This thesis presents empirical validation of the effect of sparsity on CNNs trained on the CIFAR, MNIST, and ImageNet datasets that have, since their initial proposal, been extensively leveraged and replicated by other research groups.

1.2.4 Outline

After introducing preliminary concepts, this thesis is divided into three parts. Chapter 3 presents the Random Walker Cosegmentation model that extends the Random Walker segmentation to include foreground histogram consistency terms as an additional constraint. Next, Chapter 4 describes a highly parallel method for Spectral Clustering that can also incorporate harder must-link constraints and provides general results for including side information into the clustering. Chapter 5 is the final work presented, showing how the training optimization of neural networks can include a sparsity penalty as a side constraint in order to produce smaller and more efficient CNNs.

Chapter 2

Preliminaries

In this chapter, I will briefly introduce preliminary concepts and background upon which this thesis builds. The initial section will cover the basics of continuous optimization, the core technology used throughout this thesis. Next, I'll summarize the literature on stochastic gradient descent (SGD), the type of optimization algorithm used in Chapters 4 and 5. In Chapter 4 specifically, I combine concepts from SGD with optimization on manifolds, a body of work introduced in this chapter in Section 2.3. Finally, to motivate some of the methods in Chapters 3 and 5, I will briefly cover the techniques of sparse linear algebra, a key part of how the implementations produced by the following work are made efficient.

2.1 Continuous Optimization Models and Solvers

To establish terminology, take the general form of a continuous minimization problem:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^d} \quad & f(x) \\ \text{s.t.} \quad & \mathbf{x} \in S. \end{aligned} \tag{2.1}$$

This denotes a problem where the goal is to find the right choice for a d-element vector of *decision variables* $\mathbf{x}$. This vector is chosen from the *feasible set* $S \subseteq \mathbb{R}^n$. This feasible set will be defined by a set of *constraints* on $\mathbf{x}$. For example, if we take S to be the non-negative orthant:

$$\{\mathbf{x} \in \mathbb{R}^d : x_i \geq 0 \text{ for } i = 1, ..., d\}, \tag{2.2}$$

then each expression $x_i \geq 0$ is one of these constraints. Note that in the following text the condition in (2.2) will be abbreviated as $\mathbf{x} \geq 0$, to mean that each element in the vector-valued left-hand side satisfies this inequality.

The structure of the objective function and constraints of an optimization model determines what types of guarantees can be made and the types of solvers that can be applied. Linear programming problems constitute one of the simplest classes, but they are still quite powerful [53]. These are the problems that have a linear objective and linear constraints. All such problems can be expressed as:

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^d} \quad & c^T \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \geq b. \end{aligned} \tag{2.3}$$

A classical method for solving these problems is the simplex method [38].

To build more complex formulations, one of the key tools is the set of *convex* functions. Given a real-valued function $f : \mathbb{R}^d \to \mathbb{R}$, it is convex if Jensen's inequality [18] holds:

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y) \tag{2.4}$$

for all choices x, y in the domain of the function, and all choices λ in the unit interval $[0, 1]$.

One construction we can define on a real-valued function is its sublevel sets. The α-sublevel set of f is defined as:

$$\{\mathbf{x} \in \mathbb{R}^d : f(\mathbf{x}) \leq \alpha\}. \tag{2.5}$$

All sublevel sets of a convex function f will be convex sets [18]. A set $S \subseteq \mathbb{R}^d$ is convex if

$$(1-\lambda)\mathbf{x} + \lambda \mathbf{y} \in S \tag{2.6}$$

for all choices of $\mathbf{x}, \mathbf{y} \in S$ and $\lambda \in [0, 1]$.

This thesis will consider in greater detail the class of convex problems that fall under Quadratic Programming (QP) problems. These are minimization problems where the objective is a quadratic function of the decision variables, and the constraints on the decision

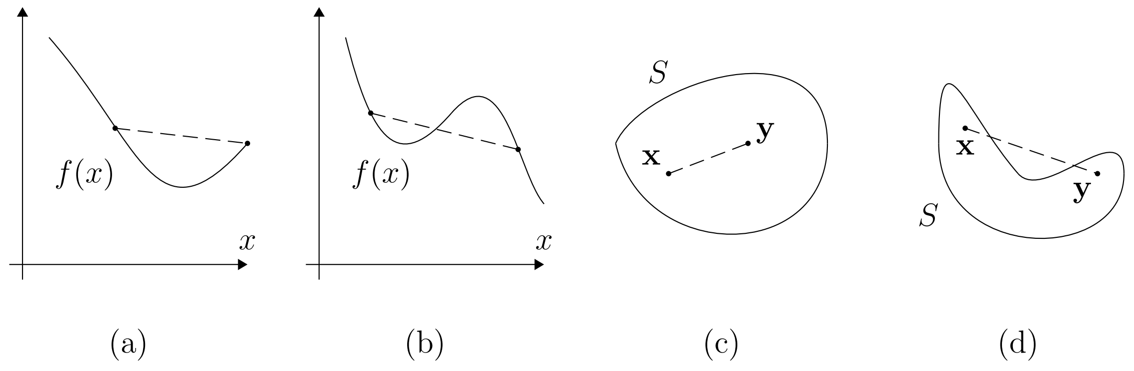


Figure 2.1: Convexity can be a property of both real-valued functions and arbitrary sets. A convex function (a) will satisfy Jensen's inequality (2.4), and will lie below lines between any two choices of $(x, f(x))$ as illustrated. A non-convex function (b) will not. For a convex set S, lines between two points in S as constructed in (2.6) will be contained in the set as illustrated in (c). This will not be the case for a non-convex set (d).

variables are all linear. Quadratic problems are closely related to linear programs, and can be solved by extensions of some of the same methods [198]. These problems take the form:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^d} \quad & \mathbf{x}^T Q \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \geq b. \end{aligned} \tag{2.7}$$

A more general class of important optimization problems is those that are *convex* [18]. Given a problem of the form:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^d} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g_i(\mathbf{x}) \leq 0 \text{ for } i = 1, ..., m \end{aligned} \tag{2.8}$$

it is convex if both the objective function f and all of the m inequality constraint functions g_i are convex.

Related to convexity are a number of weaker and stronger conditions. One of the more general classes of functions for which we have algorithms that yield a guaranteed global solution is the *quasiconvex* functions. These are defined as the functions for which the sublevel sets are convex. Note that this means, by the above discussion, that every convex function is also a quasiconvex function, though the converse is not true. One can also define quasiconvex functions in terms of an inequality analogous to that given for convex functions in (2.4). For quasiconvex functions this condition is given instead by, as in [7], the inequality:

$$f((1-\lambda)x + \lambda x') \leq \max\{f(x), f(x')\}. \tag{2.9}$$

More discussion appears in Chapter 3.

In the opposite direction, there are conditions that define subsets of the convex functions. For instance, a function f is *strongly* convex if for any x we have the inequality (from [18]):

$$\nabla^2 f(x) \succeq mI \tag{2.10}$$

for some positive m and denoting the identity matrix with I. Specifically, this means the minimum eigenvalue of the Hessian is at least m at all points in the function's domain. This excludes objective functions that can display phenomena that make fast convergence arbitrarily difficult. For instance, given a very ill-conditioned Hessian, we may see very shallow descent in some directions and very steep descent in others, making it difficult to establish in general that good progress will be made in the direction of more shallow descent.

The class of convex problems, and especially strongly convex problems, is the most well-characterized subset of optimization problems. There exists a wealth of algorithms and solvers with theoretically proven global convergence guarantees [18], [140]. One of the most basic is gradient descent, or more generally steepest descent. This provides guarantees of a correct globally optimal solution in the case that we have an unconstrained problem $\min_{x \in \mathbb{R}^d} f(x)$. In this procedure, one takes the step

$$x^{(k+1)} \leftarrow x^{(k)} - \alpha_k \nabla f(x^{(k)}) \tag{2.11}$$

for some step length α_k.

The choice of step length at each iteration can dictate the speed of convergence. Some computationally efficient schemes can be performed to find a step length that satisfies the

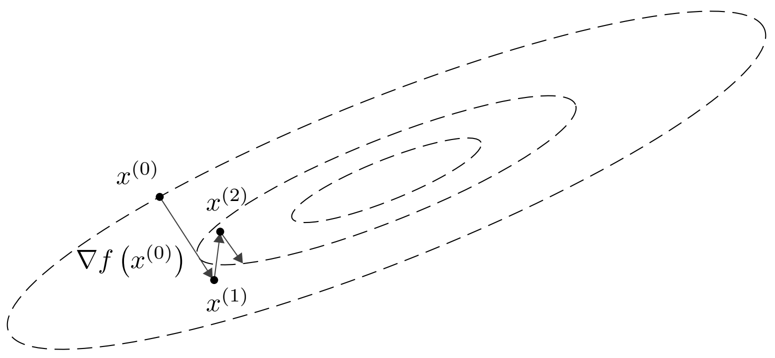


Figure 2.2: Illustration of the gradient descent method. The dashed lines show the level sets of some function. The gradients, shown as arrows, will be perpendicular to the level sets at that iterate. This procedure would continue, typically for more iterations than shown here, until some convergence condition was met.

Armijo and Wolfe conditions, as described in [140], and convergence results can be shown for step lengths so chosen. Also theoretically interesting is *exact* line search, which chooses:

$$\alpha_k := \arg\min_{\alpha>0} f\left(x^{(k)} - \alpha\nabla f(x^{(k)})\right). \tag{2.12}$$

This is frequently impractical to compute except for simpler functions. It can be shown [18], though, that this yields linear convergence for a strongly convex f, meaning that

$$f(x^{(k)}) - f(x^*) \leq c^k \left(f(x^{(0)}) - f(x^*)\right) \tag{2.13}$$

for a constant c that depends on f and x^*, the point to which the descent procedure converges.

One improvement on the search direction is to use the *conjugate gradient.* This combines the gradients evaluated from multiple steps to select the direction of descent. Applied to unconstrained minimization, one such conjugate gradient is given by the Fletcher-Reeves method [54], [140], which will be used in Chapter 3. This chooses a descent direction $p^{(k)}$ at each iteration with the procedure:

$$p^{(k+1)} \leftarrow -\nabla f\left(x^{(k+1)}\right) + \frac{\|\nabla f\left(x^{(k+1)}\right)\|^2}{\|\nabla f\left(x^{(k)}\right)\|^2} p^{(k)}. \tag{2.14}$$

Notably, this method does not require significant additional computation beyond ordinary gradient descent, still requiring one gradient evaluation per iteration. Under mild conditions, it can be shown that the Fletcher-Reeves method will converge to a point with arbitrarily small gradient [140].

One can get significantly faster convergence with Newton and Quasi-Newton methods. These are *second-order* methods, as they use the Hessian matrix of second derivatives of the function, or an approximation of the Hessian. The basic Newton's method takes a step at each iteration of:

$$x^{(k+1)} \leftarrow x^{(k)} - \alpha_k \left[\nabla^2 f\left(x^{(k)}\right)\right]^{-1} \nabla f\left(x^{(k)}\right) \tag{2.15}$$

with step length α_k chosen with a line search. Once one finds an iterate close enough to the solution, Newton's method displays *quadratic convergence*, meaning

$$f\left(x^{(k)}\right) - f(x^*) \in O\left(\left(\frac{1}{2}\right)^{2^k}\right). \tag{2.16}$$

This is quite rapid convergence; it effectively means that the number of additional iterates required "can be considered a constant for practical purposes, say five or six" [18].

One more concept used in many convex optimization algorithms is that of the *active set*. This is the subset of the inequality constraints in an optimization problem of the form in (2.8) that are true with equality. For any point in the feasible set, given by a choice of decision variables $\mathbf{x}$, the active set is defined by:

$$\mathcal{A}(\mathbf{x}) := \{i : g_i(\mathbf{x}) = 0 \text{ for } i \text{ in } 1, ..., m.\} \tag{2.17}$$

Some types of problems have optimality conditions that can make finding a solution much simpler, but are expressed in terms of the unknown active set. For instance, the solution to a linear program can be specified entirely in terms of the choice of the vertex on which the solution lies. For that vertex, any member point will be optimal. Indeed, the simplex method can be considered as the simplest case of a class of optimization techniques called "active set methods" [140]. Active set methods for constrained nonlinear optimization work by alternately solving a problem that assumes a given choice of active set, called the "working set," and updating this choice until it matches the active set at the solution. The active set is also relevant for descent methods, as some may require that we modify the descent direction in order to stay within the feasible set. Locally, whether the descent direction at x points outside the feasible set can be determined by looking at the constraints in the active set.

This discussion includes only one small subset of the optimization algorithms that have been studied for convex problems. Trust region methods, sequential quadratic programming, interior point methods, and a host of other techniques are relevant to such a discussion but are not directly considered in the following work.

2.2 Stochastic Gradient Descent

One optimization algorithm that serves as the crucial building block to many scalable solvers is *stochastic gradient descent* (SGD). Suppose we have an optimization problem:

$$\min_{\theta \in \mathbb{R}^d} \quad f(\theta) := f_1(\theta) + f_2(\theta) + ... + f_m(\theta) \tag{2.18}$$

defined by a collection of differentiable functions $f_i : \mathbb{R}^d \to \mathbb{R}$. In many practical applications of optimization, the objection function can be expressed in this form [14].

Stochastic gradient descent works by subsampling the terms of this objective. In each iteration, first pick some subset $\mathcal{I} \subseteq \{1, ..., m\}$, where each index belongs to $\mathcal{I}$ with equal probability. This can be achieved, for example, by sampling the desired number of indices one at a time, independently and identically distributed, without replacement. Let $f_{\mathcal{I}}(\theta) = \sum_{i \in \mathcal{I}} f(\theta)$. Then two important properties hold:

$$\mathbb{E}_{\mathcal{I}}[f_{\mathcal{I}}(\theta)] = f(\theta) \tag{2.19}$$

$$\mathbb{E}_{\mathcal{I}}[\nabla f_{\mathcal{I}}(\theta)] = \nabla f(\theta) \tag{2.20}$$

meaning that both the objective value and the gradient of the subsampled objective at θ are, in expectation over $\mathcal{I}$, equal to the value and gradient of (2.18)'s original objective function [17]. In the vanilla SGD algorithm, at each iteration t we take a gradient step computed on a set of sub-objectives specified by an $\mathcal{I}_t$ sampled at that iteration:

$$\theta_t \leftarrow \theta_{t-1} - \delta_t \nabla f_{\mathcal{I}_t}(\theta_{t-1}) \tag{2.21}$$

given the iterate from the previous step θ_{t-1} and some step length δ_t.

Some simple common choices for the step length δ_t, also called in ML contexts the "learning rate," are exponential or inverse decaying functions of the step t. Convergence guarantees can be shown for the case that $\delta_t \in \Theta(1/t)$ [14], [132], or more generally learning rates that satisfy:

$$\sum_{t=1}^{\infty} \delta_t = \infty \text{ and } \sum_{t=1}^{\infty} \delta_t^2 < \infty. \tag{2.22}$$

It can be shown under these step length conditions that the objective converges to the true minimum following

$$f(\theta_t) - f(\theta^*) \in O(1/t) \tag{2.23}$$

for θ^* the solution to which the SGD procedure will converge.

Other decaying learning rate schedules are also common in practice. Also common in SGD-based "training" procedures are empirical methods to determine the learning rate, or *hyperparameters* that determine the schedule. These will, for instance, optimize the schedule to maximize the performance on a held-out tuning set that is not used for the SGD training [170]. The learning rate can also be adjusted adaptively, based on the history of previous iterations [206], as another empirical means to determine what this value should be.

Stochastic gradient descent is a practical choice when computing gradients of the full objective is expensive or infeasible [14]. This can be the case when either each f_i is expensive to compute, or the number of terms of the objective is very large. In applications such as primal SVM solvers [166], neural network training [154], and other Machine Learning applications, the objective might be a sum over millions or billions of terms.

One especially important class of problems suitable for SGD is in supervised Machine Learning [14]. Here, we are given a set of *training* examples $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)$, each consisting of input features $\mathbf{x}_i \in \mathcal{X}$ paired with a ground-truth label $y_i \in \mathcal{Y}$. The designer of a Machine Learning model will specify some function $f(\mathbf{x}, \theta)$ that produces a predicted label in $\mathcal{Y}$ given the input features and a collection of unknown weights or parameters θ. One must also select a loss function to compare the labels $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. The value of $\mathcal{L}(y, y')$ will be low if the predicted label y and the true label y' are identical or similar, and will be high if the labels are very dissimilar. The "training" procedure will then be to choose a set of parameters so that the predicted labels will tend to match the true set. In the absence of regularization or side information, one can express the training as an optimization problem of the form:

$$\min_{\theta} \sum_{i=1}^{n} \mathcal{L}(f(\mathbf{x}_i, \theta), y). \tag{2.24}$$

This problem lends itself very well to stochastic gradient descent methods, and each iteration will choose a subset of the input examples.

In addition to the computational benefits, stochastic gradient has some unique and desirable convergence properties when applied to Machine Learning problems. Because SGD acts on a random sampling, it will tend to converge on minima that are stable with respect to this sampling. If a small perturbation of one data point [70] or a small perturbation of the decision variables [93] were to significantly increase the objective in the neighborhood of a local minimum, SGD is less likely to converge to that minimum than one that is more stable. This can improve the *generalization* of an ML model that is trained on this objective, as its output on new unseen data points will be based only on properties of the training set that are robust to these perturbations. These "robust" properties are more likely in practice to be based on the underlying true distribution than the training sample. Stochastic gradient descent can also escape from saddle points [58] on non-convex objectives. Where ordinary gradient descent may see slow progress in an area of flat gradients, even if there are descent directions nearby, the random perturbations in the iterates of an SGD method will cause it to search the neighborhood of the saddle point and find these descent directions.

An important extension to stochastic gradient descent is the use of *accelerated* gradients. One of the most popular variants is SGD with a *momentum* term [174], [196]. This uses as a descent direction a momentum vector $v_t \in \mathbb{R}^d$ that accumulates the gradients over multiple iterations, instead of subtracting the gradient of only $f_{\mathcal{I}_t}$. Both this vector and the current choice of the decision variables are updated in each iteration, as in:

$$v_t \leftarrow \mu v_{t-1} - \delta \nabla f_{\mathcal{I}_t}(\theta_t) \tag{2.25}$$

$$\theta_t \leftarrow \theta_{t-1} + v_t \tag{2.26}$$

where $\mu \in (0, 1)$ is a momentum hyperparameter that is chosen empirically.

Other similar methods such as Nesterov's Accelerated Gradient [135] and Adam [95] have also been shown to yield good convergence in Machine Learning applications.

One extension to the SGD algorithm that can handle more complex problems is *proximal* stochastic gradient descent [201]. This considers problems of the form

$$\min_{\theta \in \mathbb{R}^d} f(\theta) + g(\theta) \tag{2.27}$$

where f is a smooth function from which one can sample stochastic gradients, and g is a possibly non-smooth convex function equipped with a *proximal* operator [145]. The proximal operator for g is the mapping:

$$\operatorname{prox}_g(\theta) = \arg\min_{\theta'} \left(g(\theta) + \frac{1}{2} \|\theta - \theta'\|_2^2 \right). \tag{2.28}$$

A number of different functions have proximal operators that can be computed efficiently. For instance, simple proximal operators can be derived for g being an ℓ_1- or ℓ_2-norm regularization. Note also that when g is the indicator function of a closed convex set G:

$$g(\theta) = \begin{cases} 0 & \text{if } \theta \in G \\ \infty & \text{if } \theta \notin G, \end{cases} \tag{2.29}$$

then the proximal operator for g is also the projection operation onto G. The basic proximal gradient algorithm produced iterates according to [138]:

$$\theta^{(k+1)} \leftarrow \operatorname{prox}_g \left(\theta^{(k)} - \alpha_k \nabla f \left(\theta^{(k)} \right) \right). \tag{2.30}$$

Further, the gradient in (2.30) can be replaced with a stochastic sample of the gradient, or an accelerated version [138], and so forth to derive a class of proximal optimization algorithms.

One extension of the stochastic gradient descent method, which will be a focus of Chapter 4, is *distributed* SGD. It is possible to reduce the time to compute a solution by tasking multiple processors to compute and optimize each with different gradient samples. Zinkevich et al. [109], [210] consider the typical case of SGD being used to minimize the empirical loss of a Machine Learning problem. The training dataset is distributed among the workers, with each machine being given a random subset of the examples. The workers can then compute gradients and updates to the parameter choices, in some presented algorithms even running full gradient optimizations. One of the main performance limitations that must be taken into account is the time and communication cost required to synchronize and send parameter updates between workers. The work in [139] showed that, for some problems, this may not be necessary, and workers can update the decision variables in an arbitrary asynchronous way.

Another optimization method, called *coordinate descent* or block coordinate descent, is closely related to stochastic gradient descent. In block coordinate descent methods, instead of sampling a subset of terms of the objective function, one selects a subset of the decision variables at each iteration. To illustrate the method, I'll describe the ordinary coordinate descent method that uses "blocks" of a single coordinate; the more general case does a similar procedure on multiple coordinates in a single step. Given a basic unconstrained minimization problem $\min_{x \in \mathbb{R}^d} f(x)$, and a starting point $x^{(0)}$, the procedure is to repeatedly:

1. select some $i_k \in \{1, ..., d\}$, and
2. update $x^{(k)} \leftarrow x^{(k-1)} - \delta_k \frac{\partial f}{\partial x_{i_k}}(x^{(k-1)}) e_{i_k}$.

In the above notation, $\frac{\partial f}{\partial x_{i_k}}(x)$ is the partial derivative of f w.r.t. the i_k^{th} coordinate, and e_{i_k} is a basis vector with the i_k^{th} element equal to 1 and the remaining elements equal to 0.

The choice of blocks can be either chosen by some algorithmic strategy, such as choosing a block that has a large gradient, or by randomly choosing blocks. There are a number of ways of choosing the coordinates with strong convergence guarantees [118], [159]. Even a randomized strategy can show very rapid convergence on some problems [152], [167].

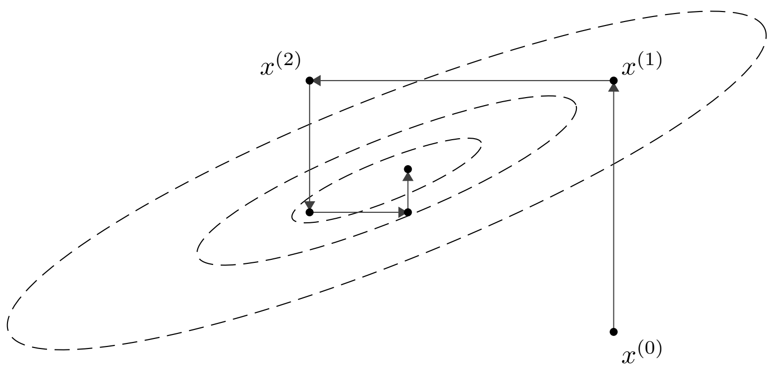


Figure 2.3: Illustration of the coordinate descent method on a function in $\mathbb{R}^2$. Each step alternates between the two input dimensions.

Much like stochastic gradient descent, block coordinate descent is especially suited for optimization in distributed settings. Individual workers can each optimize on disjoint subsets of the optimization variables, or even overlapping subsets [122], [139]. This significantly reduces the amount of information that must be communicated between different optimization workers. Since this communication cost is frequently the bottleneck on distributed computing, block coordinate methods can scale to much larger instances of the optimization problems. Block coordinate descent can further simplify the computation of the descent direction and enjoys some of the same benefits of SGD in avoiding the computation of expensive objective functions [136].

2.3 Optimization on Manifolds

A manifold is a space such that, for each element of the manifold, there exists a homeomorphism from the neighborhood around that point to a subset of the Euclidean space $\mathbf{R}^d$ for some d. A manifold $\mathcal{M}$ is differentiable if the mappings from $\mathbb{R}^d$ to $\mathcal{M}$ are differentiable [44].

At each point on a differentiable manifold, one can define a *tangent space*. For any differentiable $\alpha : (-\epsilon, \epsilon) \to \mathcal{M}$, one can define a tangent vector at $p = \alpha(0)$ to be the velocity of α at 0. The set of all such vectors at $p \in \mathcal{M}$ is the tangent space $T_p\mathcal{M}$ [44]. Further, a *Riemannian manifold* is one where the tangent spaces of the manifold are equipped with some inner product. This inner product acts on pairs of tangent vectors, and defines the Riemannian metric on that manifold.

Geodesic curves are a common building block of optimization algorithms that act on Riemannian manifolds. A curve on a manifold can be parameterized as a mapping from some interval to the manifold $\gamma : \mathbb{R} \supset [a, b] \to \mathcal{M}$. If the curve produced by γ is the shortest curve between $\gamma(a)$ and $\gamma(b)$, as measured by the Riemannian metric on $\mathcal{M}$, then it is a geodesic. If, for any $t_0, t_1 \in [a, b]$, the arc length of γ between $\gamma(t_0)$ and $\gamma(t_1)$ is equal to $t_1 - t_0$, then γ has a constant speed of 1 [44].

Building on geodesic curves, one can further define the *exponential map*. Given $p \in \mathcal{M}$ and $v \in T_p\mathcal{M}$, this is defined as:

$$\exp_p(v) = \gamma_{p,v}(1), \tag{2.31}$$

where $\gamma_{p,v}$ is a unit-speed parameterization of a geodesic such that $\gamma(0) = p$ and $\gamma'(0) = v$. This geodesic is unique. Importantly for optimization, the exponential map allows us to define a geodesic curve at a given point in a given direction, for instance the descent direction of an objective function at the current solution [2]. The *log map* is then the inverse of the exponential map. Given $p, q \in \mathcal{M}$, $\log_p(q)$ will be a vector in $T_p\mathcal{M}$ such that $\exp_p(\log_p(q)) = q$. Basically, this gives the direction on the manifold at p that points toward q, with magnitude equal to the geodesic distance between them.

Examples of manifolds commonly used in Computer Vision include the rotation groups [176], the Grassmannian and Stiefel manifolds [47], and the positive semi-definite matrices [69]. The manifolds on orthogonal and orthonormal matrices are of special interest [47]. The most basic definition among these is the orthogonal group. This consists of the set of matrices

$$O_n = \left\{ X \in \mathbb{R}^{n \times n} \mid X^T X = I_n \right\} \tag{2.32}$$

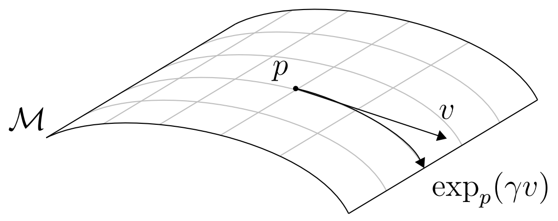


Figure 2.4: Illustration of a manifold $\mathcal{M}$, a tangent vector $v \in T_p\mathcal{M}$, and the geodesic generated by $\exp_p(\gamma v)$ for $\gamma \geq 0$.

where I_d is the $d \times d$ identity matrix. On rectangular matrices, there is the related *Stiefel manifold*

$$V_{n,p} = \left\{ X \in \mathbb{R}^{n \times p} \mid X^T X = I_p \right\}. \tag{2.33}$$

Matrices in the Stiefel manifold will have columns that are n-vectors of unit length that are all mutually orthogonal. This means that these columns are a basis of a p-dimensional subspace of $\mathbb{R}^n$. These subspaces themselves form the Grassmannian manifold $G_{n,p}$. This means the Grassmannian manifold can be represented by the quotient group $V_{n,p}/O_p$, reducing all bases that can be mapped to each other by a p-dimensional rotation within the subspace to the same element of the Grassmannian. The discussion on matrix manifolds is extended in Chapter 4.

A *manifold optimization* is a problem of the form:

$$\min_{\mathbf{x} \in \mathcal{M}} f(\mathbf{x}) \tag{2.34}$$

where $\mathcal{M}$ is a Riemannian manifold. This is related to ordinary constrained optimization on vector spaces. Take a manifold embedded in $\mathbb{R}^d$, that is defined by equality constraints. Given a definition of the form

$$\mathcal{M} = \{\mathbf{x} \in \mathbb{R}^d \ : \ g(\mathbf{x}) = 0\} \tag{2.35}$$

for some real-valued function g. Then the problem in (2.34) is equivalent to

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^d} \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & g(\mathbf{x}) = 0. \end{aligned} \tag{2.36}$$

However, one can exploit the structure of $\mathcal{M}$ within the optimization to design algorithms distinct from ordinary constrained optimization. These are, specifically, what fall into the category of manifold optimization methods.

There are roughly two closely related categories of manifold optimization methods, which, in this thesis, I will refer to as projected and feasible descent methods. These differ in whether iterates or candidates are computed that lie in a space in which the manifold is embedded, but do not lie in the manifold.

In a *projected* method, or more generally one using *retractions* [3], one considers a manifold $\mathcal{M}$ embedded in some Euclidean space $\mathbb{R}^d \supset \mathcal{M}$, and takes the objective function f to have domain $\mathbb{R}^d$. Given an iterate $\mathbf{x}^{(k)} \in \mathcal{M}$, we can produce a candidate for the next iterate by moving along the gradient of f in the $\mathbb{R}^d$. In order to produce a solution, we must at some point then project back onto $\mathcal{M}$.

Feasible descent, by contrast, does not require the choice or construction of a space in which to embed the manifold [195]. Instead, one constructs a *descent curve.* Given an objective function on the manifold $f : \mathcal{M} \to \mathbb{R}$, this is some curve α on the manifold such that $\alpha(0) = \mathbf{x}^{(k)}$ and $(f \circ \alpha)'(0) < 0$. One can then decrease the objective by moving along the descent curve for some distance, and this can be repeated until a solution is found.

Manifold optimization problems are well-studied in Computer Vision and Machine Learning, with software toolboxes such as Manopt [16] and Pymanopt [96] available and targeting these use cases. Eynard et al. [50] optimize a joint diagonalization problem over the Stiefel manifold to do face pose alignment and image clustering. Manifold optimization can also be applied to the PCA problem, to produce regularized reduced-dimensionality representations of examples [65], [184]. Ring and Wirth [153] demonstrate a segmentation method in the style of curve evolution, but instead with optimization on a manifold over shapes.

2.4 Sparse Matrices

One of the best ways to make optimization implementations more efficient is to leverage *sparsity*. A vector or matrix is sparse if a large portion of its elements are zero. Many common linear algebra operations can be performed on sparse arguments with computational cost proportional not with the size of the vector or matrix, but rather with the count of nonzeros. In cases where a large amount of redundant work is being done, an easy way to model the problem so that this work is skipped is to use sparse operations.

As a simple example, consider the dot product between two vectors in $\mathbb{R}^d$:

```
Require: $u, v \in \mathbb{R}^d$
  $p \leftarrow 0$
  for $i = 1, ..., d$ do
    $p \leftarrow p + u_i v_i$
  end for
  return $p$.
```

All elements of each vector will typically be stored consecutively in memory. Computing and accumulating the products of each element will be $O(d)$ operations.

Suppose instead that u and v are sparse. Take $\text{nnz}(u)$ to be the number of non-zero elements in u, and the same for v. One efficient way to store the vector u is to have two arrays: an array IU of the indices of these non-zero elements in ascending order, and an array of the non-zero elements themselves. The dot product of these sparse vectors could be computed instead with:

```
Require: $u, v \in \mathbb{R}^d$ and $IU, IV \subseteq \{1, ..., d\}$
  $p \leftarrow 0$
  $j \leftarrow 1$
  for $i = 1, ..., \text{nnz}(u)$ do
    while $\text{IU}_i > \text{IV}_j$ do
      $j \leftarrow j + 1$
```

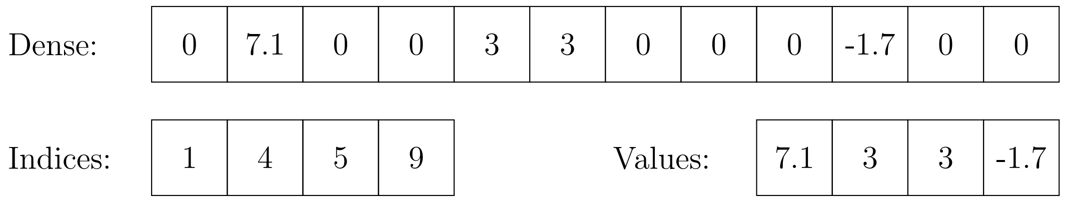


Figure 2.5: A sparse vector can be stored as a dense array, including zero entries, as shown in the top array. Alternately, one can store the non-zero indices and values separately, as in the lower two arrays.

```
            **if** $j > \text{nnz}(v)$ **then**
                **return** $p$
            **end if**
        **end while**
        **if** $\text{IU}_i = \text{IV}_j$ **then**
            $p \leftarrow p + u_{\text{IU}_i} v_{\text{IV}_j}$
        **end if**
    **end for**
    **return** $p$.
```

This will instead require $O(\max(\text{nnz}(u), \text{nnz}(v)))$ operations. Typically, a vector $u \in \mathbb{R}^d$ will be considered "sparse" only when $\text{nnz}(u) \ll d$, so this can represent significant computational savings.

Work on sparse linear algebra typically deals with sparse *matrices*. One of the most popular storage formats is with Compressed Sparse Row (CSR) matrices. These store the rows of the matrix in a form similar to the sparse vector format described above and illustrated in Figure 2.5. Typically, the column indices and elements of the nonzero elements are all stored consecutively. The indices and elements for row i are then stored in a contiguous range, and an additional pointer array gives the position of this range, for each row i, in the overall order. CSR matrices are especially good for common operations such as matrix-vector multiplication. However, there are also a number of other formats. Important general-purpose formats include Compressed Sparse Column (CSC) and Coordinate (COO). There are also special data structures supported by common codes that can represent special structures of sparse matrices, such as banded or block diagonal matrices, or general block-sparse matrices. For details see [158] or further discussion in Chapter 5.

Modelling the performance of linear algebra operations by counting the number of operations, as in the example above, is incomplete. When the main constraint on performance is the number of floating-point operations that can be computed per second, this will tend to be an accurate way to compare the speed of different algorithms or implementations. However, on modern SIMD processors and GPUs, the "floating-point throughput" is high enough to no longer be the main factor in sparse computations. Instead, memory access and the bandwidth for the processing units to access the input and output data are the primary considerations [9]. These tend to be easy to optimize for in the case of dense matrices, where most operations access elements consecutively or in a regular pattern. For sparse operations, memory access may be to arbitrary locations as determined by the nonzero indices. This means that ordinary CSR can lead to suboptimal performance, but linear algebra computations can still be made efficient through the use of special storage formats [9] and in the design of the algorithms operating on them [8].

Chapter 3

Cosegmentation with Random Walker

Images frequently occur within groups in a variety of contexts. On the Web, images may be compiled into albums as for example in social media, or appear embedded into a document. When the same object appears in multiple images, this can be leveraged by Computer Vision inference to extract more information about that object by analyzing the images as a group rather than as a collection of isolated images. This is especially interesting for the segmentation task. Segmentation, a pixel-by-pixel label of the object or object instance to which that pixel belongs, is very difficult to do accurately on single images. It is possible to get better results in the context of a set of related images.

Cosegmentation seeks to segment a set of images that contain a common foreground object, or multiple such objects, and uses a shared model of this object across the entire image set to improve the accuracy of the segmentation. The following chapter presents work in which I demonstrate an optimization procedure for cosegmentation based on Random Walker Segmentation. The special structure of one form of this model leads to a highly parallelizable algorithm for finding the segmentations across the entire image set. This is further extended to a scale-free model for which one can efficiently get solutions of theoretically guaranteed quality despite the non-convexity of the resulting problem.

3.1 Related Work

3.1.1 Markov Random Fields

A classical formulation of the segmentation problem uses Markov Random Fields (MRFs) [19]. These are a class of probabilistic models that express the likelihood of possible pixel labellings over the image. The main tool provided by MRFs is the encoding of conditional dependence and independence between the probability distributions of each pixel's label, for instance the expectation that adjacent pixels are more likely to share the same label while distant pixels may be independent. Different models of object appearance and shape can then be included in the individual energy functions in the MRF.

I more specifically formulate object segmentation as an optimization that is equivalent to finding the most likely labelling under the MRF. Let Ω be the set of pixels in the image, or some other representation of the spatial domain of the image, and let $\mathcal{L}$ be the set of labels. The goal is a labelling $\mathbf{x} : \mathcal{V} \to \mathcal{L}$ that minimizes some measure of segmentation cost, or a segmentation "energy" function E^{MRF}. One can construct some graph $(\mathcal{V}, \mathcal{E})$ of the MRF dependencies where each pixel label has a vertex in the graph and $\mathcal{E}$ is a set of edges between pixels that are close enough to have dependent label distributions.

Then, the optimization problem is:

$$\min_{\mathbf{x} \in \mathcal{L}^{|\Omega|}} E^{\text{MRF}} = \sum_{i \in \mathcal{V}} p_i(x_i) + \sum_{(i,j) \in \mathcal{E}} w_{ij} \delta(x_i, x_j), \tag{3.1}$$

where p_i is a per-pixel label prior, w_{ij} is the pairwise similarity, and $\delta(\cdot, \cdot)$ is a label similarity such as the Potts function [19].

3.1.2 Markov Random Fields for Cosegmentation

The initial model for cosegmentation in [156] provided means for including global constraints to enforce consistency among the two foreground histograms in addition to the MRF segmentation terms for each image. The objective function incorporating these ideas was expressed as

$$E_{\text{coseg}} = E^{\text{MRF}}_{\text{image 1}} + E^{\text{MRF}}_{\text{image 2}} + \lambda E^{\text{global}}(h_1, h_2), \tag{3.2}$$

where $E^{\text{global}}(\cdot,\cdot)$ was assumed to penalize the ℓ_1 variation between each h_1 and h_2 (the two foreground histograms obtained *post* segmentation).

Histograms, and similar properties, of the foreground object provide a flexible and simple way to express the appearance of the object and include the similarity of its appearance between the images within the cosegmentation model. Given an appropriate choice of features on which to compute the histogram, the design of which is described in more detail below, this can be reasonably robust to changes in illumination, viewpoint, and deformation of the object. The histogram of only the foreground object, importantly, can also be expressed as a linear function of the segmentation labels. This is the key reason they are useful for building models that are useful within a fast cosegmentation optimization.

The appearance model for the histogram was assumed to be generative (i.e., Gaussian), and a novel scheme called *trust region graph cuts* was presented for optimizing the resulting form of (3.2). Subsequently, [129] argued in favor of using an ℓ_2^2-distance for $E^{\text{global}}(\cdot,\cdot)$, whereas [75] developed a reward-based model. A scale-free method is presented in [130, 131], which biases the segmentation towards histograms belonging to a low-rank subspace. Recently, [188] compared several existing MRF-based models, and presented a new posterior-maximizing scheme that was solved using dual decomposition. One reason for these varied strategies for the problem is that when a histogram difference term is added to the segmentation, the resultant objective is no longer *submodular* (therefore, not easy to optimize). Therefore, the focus has been on improved approximate algorithms for different choices of E^{global}.

3.1.3 Random Walker Segmentation

An expanded family of interactive segmentation formulations includes Random Walker segmentation [61]. Given a set of constrained seed points that have a manual or externally provided label, one can imagine placing a "walker" at one such seed. From there, the walker moves to an adjacent pixel with probability based on its similarity with the pixel at the walker's current position. This is then repeated for multiple hops. As the number

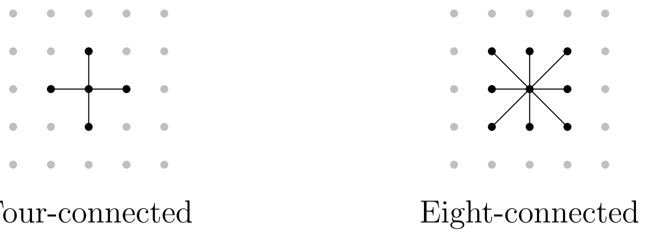


Figure 3.1: Illustration of neighborhoods of different connectivity, used to construct the graphs for MRF and Random Walker Segmentation. In the full graph, the highlighted neighborhoods will be repeated, centered on each grid vertex.

of iterations approaches infinity, the probability that the walker is at any given pixel will approach a steady-state distribution over the pixels. The probability for each pixel can then be used as a segmentation potential, and one will expect to see higher probabilities associated with pixels near and similar to a foreground seed, and lower probabilities for the opposite.

It can be shown that this steady-state probability is the solution to a combinatorial Dirichlet problem [61]. This can be found by solving a linear system constructed from a graph Laplacian. In equivalent optimization terms, as this problem is posed below, this linear system finds the point with zero gradient for a quadratic optimization objective.

Both Random Walker and MRF segmentation are included in a general family of segmentation optimization formulations called Power Watersheds [36]. This shows that the distinction between MRF and Random Walker segmentations is fundamentally a choice between the type of norm, the ℓ_1 and ℓ_2 norms respectively, used to penalize difference between labels between adjacent pixels and with their unary priors.

3.2 Random Walker for Cosegmentation

I begin by rewriting the Random Walker algorithm for a single image as a quadratic minimization problem. As is common, I assume a 4-connected neighborhood over the image, weighted according to a Gaussian function of normalized Euclidean distances between pixel intensities, $w_{ij} = \exp(-\beta \| p_i - p_j \|)$. Let L be the Laplacian matrix of this graph. A graph

Laplacian is an $n \times n$ matrix over a graph of n vertices, and can be constructed as:

$$L_{ij} = D - A = \begin{cases} \sum_{ik \in \mathcal{E}} w_{ij} & \text{if } i = j \\ -w_{ij} & \text{if } ij \in \mathcal{E} \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

The Laplacian is diagonally dominant, and so $L \succeq 0$, I can derive the convex quadratic program:

$$\begin{aligned} \min_x \quad & x^T L x \\ \text{subject to} \quad & x^{(s)} = S, \end{aligned} \tag{3.4}$$

where $x^{(s)}$ are the label values for certain seed pixels, and S is the known value of those seeds (i.e., foreground or background). Each component of the solution x^* will then be a pixel's probability of being assigned to the foreground. To output a $\{0, 1\}$ segmentation, one may threshold x^* at $\frac{1}{2}$, to obtain a hard $x \in \{0, 1\}^n$ segmentation that matches the solution from [61].

3.2.1 Pre-processing

Cosegmentation methods [75] use a pre-processing procedure to determine inter- (and intra-) image pixel similarity. This is generated by tessellating the RGB color space (i.e., pixel distribution) into clusters or by using SIFT—or color pattern models, edge profiles, textures, etc.—based correspondence methods, see [37]. One can derive a matrix H_i for each image i, constructed as:

$$H_i^{kj} = \begin{cases} 1 & \text{if pixel } j \text{ (of image } i\text{) is in histogram bin } k \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

Pixels are assigned to the same bin if they are similar. With an appropriate H, the global term E^{global} from (3.2) requires that at the level of individual histogram bins k, the algorithm assigns approximately the same number of pixels to each foreground region (the objective incurs a penalty proportional to this difference). This ensures that the appearance models of the two foregrounds based on the features of interest are similar.

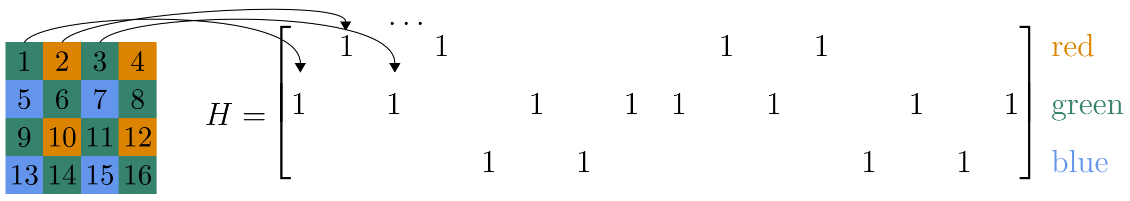


Figure 3.2: Construction of the histogram assignment matrix. This is a linear operator that maps a segmentation labelling to a histogram of the foreground object.

3.2.2 Cosegmentation for Two or More Images

Let $L_1, \cdots, L_m$ be the Laplacian matrices of graphs constructed using each of the images, and $H_1, \cdots, H_m$ be the histogram assignment matrices as above, with the property that $H_i^{kj} = H_{i'}^{kj'} = 1$ for pixels j and j' if and only if j and j' are similar. Given a segmentation for n pixels, $x \in \{0,1\}^n$, one may use the H matrix to write the histogram of only the foreground pixels as $h = Hx$. Now, I seek to segment two or more images simultaneously, under the constraint that their histograms match. For this purpose, consider the optimization model

$$\begin{aligned} \min_{x_i, h_i, \bar{h}} \quad & \sum_i x_i^T L_i x_i + \lambda \|h_i - \bar{h}\|_2^2 \\ \text{s.t.} \quad & x_i \in [0,1]^{n_i} \\ & x_i^{(s_i)} = S_i \qquad\qquad i = 1...m. \\ & H_i x_i = h_i \end{aligned} \tag{3.6}$$

The second term in the objective above corresponds to $E^{\text{global}}(h_1, h_2)$ in (3.2), and the last constraint sets up the foreground histograms h_i. They are defined using H_i. Bin k in H_1 corresponds to bin k in $H_2, \cdots, H_m$, which makes a direct comparison between histograms possible. The objective compares the histograms to a common *global* histogram $\bar{h}$ that at the optimum will be the centroid of the h_i's.

This model additionally extends to multiple labels by adding additional columns to the optimization variables identically to the original Random Walker [61]. The resulting problem can be easily decomposed into separate segmentations for each object class.

Theorem 3.1. *For $\lambda \geq 0$, the Random Walker Cosegmentation in (3.6) is a convex optimization problem.*

Proof. Each Laplacian matrix L_i is diagonally dominant by construction, and therefore positive semidefinite. For any positive semidefinite matrix L_i, the quadratic form

$$x_i^T L_i x_i \tag{3.7}$$

will be a convex function of x_i.

The squared L_2 norm is a convex function, and the difference $h_i - \bar{h}$ is a linear function. The composition of these $\|h_i - \bar{h}\|_2^2$ is convex, and so is the product with a non-negative λ. With each term in the objective being a convex function of the decision variables, the sum will be, as well.

Further, the feasible region is the intersection of bound constraints and linear equalities. Each individual constraint will define half-space or subspace respectively, and since these are convex sets, the feasible region defined by their intersection will be, as well.

Combined, we therefore have that (3.6) has a convex objective function and a convex set as a feasible region. □

3.2.3 Deriving an Equivalent Box-QP

The model in (3.6) can already be solved using widely available convex programming methods, and provides the desired solution to the cosegmentation problem using the Random Walker segmentation function. Next, I will derive an equivalent model that will allow the design of specialized solvers. This model will have a structure that allows the lowest-level steps of the optimization to be done independently on each decision variable coordinate, providing an opportunity for fine-grained parallelism.

Consider the left-hand side of the equality constraint on each h_i substituted into the objective function. Further, let us choose bounds to limit x to the unit box as well as suitably enforce the seed constraints. This process gives a quadratic problem of the form

$$\min_{x_1,x_2,\dots,x_m,\bar{h}} \begin{bmatrix} x_1 \\ \vdots \\ x_m \\ \bar{h} \end{bmatrix}^T \begin{bmatrix} L_1 + \lambda H_1^T H_1 & & & -\lambda H_1 \\ & \ddots & & \vdots \\ & & L_m + \lambda H_m^T H_m & -\lambda H_m \\ -\lambda H_1^T & \dots & -\lambda H_m^T & \lambda m I \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_m \\ \bar{h} \end{bmatrix} \tag{3.8}$$
$$\text{s.t.} \quad l_i \le x_i \le u_i \quad x_i \text{ is of size } [0,1]^{n_i} \quad i = 1, 2,$$

where the 2-tuple (l_i, u_i) is $(1,1)$ for foreground seeds, $(0,0)$ for background seeds, and $(0,1)$ otherwise. The objective here is a quadratic function over a vector that is the concatenation of all the pixels in all of the input images, along with the common model histogram $\bar{h}$.

It can be verified that (3.8) is equivalent to (3.6). The difference is that it is now expressed as a bound-constrained quadratic problem, or *Box-QP*, due to the box constraints [24]. Like (3.6), the model in (3.8) also permits general purpose convex programming methods. This can be somewhat inefficient to pose as a quadratic problem in many otherwise suitable optimization methods, though, as the blocks are very large and possibly very dense matrices. A method that requires explicitly constructing the objective matrix in (3.8) can rapidly become impractical due to memory constraints as image dimensions or the number of images grows. However, one can design means to exploit its special structure since the model is a quadratic problem with very simple constraints.

3.3 Optimization

This section describes our strategy for solving (3.8) to optimality in a highly efficient manner. I use a projected gradient-based method that would additionally form the key component if one were to use any variation, e.g. augmented Langrangian, as stated explicitly in [140].

3.3.1 Identifying a Sparse Structure:

Expressing our model as a purely bound-constrained problem as in (3.8) requires the formation of the $H_i^T H_i$ products, which are dense $n \times n$ matrices. Consequently, our optimization method must be careful not to explicitly form these matrices. Fortunately, one can observe that explicit calculation of these matrices may be avoided by gradient projection methods, in which it is only necessary to be able to calculate matrix-vector products. Here, this product can be distributed over the sparse components as

$$(L + H_1^T H_1)x_1 = Lx_1 + H_1^T(H_1 x_1). \tag{3.9}$$

With this modification, one can solve this Box-QP in (3.8) by adapting the Gradient Projection/Conjugate Gradient (GPCG) algorithm of [127]. I describe this strategy next.

3.3.2 Gradient Projection/Conjugate Gradient Method (GPCG)

GPCG solves quadratic problems with a rectilinear feasible region, one that can be defined as:

$$\Omega = \{x : l \leq x \leq u\} \subseteq \mathbb{R}^n. \tag{3.10}$$

The algorithm alternates between two main "phases:" Gradient Projection (GP) and Conjugate Gradient (CG). These correspond to alternately estimating the active set at the minimum and finding the minimum while keeping the active set fixed.

The gradient projection phase searches along a projected gradient. Given an objective function $\mathcal{O} : \mathbb{R}^n \to \mathbb{R}$, one can take a gradient at any point in Ω, but the gradient at a point on the boundary $x \in \bar{\Omega}$ may point in a direction such that

$$x - \epsilon \nabla \mathcal{O}(x) \notin \Omega \tag{3.11}$$

for any positive choice of ϵ. To take a descent direction that points within the feasible region, one instead uses a projected gradient $\nabla_\Omega \mathcal{O}(x)$ that has been modified so the corresponding search direction $d = -\nabla_\Omega \mathcal{O}$ does not point outside the feasible region Ω. It then uses a

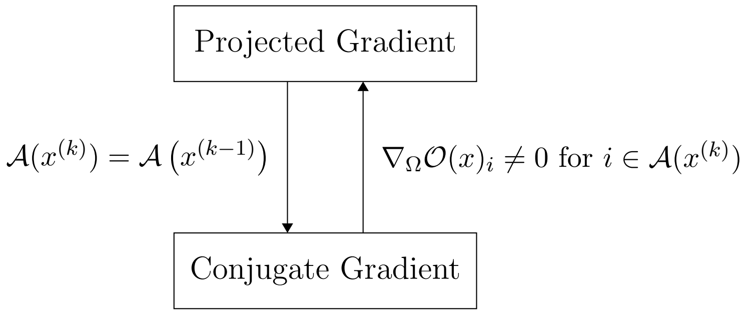

Projected Gradient
$\mathcal{A}(x^{(k)}) = \mathcal{A}\left(x^{(k-1)}\right)$
$\nabla_\Omega \mathcal{O}(x)_i \neq 0$ for $i \in \mathcal{A}(x^{(k)})$
Conjugate Gradient

Figure 3.3: Phases in the Gradient Projection/Conjugate Gradient (GPCG) algorithm of [127].

projected line search to arrive at a step length α and update $x \leftarrow P(x + \alpha d)$, where P describes the projection function to Ω.

The phase decisions in GPCG are made based on the active set: those bound constraints that hold with equality:

$$\mathcal{A}(x) := \{i : x_i = l_i \text{ or } x_i = u_i\}. \tag{3.12}$$

GPCG switches to the conjugate gradient phase if the active set remains unchanged between subsequent iterations on x.

GPCG searches within a given face of the feasible region of our model using the conjugate gradient phase. Given the active set, GPCG calculates a search direction conjugate to the previous direction (under the projection onto the free variables). This method of generating a search direction is the same as applying ordinary conjugate gradient descent to a restriction of the QP to the current minimal face. In the following, note that this face of the feasible polytope will correspond to a given choice of the active set. If the projected gradient points out of the current face, then it switches back to the gradient projection (GP) phase. Note that the "phase switch" condition will never be satisfied for the face that contains the global minimum for our model. Thus, when the gradient projection phase has found the correct active set, conjugate gradient iterations suffice to explore the final face.

Projected Line Search. The projected line search in the gradient phases modifies the active set by an arbitrary number of elements, thereby lending GPCG its key advantage that it may converge on the correct active set in far fewer iterations than the number of constraints. Given a starting point x and search direction d, the line search finds $\alpha > 0$ that produces a *sufficient decrease* under the Armijo rule of the function $\phi(\alpha) = \mathcal{O}\left(P[x + \alpha d]\right)$. This can be thought of as a "line search" along a 1-manifold that bends to stay within Ω (thus, not a ray). Rather than directly finding all the piecewise quadratic segments of $\phi(\alpha)$, one can efficiently produce a sufficient decrease using an estimate of ϕ by sampling one point at a time (as in [127]). It can be verified that *all* operations above can be expressed as Level 1 and 2 BLAS operations. This allows a highly parallel implementation, described next.

GPU Implementation. Graphical Processing Units (GPUs) have gained attention from across the scientific computing community for their ability to efficiently address parallel problems [112]. These architectures operate by running multiple instances of a piece of code simultaneously, operating on different parts of a dataset. While this approach is not well-suited to all algorithms, Level 1 and 2 BLAS operations used in our algorithm are known to fit well with this architecture and can therefore exhibit a significant speedup. The linear algebra operations comprising our GPCG algorithm for cosegmentation may be easily parallelized using high-level languages such as CUDA. In fact, the CUSPARSE toolkit (used here), supports Level 2 and 3 BLAS operations on sparse matrices, as well. Further, the control flow of our procedure relies only on the results of *accumulations*, so the standard bottleneck of transferring data between main and GPU memory is not a major factor, and entry-level hardware is sufficient.

3.4 Scale-Free Cosegmentation

A limitation of early cosegmentation methods is their sensitivity to the scale of the target object, since histogram-based priors are dependent on scale. For example, if an otherwise identical object appears in the second image such that it occupies twice as many pixels as in

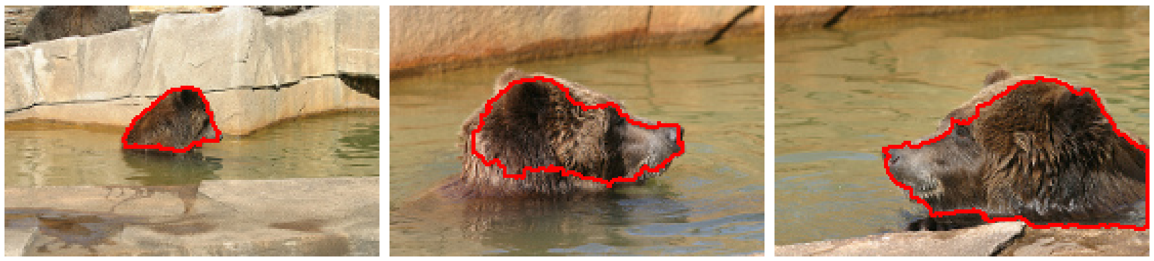


Figure 3.4: Segmentation using the model of Section 3.4 on a set of images with differences in scale.

the first image, then $h_2 = 2h_1$. Consequently, $||h_2 - h_1|| > 0$, meaning that the larger scale is penalized. I show here how this formulation may be made scale-invariant. Formally, our goal is a cosegmentation term E that satisfies

$$E(sh_i, \bar{h}) = E(h_i, \bar{h}) \quad \forall s \in \mathbb{R}_+. \tag{3.13}$$

This property may be satisfied by a normalization step,

$$E(h_i, \bar{h}) = \left\| \frac{h_i}{\|h_i\|} - \frac{\bar{h}}{\|\bar{h}\|} \right\|^2 = 2 - 2\frac{h_i^T \bar{h}}{\|h_i\| \|\bar{h}\|}. \tag{3.14}$$

For any given choice of λ, the optimal choice of h_i and $\bar{h}$ that minimizes the cosegmentation objective with (3.14) can also be found by solving an optimization, possibly with a different choice of λ, that substitutes it with:

$$-\frac{h_i^T \bar{h}}{\|h_i\| \|\bar{h}\|} = -\cos(\angle h_i \bar{h}). \tag{3.15}$$

This substitution in (3.6) leads to the objective function:

$$\sum_i x_i^T L x_i - \lambda \frac{h_i^T \bar{h}}{\|h_i\| \|\bar{h}\|}. \tag{3.16}$$

This function is not easy to minimize, as the histogram term is non-convex in h_i and $\bar{h}$. However, in the Random Walker setting, one *can* optimize this function when the model histogram $\bar{h}$ is a fixed unit vector, eliminating the $\|\bar{h}\|$ factor. The resulting problem is related to model-based segmentation, imposing a known histogram distribution in segmenting images. For image i it solves the problem

$$\begin{aligned} \min_{x_i} \quad & x_i^T L_i x_i - \lambda \frac{\bar{h}^T H x_i}{\|H x_i\|} \\ \text{s.t.} \quad & l_i \le x_i \le u_i, \end{aligned} \tag{3.17}$$

where I name $g(x) = \frac{\bar{h}^T H x_i}{\|H x_i\|}$.

In order to proceed with the minimization of our scale-invariant energy, I must first establish some properties of g. The scale-free energy histogram on x, given by $g(x)$, is quasiconvex, defined as follows:

Definition 3.2 ([7]). *Define a function f to be* quasiconvex *if its sublevel sets are convex subsets of the domain. Equivalently, for any x, x' in the domain of f and $\lambda \in [0,1]$,*

$$f((1-\lambda)x + \lambda x') \le \max\{f(x), f(x')\}. \tag{3.18}$$

Call (3.18) *the "Jensen's Inequality for Quasiconvexity."*

Additionally, $g(x)$ is Lipschitz-smooth when $\|Hx\| > 0$, shown below in Theorem 3.5. The next section exploits these properties of g to solve the segmentation problem using this penalty. Classical global optimization schemes rely on the convexity of the problem, and proofs of the correctness of their solution and bounds on convergence tend to rely on this property. For problems that are not convex, a looser condition than convexity, called quasiconvexity, can also be used to show guarantees that appropriate optimization algorithms will yield a global optimum. In order to justify the use of one such algorithm, I first present the detailed proof of the quasiconvexity of the scale-free problem.

Define the function in terms of the image foreground histograms as:

$$\hat{g}_{\bar{h}} = -\frac{\bar{h}^T h}{\|h\|} \tag{3.19}$$

Then we have the result:

Theorem 3.3. *$\hat{g}_{\bar{h}}$ as defined in* (3.19) *is quasiconvex.*

Proof. Consider any h_1, h_2 w.l.o.g. chosen to satisfy

$$\hat{g}_{\bar{h}}(h_2) \leq \hat{g}_{\bar{h}}(h_1),$$
$$\frac{\bar{h}^T h_2}{\|h_2\|} \geq \frac{\bar{h}^T h_1}{\|h_1\|},$$

multiply by $(1-\lambda)\|h_1\|\|h_2\| \geq 0$

$$(1-\lambda)\bar{h}^T h_2\|h_1\| \geq (1-\lambda)\bar{h}^T h_1\|h_2\|,$$

add $\lambda\bar{h}^T h_1\|h_1\|$,

$$\begin{aligned}\lambda\bar{h}^T h_1\|h_1\| + (1-\lambda)\bar{h}^T h_2\|h_1\| &\geq \lambda\bar{h}^T h_1\|h_1\| + (1-\lambda)\bar{h}^T h_1\|h_2\| \\ \left(\lambda\bar{h}^T h_1 + (1-\lambda)\bar{h}^T h_2\right)\|h_1\| &\geq \bar{h}^T h_1\left(\lambda\|h_1\| + (1-\lambda)\|h_2\|\right)\end{aligned}$$

taking any $\lambda \in [0,1]$.

Since all these vectors are in the nonnegative orthant, $\bar{h}^T h_1, \bar{h}^T h_2 \geq 0$, and $-\hat{g}_{\bar{h}}(h_2) \geq -\hat{g}_{\bar{h}}(h_1)$, this inequality is equivalent to:

$$\frac{\lambda\bar{h}^T h_1 + (1-\lambda)\bar{h}^T h_2}{\lambda\|h_1\| + (1-\lambda)\|h_2\|} \geq \frac{\bar{h}^T h_1}{\|h_1\|} \tag{3.20}$$

Using this expression with the triangle inequality to show Jensen's inequality for quasiconvexity (3.18) gives

$$\begin{aligned}\hat{g}_{\bar{h}}(\lambda h_1 + (1-\lambda)h_2) &= -\frac{\lambda\bar{h}^T h_1 + (1-\lambda)\bar{h}^T h_2}{\|\lambda h_1 + (1-\lambda)h_2\|} \\ &\leq -\frac{\lambda\bar{h}^T h_1 + (1-\lambda)\bar{h}^T h_2}{\lambda\|h_1\| + (1-\lambda)\|h_2\|}.\end{aligned}$$

Using (3.20),

$$\begin{aligned}&\leq -\frac{\bar{h}^T h_1}{\|h_1\|} \\ &= \hat{g}_{\bar{h}}(h_1) = \max\left\{\hat{g}_{\bar{h}}(h_1), \hat{g}_{\bar{h}}(h_2)\right\}.\end{aligned}$$

The final step derives from the w.l.o.g. choice made at the beginning of this proof. □

Corollary 3.4. *The scale-free energy on x, $g_{\bar{h}}(x) = \hat{g}_{\bar{h}}(Hx)$ is quasiconvex for histogram assignment matrix H.*

Proof. This G represents the outer composition of E with an affine function. This operation preserves quasiconvexity [18]. □

Theorem 3.5. *$g_{\bar{h}}(x)$ is Lipschitz-smooth when $\|Hx\| > 0$.*

Proof. Let $\hat{\cdot} = \frac{\cdot}{\|\cdot\|}$. Assume w.l.o.g. $\|h_1\| \geq \|h_2\|$. Then:

$$\begin{aligned}
\|h_1 - h_2\| &\geq \left\| \frac{\|h_2\|}{\|h_1\|} h_1 - h_2 \right\| \\
&= \|h_2\| \times \|\hat{h}_1 - \hat{h}_2\| \\
\frac{1}{\|h_2\|} \|h_1 - h_2\| &\geq \|\hat{h}_1 - \hat{h}_2\|.
\end{aligned} \tag{3.21}$$

Thus, for any function f which is L-smooth,

$$\begin{aligned}
\|f(\hat{h}_1) - f(\hat{h}_2)\| &\leq L \|\hat{h}_1 - \hat{h}_2\| \\
&\leq \frac{L}{\|h_2\|} \|h_1 - h_2\|.
\end{aligned} \tag{3.22}$$

Further, if we lower-bound $\|h_1\| \geq \|h_2\| \geq C > 0$, then $f(\hat{\cdot})$ is $\frac{L}{C}$-smooth.

In our case, $g_{\bar{h}}(h)$ is an affine function of $\hat{h}$, and any affine function will be Lipschitz. □

3.4.1 Optimizing the Scale-Free Model

For the following, I consider the setting of minimizing $h = f + g$, such that f is convex, g is quasiconvex and both are bounded below. Note that under these conditions, h is not necessarily quasiconvex and may have multiple local minima. Nonetheless, our method proposed below can optimize our segmentation objective with $f(x) = x^T L x$ and g being used as defined above. Let $x_f^* = \operatorname{argmin}_x f(x)$ and define x_g^* and x_h^* similarly.

Define

$$P(\alpha) = \begin{pmatrix} \underset{x \in X}{\operatorname{argmin}} & f(x) \\ \text{s.t.} & g(x) \leq \alpha \end{pmatrix} \tag{3.23}$$

If x is any solution to $P(g(x_h^*))$, then $h(x) = h(x_h^*)$. Also, $P(\alpha)$ has no solutions for $\alpha < g(x_g^*)$, and x is a solution to $P(g(x_f^*))$ if and only if x is a solution $\forall \alpha > g(x_f^*)$. Consequently, we have the result:

Lemma 3.6. *There exists some α in a bounded interval such that the solution to the subproblem $P(\alpha)$ from* (3.23) *provides a solution to the overall scale-free problem in* (3.17).

This definition reduces the optimization problem to finding the α that minimizes $h \circ P(\alpha)$. I claim that $h \circ P$ is one-sided Lipschitz:

Definition 3.7 ([56])**.** *A function $f : [a, b] \to \mathbb{R}$ is* one-sided Lipschitz *if for any $x_1, x_2 \in [a, b]$*

$$(x_1 - x_2)(f(x_1) - f(x_2)) \leq m(x_1 - x_2)^2 \tag{3.24}$$

for some m.

Intuitively, in the case that f is continuously differentiable, the Lipschitz condition bounds $|f'(x)|$ while this condition only guarantees an upper bound on $f'(x)$. This is illustrated in Figure 3.5.

Lemma 3.8. *Let $\circ$ denote the composition operator: $(g \circ P)(\alpha) = g(P(\alpha))$. I claim:*

$$(g \circ P)(\alpha) = \alpha \textit{ for any } \alpha \in \left[g(x_g^*),\ g(x_f^*)\right]. \tag{3.25}$$

Proof. Let $x = P(\alpha)$. x satisfies the KKT conditions for the $P(\alpha)$ problem and either:

Case 1) x is the unconstrained minimum of f (*i.e.* $0 \in \partial f(x)$), or

Case 2) x lies on the boundary of the feasible region (so the $g(x) \leq \alpha$ constraint is active and $-\partial f(x) \cap \partial g(x) \neq \emptyset$).[1]

[1]Here ∂f denotes the subdifferential of f.

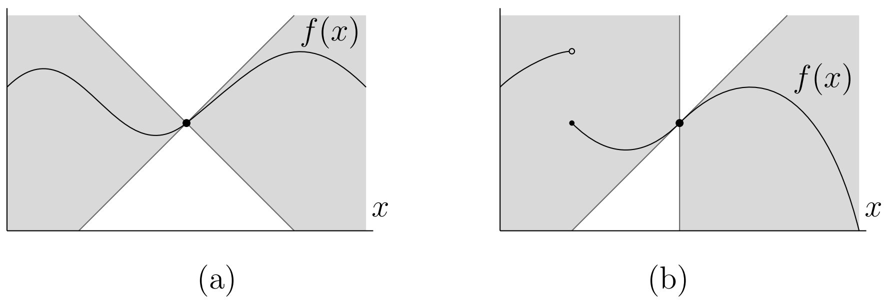


Figure 3.5: Given a pair $(x, f(x))$ for an L-Lipschitz function f, the Lipschitz condition guarantees that the function will always lie between two lines of slope L and $-L$ through this point and in the shaded region shown in (a). The *one-sided Lipschitz condition* in Definition 3.7 only bounds the rate of increase, and the function must lie in the shaded region shown in (b).

As previously used above, case 1 will only be true for $\alpha \geq g(x_f^*)$, with equality in the range I consider here. In case 2, we have the theorem simply from the fact that the constraint is active. □

Lemma 3.9. *$f \circ P$ is monotonically non-increasing.*

Proof. Take any $\alpha_1 < \alpha_2$. Let $x_1 = P(\alpha_1)$ and $x_2 = P(\alpha_2)$. Since $g(x_1) = \alpha_1 < \alpha_2$, we know x_1 is feasible for the $P(\alpha_2)$ problem. Thus, since x_2 is a minimizer for the $P(\alpha_2)$ problem, $f(x_1) \geq f(x_2)$. □

Theorem 3.10. *$h \circ P$ is one-sided Lipschitz.*

Proof. Take any $\alpha_1, \alpha_2 \in \left[g(x_g^*),\ g(x_f^*)\right]$.

Since $f \circ P$ is monotonically non-increasing,

$$(\alpha_1 - \alpha_2)\left((f \circ P)(\alpha_1) - (f \circ P)(\alpha_2)\right) < 0.$$

Let $m \geq 1$, and because $(\alpha_1 - \alpha_2)^2 \geq 0$,

$$\begin{aligned}(\alpha_1 - \alpha_2)\left((f \circ P)(\alpha_1) - (f \circ P)(\alpha_2)\right) &\leq (m-1)(\alpha_1 - \alpha_2)^2 \\ (\alpha_1 - \alpha_2)\left((f \circ P)(\alpha_1) - (f \circ P)(\alpha_2) + (\alpha_1 - \alpha_2)\right) &\leq m(\alpha_1 - \alpha_2)^2.\end{aligned}$$

Using Lemma 3.8,

$$(\alpha_1 - \alpha_2)\left((h \circ P)(\alpha_1) - (h \circ P)(\alpha_2)\right) \leq m(\alpha_1 - \alpha_2)^2$$

so that h is one-sided Lipschitz with constant m. □

As a result, by simply sampling $h(P(\alpha))$ densely enough, one can get an estimate of this function to arbitrary precision over the entire interval $\alpha \in \left[g(x_g^*),\ g(x_f^*)\right]$. If one selects a global minimum of this estimated function, this can derive a point with objective *arbitrarily* close to the true minimum of h. The bounds are given from the following theorem:

Theorem 3.11. *Take an increasing sequence $\alpha_1, ..., \alpha_n$ from the domain of $h \circ P$ with $\alpha_1 = g(x_g^*)$ and $\alpha_n = g(x_f^*)$. Let τ be the maximum gap $\alpha_{i+1} - \alpha_i$. Denote the minimum sample by α^*, chosen such that $(h \circ P)(\alpha^*) \leq (h \circ P)(\alpha_i)$ for all i. Then the function h has lower bound:*

$$h(x) \geq (h \circ P)(\alpha^*) - \tau \quad \forall x. \tag{3.26}$$

I start the proof of this theorem with the following lemma:

Lemma 3.12. *For any interval $[\alpha_1, \alpha_2] \subseteq \left[g(x_g^*),\ g(x_f^*)\right]$ and any $\alpha \in [\alpha_1, \alpha_2]$,*

$$\begin{aligned}(h \circ P)(\alpha) &\geq (h \circ P)(\alpha_2) - (\alpha_2 - \alpha) \\ &\geq (h \circ P)(\alpha_2) - (\alpha_2 - \alpha_1).\end{aligned} \tag{3.27}$$

Proof. Take $m = 1$ in Lemma 3.12, applied to this particular α, α_2:

$$(\alpha - \alpha_2)\left((h \circ P)(\alpha) - (h \circ P)(\alpha_2)\right) \leq (\alpha - \alpha_2)^2$$

by interval construction $\alpha - \alpha_2 \leq 0$, so

$$(h \circ P)(\alpha) - (h \circ P)(\alpha_2) \geq (\alpha - \alpha_2)$$

$$(h \circ P)(\alpha) \geq (h \circ P)(\alpha_2) - (\alpha_2 - \alpha).$$

□

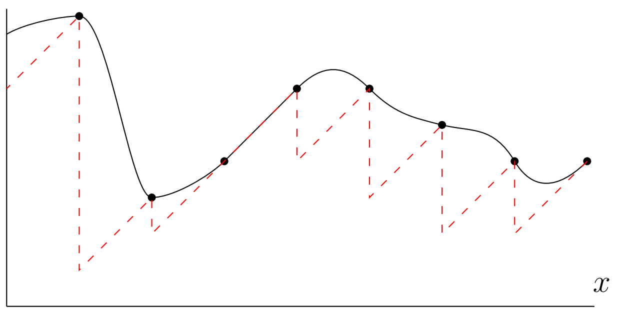


Figure 3.6: Illustration of the lower bound used in Lemma 3.12. If we only have the sampled points shown, we can nonetheless guarantee that a function which is one-sided Lipschitz will lie above the dashed lines.

Proof. Proof of Theorem 3.11

Since the minimum of h is guaranteed to lie in the co-domain of P,

$$(h \circ P)(\alpha) \geq (h \circ P)(\alpha^*) - \tau \quad \forall \alpha. \tag{3.28}$$

Consider any α. By our choice of $\alpha_1, ..., \alpha_n$, there is an α_i, α_{i+1} such that $\alpha \in [\alpha_i, \alpha_{i+1}]$. By Lemma 3.12,

$$(h \circ P)(\alpha) \geq (h \circ P)(\alpha_i) - (\alpha_{i+1} - \alpha_i)$$

by the construction of α^*,

$$\geq (h \circ P)(\alpha^*) - (\alpha_{i+1} - \alpha_i),$$

and since $\tau \geq \alpha_{i+1} - \alpha_i$

$$\geq (h \circ P)(\alpha^*) - \tau.$$

□

In the scale-free cosegmentation setting, f is convex and g is quasiconvex. The resulting P problem can therefore be solved efficiently using ordinary methods. One can calculate x_f^* as the solution to the ordinary random-walker segmentation, and x_g^* is the projection of $\bar{h}$ onto the feasible set under seed constraints.

3.5 Histogram Construction

In this section I discuss the fact that a wide variety of *different* preprocessing steps can take advantage of the *same* core method to produce a cosegmentation result.

At the heart of cosegmentation is some notion of similarity between pixels. The key property of cosegmentation over independent cosegmentation is that the foregrounds should contain *similar* distributions of pixels. In the group of cosegmentation methods we consider here, I rely on *histograms*, in which similar pixels are placed in the same histogram bin and dissimilar pixels are placed in different histogram bins. This is done through a *histogram assignment matrix* H, as described in Section 3.2.1.

In order to apply a cosegmentation method to a pair of images, we must first construct this matrix as in (3.5). One aspect of the problem is to assign a *bin* to some subset (or all) of the pixels between the two images. This needs to be done in such a way that similar foregrounds have similar histograms, so that the distance between the histograms matches a common-sense idea of when two foregrounds are similar. This includes some invariance to lighting, rotation, etc.

The bin assignment problem can be divided into two steps:

1. Assign local descriptors [126] to each pixel independently.

2. Find sets of "similar" pixels.

See Table 3.1 for a summary of the ways we might consider performing each of these steps in a typical case.

3.5.1 Texture-Based Histograms

I empirically found that high-quality dense histograms were most consistently produced with the middle options in Table 3.1. I apply the bank of texture filters shown in Figure 3.7. The responses at each pixel can then be clustered using nonparametric methods such as mean-shift or the greedy clustering of [60] with modified stopping conditions.

Image Features	**Correspondences**
Color	Axis-aligned binning
Winn Filters [197]	Clustering (k-means, spectral, ...)
SIFT, DAISY, etc	High-confidence matches

Table 3.1: Table of options for each step of cosegmentation.

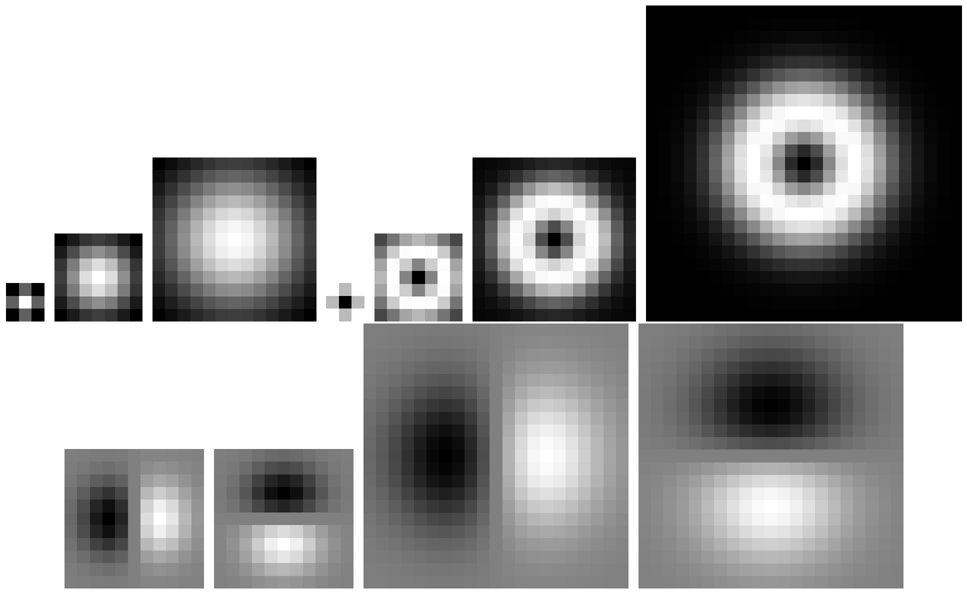
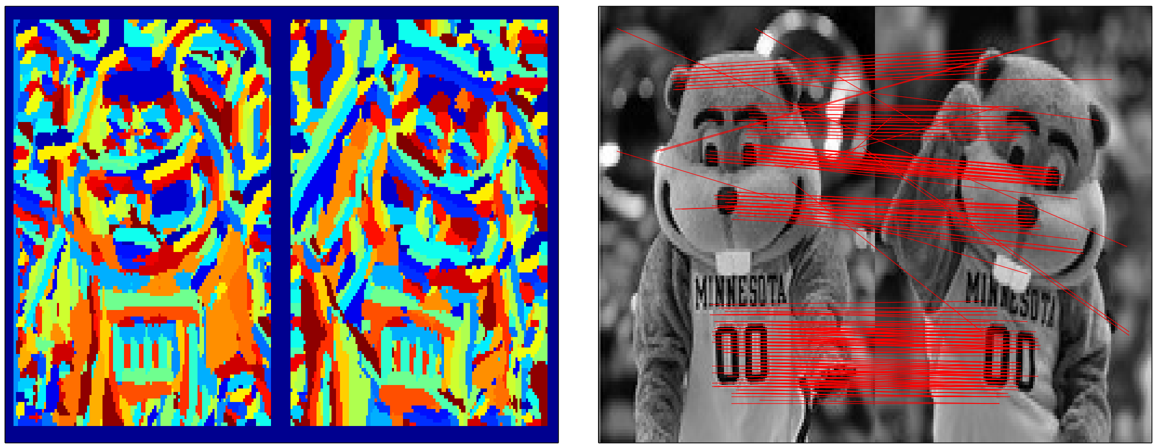


Figure 3.7: Winn filters from [197] used in our histogram generation. The first three blur filters are applied to each channel of the Lab colorspace, with the rest applied only to the lightness channel.



(a) (b)

Figure 3.8: Clustering (a) and matching (b) results generated using VLFeat [186].

The authors of [197] further present a method of training a texture dictionary from an incompletely labeled dataset. They present an agglomerative clustering algorithm based on maximizing a probability

$$\tilde{P}(\hat{\mathbf{c}}|\{\mathbf{H}_n\}) = \frac{P(\{\mathbf{H}_n\}|\hat{\mathbf{c}})}{P(\{\mathbf{H}_n\}|\hat{\mathbf{c}}) + P(\{\mathbf{H}_n\}|\mathbf{c}^{\text{same}})} \tag{3.29}$$

for histograms $\mathbf{H}_n$ and training labels $\hat{\mathbf{c}}$. This clustering can be leveraged in texture-based cosegmentation algorithms, producing a binning method optimized to handle differentiating between object classes in the training data.

SIFT-based. A popular class of descriptors are those based on gradient binning, including SIFT, GLOH, etc. The high dimension of these descriptors makes it relatively difficult to find high-quality dense *matches* between the feature vectors of different pixels. In this setting, which does not allow, for instance, assuming a homography, the matching was found to either be sparse or overly specific. Representative matching and clustering results are shown in Figure 3.8, which are reasonable but did not lead to better segmentation results.

3.5.2 Histograms from Optical Flow

The flexibility of our method with respect to histogram construction allows the use of *application-specific* preprocessing steps. There is a natural parallel between cosegmentation and the task of segmenting *video*. When the images are temporally related, we have a relationship between foreground pixels determined by the movement shown in the video. This is a well-studied problem in Computer Vision, allowing us a class of *optical flow* methods.

We can find corresponding pixels using optical flow, placing pixels i and i' in the same bin if the position of i maps to i' in the next image. Applying this scheme to the frames of a video sequence suggests the model

$$\min_{x \in X} \quad \sum_i x_i^T L_i x_i + \lambda \sum_i \|H_i x_i - H'_{i+1} x_{i+1}\|_2^2 \tag{3.30}$$

where the second sum compares subsequent frames of an image sequence.

3.5.3 Advantages of Low-Entropy Histograms

Finally, I provide some additional information on the core advantage of our formulation for cosegmentation.

I found while tuning the histogram construction that using low-entropy histograms allows for more accurate matching between images. Intuitively, take the example from Figure 3.4, where a high-entropy histogram may try to differentiate between different patches of fur on a brown bear. This cannot be done in a consistent manner across realistic images, so erroneous matches are introduced. By contrast, our histogram matching technique better describes the the true texture description of the bear's fur as a combinations of very few homogeneous textures. I verify this experimentally in Figure 3.9, which plots the statistical distance between the histograms of the true foregrounds. This is computed for a sample of images from the dataset.

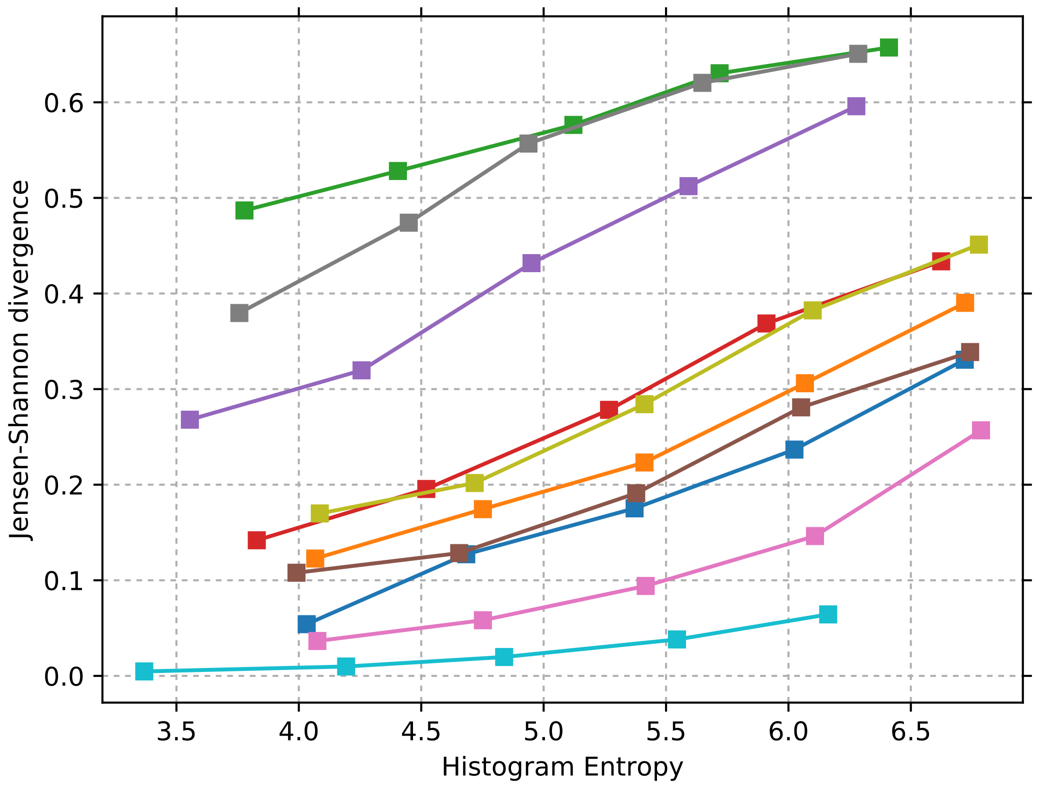


Figure 3.9: Comparing the entropy of the constructed histogram (x axis), with a symmetric KL-divergence between the true foreground histograms (y axis), as we vary the number of clusters. Each line shows one image pair. We consistently see higher-entropy histograms producing greater divergence for the target segmentation.

3.6 Experiments

Summary. My experimental evaluations included comparisons of my implementation (on a Nvidia Geforce 470) to another cosegmentation method [75] and the Random Walker algorithm [61] (run independently on both images). I also performed experiments using the methods in [88] and [129], but due to the problem of solving the linear system for a large L and incorporating foreground/background seeds respectively, results could not be obtained for the entire dataset described below. To assess *relative* advantages of the specialized GPCG procedure, I also compared it with a stand-alone implementation of (3.6) linked with a commercial QP solver (using multiple cores). This thesis provides a qualitative and quantitative overview of the performance w.r.t. state of the art. Additional experiments demonstrate the efficacy of the multiple-image and scale-free segmentation models.

Datasets. In order to leverage all available test data, I aggregated all images provided by the authors in iCoseg [5], Momi-coseg [32], and Pseudoflow-coseg [75], and further supplemented them with a few additional images from the web and [182]. In order to compare with algorithms that only handle image *pairs*, I selected a dataset with 50 pairs of images (100 images total). For the > 2 image case, I used the iCoseg dataset from [5]. Since a number of image sets from [5] share only semantically similar foreground objects and are unsuitable for cosegmentation with common appearance models, i selected 88 subsets comprising 389 of the 643 iCoseg images (also observed in [189]).

Winn Filters. I constructed histograms using the 17-filter bank proposed in [197] as features. Pixels across both images were assigned to bins by clustering these responses, using nonparametric methods to estimate the number of clusters k. The first step produces local contextual *descriptors* of each image pixel. The clustering step finds those pixels which are similar under the given descriptor—e.g., similar color and texture will be in the same bin. I also incorporated SIFT-based features, but given that the above histograms already provided good results, this additional module was not utilized further. In Fig. 3.17 I perform correspondences based on *optical flow* in order to segment frames of a video sequence.

Other correspondence determination methods can be used, and no change to our algorithm is needed.

Low Entropy Histograms. When the number of bins is high—the constructed histogram is flat—it is more likely that a non-trivial number of "matches" will be erroneous—especially because in cosegmentation, the two images may have distinct backgrounds without a shared baseline. In my experiments I found that as the entropy increases, so does the JS divergence measure between the histograms for the true segmentations. Low entropy histograms, however, relate to smaller divergences, which impose the global cosegmentation constraint more tightly.

Running Time Complexity. I now discuss what is the strongest advantage of this framework. I show an example in Fig. 3.10 of the running time of the proposed model relative to [75], as a function of *decreasing* entropy (number of bins). The plot suggests that for a realistic range, our implementation has a negligible computation time relative to [75] and the CPLEX-based option—in fact, our curve almost coincides with the x-axis (and even this cost was dominated primarily by the overhead from problem setup done on the CPU). For the most expensive data point shown in Fig. 3.10, the model from [75] generated 10^7 auxiliary nodes (about 12 GB of memory). Due to the utility of low entropy histograms, these experiments show a salient benefit of our framework and its immunity to the "coarseness" of the appearance model. Over all 128×128 images, the wall-clock running time of our CUDA-based model was 10.609 ± 5.230 seconds (a significant improvement over both [75], [88]). The time for a CPLEX-driven method (*utilizing four cores* in parallel) was 17.982 ± 5.07 seconds, but this increases sharply with greater problem size.

Running Time on Artificial Images. To supplement the analysis on natural images, next consider an additional experiment to measure the effect of running time on image size. An artificial image shown in Fig. 3.11 was used in order to allow for isolation of specific variables and mitigate variability (in optimization time) as a function of the specific problem instance (image also used for running time experiments). The purpose was to quantify the

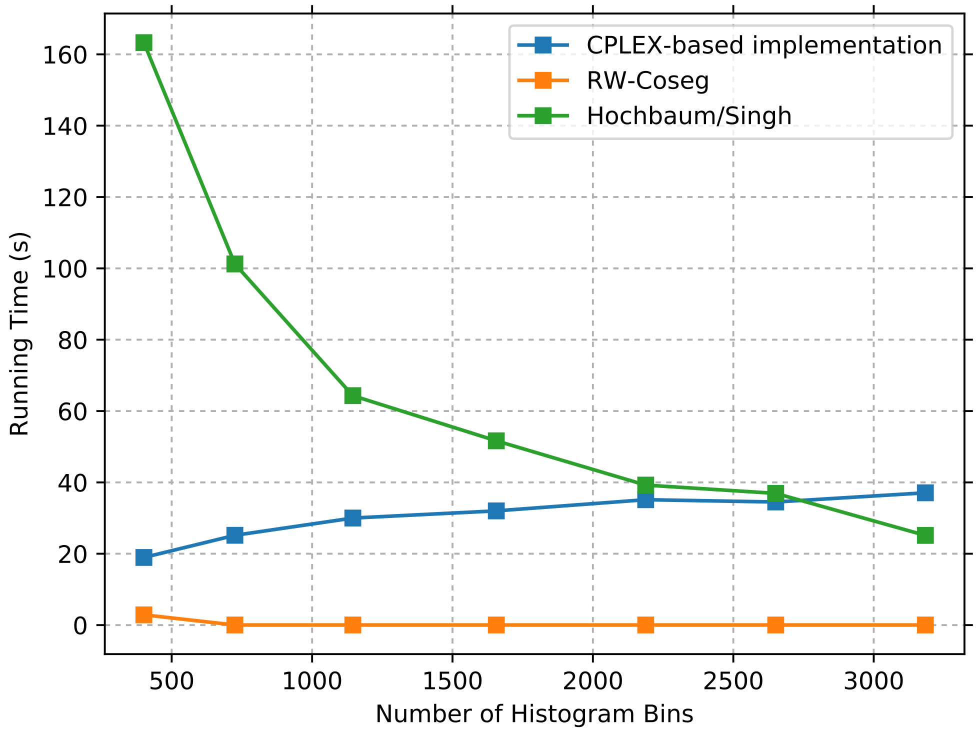


Figure 3.10: Variation in run-time with histogram granularity relative to a CPLEX-based implementation and [75].

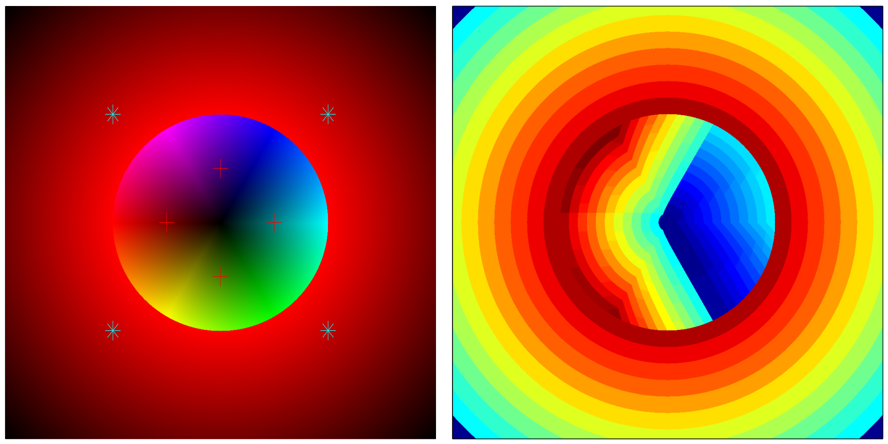


Figure 3.11: Artificial image used in computation time experiments (left) and corresponding color histogram bins (right). Seed points were placed as shown in both foreground and background. To create a cosegmentation problem, the same image was used twice.

advantage offered by the specialized optimization procedure for cosegmentation, compared to the use of an industry-strength QP solver.

Running Time Relative to Image Size. With an increase in the image size, the running time of the model is expected to increase because of two reasons. First, the number of pixels to segment determines the size of the input problem and the dimensionality of the decision variables in the optimization. Second, if the histogram is generated the same way across different sizes, the number of pixels in each histogram bin also increases. An analysis of this behavior is presented in Figure 3.12. The image shown in Fig. 3.11 was generated for various sizes, with the number of pixels along each side plotted along the x axis in Fig. 3.12. Our specialized GPU-based cosegmentation library and an implementation based on the CPLEX solver were used for Random Walker Cosegmentation. In this result, we see only marginal increase in the overall running time of our model (the curve stays close to the x-axis); on the other hand, the running time of the CPLEX-based implementation increases quickly as we increase the size of the images. Notice how this difference is substantially magnified for larger images. The reason is that our algorithm distributes each individual BLAS operation over the GPU's computational units; as a result, higher-dimensional operations take better advantage of the parallelization in the model presented in this chapter. This method is thus especially suited for large images where computation time is a consideration.

Performance w.r.t. Pair Methods. I evaluated the quality of segmentations (0/1 loss) on the 50 image pairs described above, relative to Pseudoflow-coseg [75], LP [129] (only partial), and discriminative clustering [88]. As in [61], a few seeds were placed to specify foreground and background regions, and were given to all methods. Representative samples on the images from [5] using the method in [75], [88] are shown in Fig. 3.13. Averaged over the pair dataset, the segmentation accuracy of our method was $93.6 \pm 2.9\%$, whereas the gross values for the algorithms from [75] and [88] were 89.1% and 84.1% respectively.

Performance on Two or More Images. Across the iCoseg dataset, we achieved an accuracy of 93.7% with seeds provided by five different users. The algorithm of [88] achieves an accuracy of 82.2% across the dataset (excluding some for which the implementation provided by the authors did not complete). Representative image sets and accuracy comparisons appear in Table 3.2 and Figure 3.15.

We note that these results must be interpreted with the caveat that different methods respond differently to seed locations and the level of discrimination offered by the underlying appearance model. Since most cosegmentation models, including this work, share the same basic construction at their core (i.e., image segmentation with an appearance model constraint), variations in performance are in part due to the input histograms. The purpose of our experiments here is to show that at the very least, one can expect similar (if not better) qualitative results with our model, but with more flexibility and significant computational advantages.

Comparisons to Independent Random Walker Runs. In Fig. 3.16, I present qualitative results from our algorithm and from independent runs of Random Walker (both with up to two seeds per image). A trend was evident on all images – the probabilities from independent runs of Random Walker on the two images were diffuse and provide poorer boundary localization. This is due to the lack of global knowledge of the segmentation in

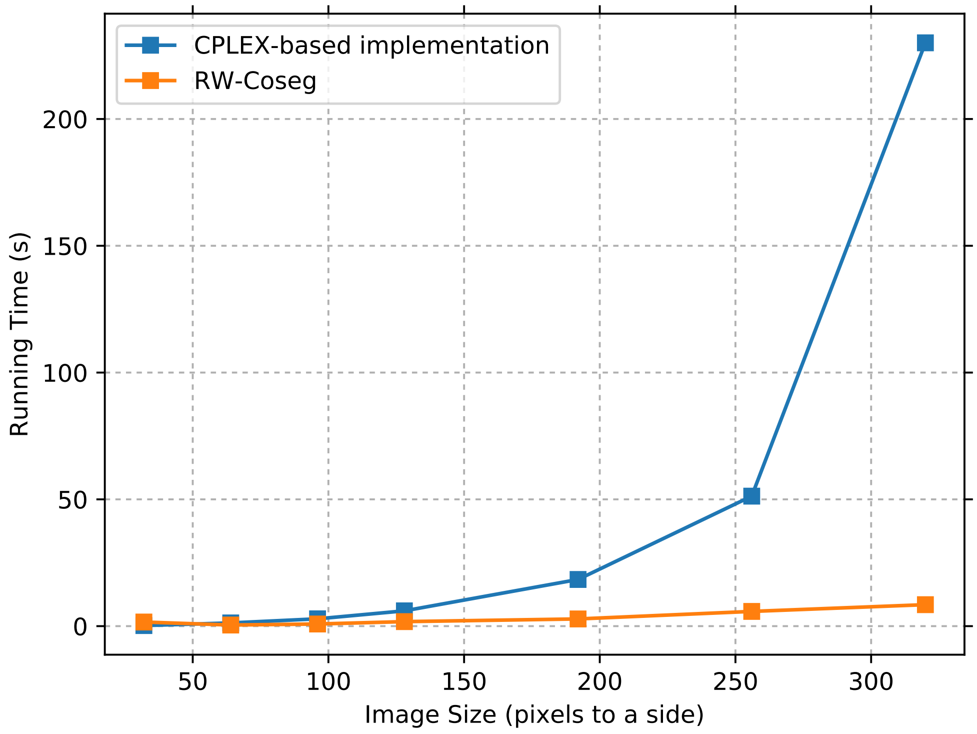


Figure 3.12: Variation in computation time with image size. For each point, the input images were rescales to a different resolution, and the cosegmentation was computed by each of the compared methods.

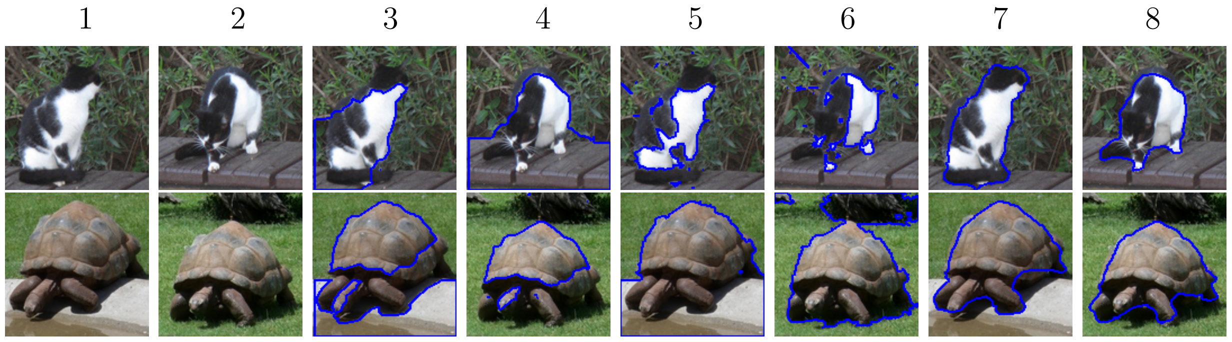


Figure 3.13: Comparison results on example images (columns 1,2) of the the Pseudoflow based method of [75] (columns 3,4) and the discriminative clustering approach of [88] (columns 5,6), with segmentation from RW-based cosegmentation (columns 7,8).

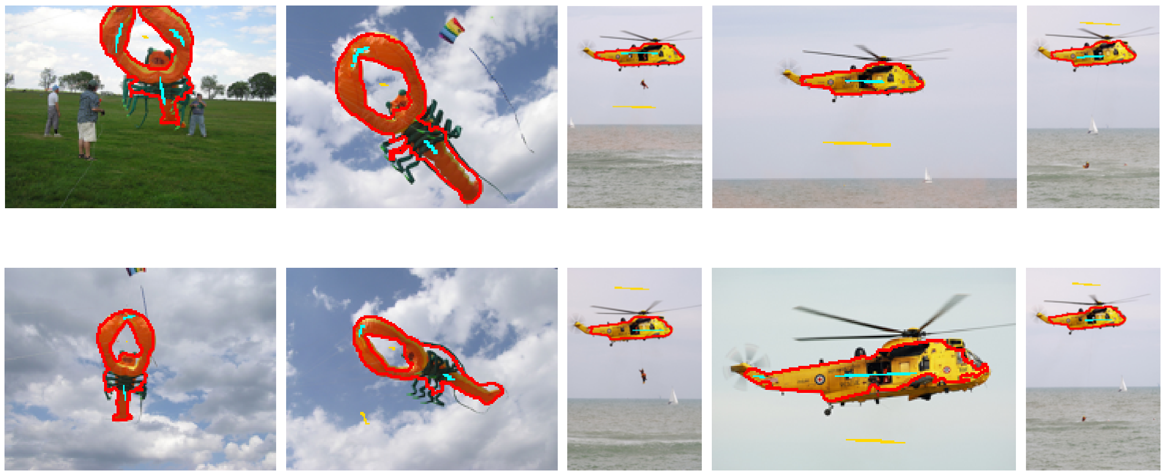


Figure 3.14: Seeded image cosegmentation on iCoseg image sets.

the other image. Random Walker-based cosegmentation is able to leverage this information, and provides better contrast and crisp boundaries for thresholding (a performance boost of up to 10%).

3.6.1 Effect of λ Parameter

The images in Fig. 3.18 demonstrate the role of the histogram consistency bias λ. For a very small λ, the segmentation probabilities are diffuse (compared to the independent Random Walker results); as the influence of the bias grows, the consistency between the histograms makes the partitions more pronounced.

3.7 Summary

The preceding chapter presented a new framework for the cosegmentation problem based on the Random Walker segmentation approach. While almost all previous cosegmentation methods viewed the problem in the MRF (graph-cuts) setting, our algorithm translates many of the advantages of Random Walker to the task of simultaneously segmenting common foregrounds from related images. Significantly, our formulation completely eliminates

Set name (subset)	Our Method	[88]
Airshow (8/21)	99.7%	80.5%
Alaskan bear (5/18)	92.2%	76.6%
Christ (6/14)	95.6%	97.3%
Ferrari (4/10)	96.4%	59.4%
Goose (13/31)	97.3%	97.5%
Helicopter (9/11)	96.9%	98.1%
Kendo (10/31)	91.4%	93.4%
Lobster Kite (4/11)	97.0%	88.5%
Monk (4/17)	83.3%	87.1%
Soccer (5/36)	93.9%	79.9%
Speed Skater (9/13)	80.8%	77.3%
Liberty (11/41)	93.2%	86.1%

Table 3.2: Segmentation accuracy for some iCoseg image sets. Subsets were chosen which have similar appearance under a histogram model.

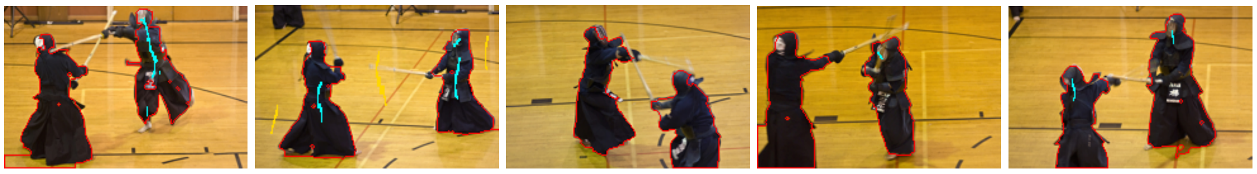


Figure 3.15: Interactive segmentation results using the multi-image model across five images from the "Kendo" set of iCoseg. As the number of images increases, *less* user input is required, as seen by the lack of such for the middle image.

a requirement in some cosegmentation methods that the overall image histogram to be approximately flat. Our model extends nicely to the multi-image setting using a penalty with statistical justification. A further extension allows model-based segmentation which is independent of the relative scales of the model and target foregrounds. The discussion included the optimization specific properties of Random Walker cosegmentation, gave a state of the art GPU based library, and showed quantitative and qualitative performance of the method.

The full set of segmentation results and code are available at `http://pages.cs.wisc.edu/~mcollins/pubs/cvpr2012.html`.

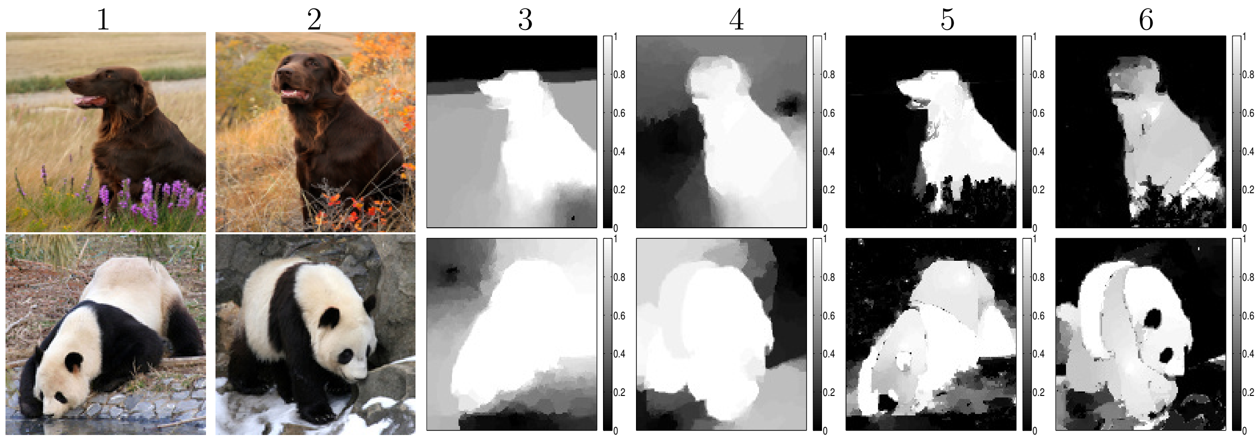


Figure 3.16: Columns (1–2): Input images; columns (3–4): segmentation potentials from independent Random Walker runs on the two images; columns (5–6): segmentation potentials from Random Walker-based cosegmentation. Note that the object boundaries have become significantly more pronounced because of the histogram constraint.

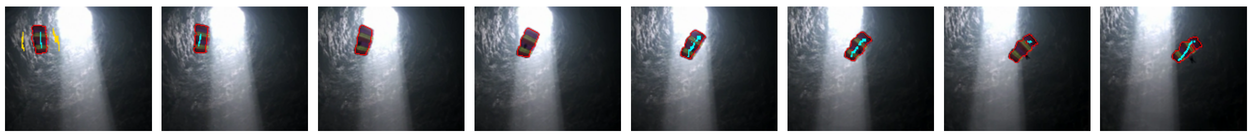


Figure 3.17: Segmentation using correspondences from optical flow on video sequence from [182], shows outline of segmented foreground in red, with foreground and background indications. Our algorithm achieves 99.3% accuracy.

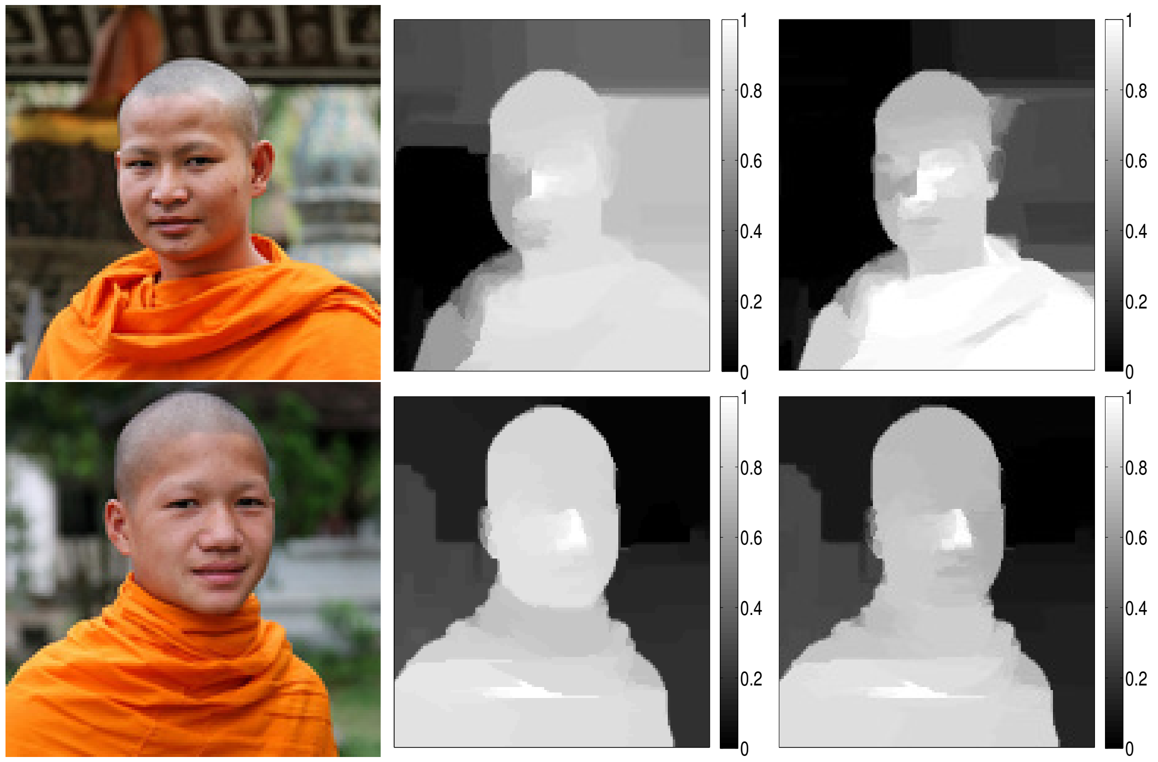


Figure 3.18: Effect of varying λ parameter on an example image for $\lambda = 10^{-8}$ in Column 2, and $\lambda = 10^{-6}$ in Column 3. A segmentation potential biased towards matching the histograms makes the partitions more pronounced.

Chapter 4

Spectral Clustering of Large Datasets

To perform inference on sets of related images, a common first step is to simply *find* groups of images that share a relevant relationship. Clustering serves as an important exploratory tool for categorizing sets of images into semantically meaningful concepts. Spectral objectives, which are the focus of this chapter, analyze the spectrum of a matrix derived from the pairwise similarities of nodes, and are especially useful when the cluster distributions correspond to more complex manifolds. In addition to the Spectral Clustering objective, this work includes ways to extend the clustering to include richer types of information about the target dataset. For instance, in clustering a set of images, we may consider different image features in each view including SIFT [123], GIST [142], color histograms, local binary patterns [141], deep-learning-based descriptors [45], and others from the vast body of literature on useful image descriptors. Further, we want to be able to identify useful clusters in the massive-scale datasets that are increasingly being used in the Computer Vision community. Accordingly, we need a procedure that can handle more powerful clustering models while being scalable and computationally efficient, and utilize additional knowledge as regularizers when possible.

This chapter describes a multi-view Spectral Clustering method and optimization algorithms over the Stiefel manifold to efficiently solve very large-scale problem instances. In this setting, we have multiple "views" of a dataset that each give us a weighted graph over its elements. Each view is expected to provide us with distinct information about the relationships between images in the set. Since any one feature will give us only very incomplete information on the content of an image, we expect the combination of multiple features derived from distinct properties of the image will provide a more complete picture. I therefore seek a Spectral Clustering solution that takes into account all of the views to produce a consistent labelling of the image set. This is further extended to consider priors using side information, such as text tags, which may be treated differently from the views derived from image features.

The two main components to building an ML model for more complex tasks are to leverage more training data and to use a more expressive model. These are mutually dependent, as an overly simple model will under-fit a large dataset, and a complex model will over-fit a small one. Training big models on big datasets, though, also requires greater computation time. The most scalable way to bring more resources to bear on a training problem is to distribute the optimization across an arbitrary number of CPU cores—this is essential if the system is expected to work efficiently on massive datasets. The parallelization extends from considering different views on separate computational units, down to distributing individual example images across cores. This work further shows provably fast convergence, extending theorems concerning stochastic gradient descent to this parallel framework to show a convergence rate that improves as we add cores.

4.1 Related Work

Multi-view clustering is a problem of combining multiple graphs or measures of similarity within a single clustering problem. It combines these views, operating directly on the graph of similarities, to produce the best clustering result, rather than working with the examples directly. This can be viewed as an ensemble [172] construction, where by combining multiple types of clustering with uncorrelated errors, one can achieve a clustering that is "greater than the sum of its parts." Under a good set of independent views, errors made in any one view would be canceled out by better similarity measures in other views. It can also be viewed as a similarity or metric learning problem [10], [90].

Multi-view Spectral Clustering was proposed in [105]. The authors describe a pair of models in which matrices U, which describe the clustering, are maintained for each view, and a Spectral Clustering objective is combined with terms that enforce consistency between the views. They minimize this combined objective using an alternating minization scheme.

Spectral Clustering refers to a class of methods that use the eigenvectors of a matrix over the examples to assign them to clusters. A simple clustering can be done as:

1. Construct the weighted similarity or affinity graph.
2. Compute its graph Lapalacian L as in (3.3).
3. Find the p smallest-magnitude eigenvectors $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_k$ of L.
4. Treating V as a matrix with columns $\mathbf{v}_i$, take the rows as $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n$, so $(\mathbf{y}_i)_j = (\mathbf{v}_j)_i$.
5. Run k-means clustering with the points $\mathbf{y}$.
6. Map the resulting cluster labels back to the input points, so example $\mathbf{x}_i$ is assigned the same labels as $\mathbf{y}_i$.

The role of the spectral algorithm here is non-obvious, as the final clustering is done by a classical k-means algorithm. Importantly, though, the result of k-means clustering applied to the $\mathbf{y}$'s is very different than if one directly applied k-means to the original input examples. The spectral operations serve to find separations between the clusters that are possibly highly non-linear, and map the points to a space where complex manifolds of similar examples are instead close by a simple Euclidean metric and are all mutually similar. See Figure 1 of [137] for a key illustrating example and more discussion.

One additional observation made by the authors in [137] is that the partition that comes from Spectral Clustering corresponds to the "mixing" in a random walk on the distance graph. A natural cluster of mutually close points will see the random walker moving within that cluster with high probability, while crossing between clusters with lower probability. This distribution over the examples also corresponds to the spectrum of this distance matrix.

Note that there are two closely related approaches to Spectral Clustering. As in [137], one can compute a matrix of *distances* between examples, where more similar examples have lesser distance values. The principal eigenvectors of this matrix are then what is used in the clustering. It is also common to use a Laplacian matrix of a similarity graph [192]. Here, very similar examples will have a high similarity value, but the corresponding entries in the Laplacian matrix will be more negative. The clustering is then done with the eigenvectors corresponding to the *smallest* magnitude eigenvectors. In this chapter, I consider only the second method based on the similarity graph Laplacian.

An important justification of Spectral Clustering is that it is a relaxation of the *ratio cut* problem [66]. This is an objective that produces "balanced" clusterings, with clusters that are roughly the same size with no extremely large or trivially small clusters. Variations in the construction of the matrix can yield relaxations of other balanced clustering objectives such as normalized cuts [168], [192]. It has been shown empirically by multiple authors [42], [66], [168] that the relaxation given by Spectral Clustering tends to produce balanced clusters.

The method proposed below belongs to a class of methods covering optimization on the Stiefel manifold [47]. Parallel optimization on the Grassmannian and Stiefel manifolds has been considered in the context of GROUSE [4] and is not novel to this work specifically. In particular, [4] considers rank-one updates of the orthogonal solution matrix V on incomplete portions of the data. However, the important difference is that those schemes include computation linear in the number of rows in the matrix V, which for our case spans the full dataset and is impractical. Our procedure instead looks at only a subset of the rows of V, and therefore has per-iteration cost that is *independent of* n.

4.2 Multi-View Spectral Clustering Model

Take a clustering problem defined over a set of examples $X = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$ to be grouped into p clusters. Spectral Clustering is a graph clustering method, in that the clustering algorithm operates on a graph of similarities or distances between the examples. Classically, Spectral Clustering achieves this task by finding the $(p + 1)$ minimum eigenvectors of a

Laplacian matrix $L \in \mathbb{R}^{n \times n}$ of this graph [192]. One such construction uses the same Laplacian matrix (3.3) used in Chapter 3 for the cosegmentation problem.

Given this matrix, the eigenvectors are typically found via iterative methods such as Lanczos and its variations [114]. The Lanczos algorithm operates on the Laplacian matrix only with matrix-vector or matrix-matrix products. This is especially appropriate for clustering on sparsely connected graphs, as these produce a sparse L. Products including L can then be computed with codes that incur a computational cost that scales with the number of nonzero entries, rather than the n^2 total entries. This means Spectral Clustering can scale to much larger numbers of examples.

As an optimization, Spectral Clustering may be viewed as a minimization of the trace over orthonormal $n \times p$ matrices V:

$$\begin{aligned} \min_{V \in \mathbb{R}^{n \times p}} \quad & \operatorname{tr}(V^T L V) \\ \text{s.t.} \quad & V^T V = I. \end{aligned} \tag{4.1}$$

The orthonormality constraint $V^T V = I$ means that the columns of V are all of unit norm, and are orthogonal. A V satisfying this constraint is an element of the Stiefel manifold, and its columns define the basis of some subspace. At the optimum, this will be the same subspace that is spanned by the eigenvectors of the p least eigenvalues of L. A hard clustering can then be derived from these V, through, for example, some naïve clustering or quantization.

In this chapter, I also consider the "multi-view" generalization of the Spectral Clustering problem [105]. Each view can correspond to a different set of features, or a different similarity measure, over the same set of examples. Assume we are given l views of a dataset. The problem is to look for a common V that balances the solution over all the views. This translates to the following model:

$$\begin{aligned} \min_{V \in \mathbb{R}^{n \times p}} \quad & h(V) := \sum_u \operatorname{tr}(V^T L^{(u)} V) \\ \text{s.t.} \quad & V^T V = I. \end{aligned} \tag{4.2}$$

4.2.1 Incorporating Group Priors

Separate from the multiple views expressed with the graph Laplacians, there is typically a great deal of side information available suggesting (with varying degrees of confidence) that certain subsets of examples are likely to belong to the same class. These can be included in the model with “must-link” constraints instead of an additional similarity function, with the distinction that the model will ensure that this constraint is satisfied in any optimal solution. While must-link constraints may be tedious to deploy via user supervision for a large set of examples, instead, one may for instance impose this prior indirectly. For example, if a set of images shares five or more tags and the data source is reputable, it yields valuable group-level advice to regularize (4.2) and complement the information extracted from the image.

Assume that we have group-prior information about examples where a group is defined as $C = \{v_1, v_2, \cdots, v_{|C|}\}$ where each v_j is a row of V corresponding to an example. We seek to include a must-link constraint between all members of C. To encode similarity in how their respective representations in V behave, one can use a group concentration term, which measures the distance of each example (in the group) to the group's center:

$$g_C(V) = \mu \sqrt{\frac{1}{|C|} \sum_{t=1}^{|C|} d(v_t, \bar{v})^2}, \tag{4.3}$$

where $\bar{v}$ is the geometric center of all points in the set, and $d(\cdot, \cdot)$ is a suitable distance function. This regularization essentially measures intra-group distances, and we obtain a multi-view Spectral Clustering problem with a group prior:

$$\begin{aligned} \min_{V} \quad & f(V) := h(V) + g(V) \\ \text{s.t.} \quad & V^T V = I, \end{aligned} \tag{4.4}$$

where $g(V) = \sum_{\forall C} g_C(V)$ is a convex real-valued function. My subsequent analysis of this problem also allows for non-smooth g, and group norms such as $\ell_{2,1}$ and others may be used based on specific needs. I denote the overall objective by $f(V)$.

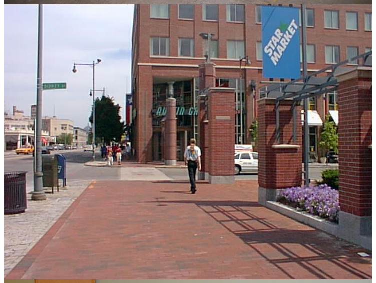
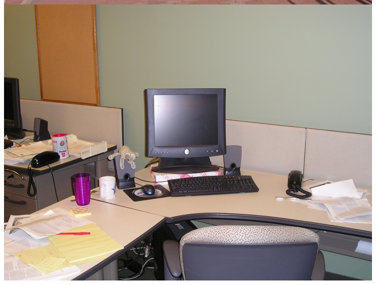
Image	**Tags**
	license plate, wheel, car, light, car right, tail light, wheel rim, window, door, back window
	person walking, text, sign
	speaker, screen frontal, keyboard, telephone, mouse, mousepad, mug, cup, desk, pinboard, armchair crop, monitor crop, papers, notes, plush toy

Figure 4.1: Tagged images from the LabelMe dataset [142]. User-provided tags are a common feature in photo sharing applications.

4.3 Stochastic Gradient Descent Procedure

In order to execute an optimization algorithm on multiple parallel processors, the work must be divided up between each processor. The problem is ideally split up so that each component can be worked on mostly independently, with minimal redundant work and limited communication between threads. The scheme presented below seeks to distribute the problem in such a way that at any given step, each processor only needs to consider a subset of the examples. This is done using the method of *stochastic gradient descent* [39], specifically a variant known as *stochastic block-coordinate descent* [199]. It is easy to see that the objective for $h(V)$ can be expressed as

$$\sum_u \sum_{ij} L_{ij}^{(u)} \langle V_{i\cdot}, V_{j\cdot} \rangle = \sum_u \sum_{i \sim j} w_{ij} \| V_{i\cdot} - V_{j\cdot} \|_2^2 \tag{4.5}$$

where the inner sum is over non-zero entries of the Laplacian matrix for the u^{th} view in the first instance and edges of the corresponding graph with weights w_{ij} in the second expression. Each term of the sum can be considered a sub-function of the objective, and so we can descend along the gradient of a randomly selected subset of the terms. Depending on the sampling strategy, we want the resulting descent direction, *in expectation*, to be equivalent to an ordinary gradient descent on the full objective. Later, this will provide convergence guarantees.

So that a single worker at a single iteration has some independent sample of the objective, we can sample the edges that give us a subset of the terms in (4.5). This subset of edges corresponds to a subset of the nonzeros in the graph Laplacian L. At iteration t, we can call a matrix with only these nonzeros $\hat{L}_t$. Under an appropriate sampling scheme, it will satisfy $\mathbb{E}(\hat{L}_t) = L$. The stochastic gradient descent algorithm can then be applied to the entire objective of (4.4), resulting in the update:

$$V_{t+1} = \mathcal{P}_\Omega(V_t - \gamma_t(\hat{L}_t V_t + \partial g(V_t))), \tag{4.6}$$

Algorithm 1 Comparison on projection (discussed in Section 3.3) and projection-free manifold (see Section 4.3.1) algorithms for solving optimization problems over the Stiefel manifold $S_{n,p}$. When done in parallel, multiple processors may perform independent iterations on different choices of $\hat{L}_t$ and $\mathcal{K}$.

```
Require: $f : \mathbb{R}^{n\times p} \to \mathbb{R}$, $V_0 \in S_{n,p}$
  for $t = 1,\ ...,\ T$ do
    Pick some $u$
    Sample $\hat{L}_t$ from $L^{(u)}$'s (see Section 4.4.1)
    Get subgradient $d \in 2\hat{L}_t V_t + \partial g(V_t)$
    Pick step size $\gamma_t$
    Take step in $\mathbb{R}^{n\times p}$: $V'_{t+1} \leftarrow V_t - \gamma_t d$
    Project onto feasible set:
      $V_{t+1} \leftarrow \mathcal{P}_{S_{n,p}}(V'_{t+1})$
  end for
```

```
Require: $f : S_{n,p} \to \mathbb{R}$, $V_0 \in S_{n,p}$
  for $t = 1,\ ...,\ T$ do
    Select $\mathcal{K} \subseteq \{1,\ \ldots,n\}$
    Take *descent curve* $Y(\tau)$ in $S_{n,p}$ s.t.
      $Y(0) = V_t$
      $\left.\frac{d(f\circ Y)}{d\tau}\right|_{\tau=0} \leq 0$
      $(Y(\tau))_{ij} = (V_t)_{ij} \quad \forall \tau, i \notin \mathcal{K}$
    Pick step size $\tau_t$
    $V_{t+1} \leftarrow Y(\tau_t)$
  end for
```

where $\Omega := \{V : \ V^T V = I\}$ and γ_t denotes step size. Because the operation

$$V_t \to V_t - \gamma_t(\hat{L}_t V_t + \partial g(V_t)) \tag{4.7}$$

does not preserve the orthonormality constraint, in order to iterate towards a feasible solution we re-project back onto the Stiefel manifold at each step. Define $\mathcal{P}_\Omega$ as a projection onto the feasible set, to be described in more detail in the following.

4.3.1 Preserving Feasibility w.r.t. Orthogonality

Ideally, we want to be able to split up the problem into subsets of examples, while producing iterates that satisfy the constraints $V^T V = I$. Say we have a subset $\mathcal{K}$ of the indices, corresponding to rows of V (the submatrix corresponding to these rows is denoted by $V_{\mathcal{K}\cdot}$). We are given a feasible iterate V, and seek to compute the next iterate W such that it is also an orthogonal matrix.

Lemma 4.1. *Given an orthonormal matrix V, and a subset of the rows of V indexed by $\mathcal{K}$, then if*

$$W_{\mathcal{K}\cdot}^T W_{\mathcal{K}\cdot} = V_{\mathcal{K}\cdot}^T V_{\mathcal{K}\cdot} \tag{4.8}$$

and $W_{\bar{\mathcal{K}}\cdot} = V_{\bar{\mathcal{K}}\cdot}$, then W is also an orthonormal matrix.

Proof. V being an orthonormal matrix means

$$\left[V^T V\right]_{ij} = \sum_{k=1}^{n} V_{ki} V_{kj} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j. \end{cases} \tag{4.9}$$

So if we take the same sum for an element of $W^T W$, one can split the terms of the sum by their membership in the row set $\mathcal{K}$:

$$\left[W^T W\right]_{ij} = \sum_{k=1}^{n} W_{ki} W_{kj} = \sum_{k \in \mathcal{K}} W_{ki} W_{kj} + \sum_{k \notin \mathcal{K}} W_{ki} W_{kj} \tag{4.10}$$

Thanks to (4.8), the left sum is equal to $\sum_{k\in\mathcal{K}} V_{ki} V_{kj}$. The right sum is also equal to the same sum over V, as these elements are unchanged from V to W.

Taking this equality over all elements, $W^T W = V^T V = I$, meaning W is an orthonormal matrix. □

Lemma 4.2. *Submatrices $W_{\mathcal{K}\cdot}$ that satisfy* (4.8) *can be parameterized by a rotation of the linearly independent columns of $V_{\mathcal{K}\cdot}$.*

Proof. W.l.o.g., assume $V_{\mathcal{K}\cdot}$ is of the form $[V_{\mathcal{K}\mathcal{I}}, V_{\mathcal{K}\mathcal{I}}R]$, where $V_{\mathcal{K}\mathcal{I}}$ is a maximal subset of linearly independent columns of $V_{\mathcal{K}\cdot}$. Let $P = {V_{\mathcal{K}\mathcal{I}}}^T V_{\mathcal{K}\mathcal{I}}$ be the matrix of inner products of these columns. We know by construction that $P \succ 0$. If U is any orthogonal matrix of the same dimension as $V_{\mathcal{K}I}$, we can construct

$$W(U) = \begin{bmatrix} UP^{1/2} & UP^{1/2}R \\ V_{\bar{\mathcal{K}},\mathcal{I}} & V_{\bar{\mathcal{K}},\bar{\mathcal{I}}} \end{bmatrix} \tag{4.11}$$

assuming w.l.o.g. above that $\mathcal{K}$ selects the first $|\mathcal{K}|$ rows of the matrix. This is constructed such that those rows of V in the *complement* of $\mathcal{K}$ (denoted by $\bar{\mathcal{K}}$) are unchanged in W. Further,

$$\begin{aligned} W(U)_{\mathcal{K}\cdot}^T W(U)_{\mathcal{K}\cdot} &= \begin{bmatrix} UP^{1/2} & UP^{1/2}R \end{bmatrix}^T \begin{bmatrix} UP^{1/2} & UP^{1/2}R \end{bmatrix} \\ &= \begin{bmatrix} P^{1/2}U^T UP^{1/2} & P^{1/2}U^T UP^{1/2}R \\ R^T P^{1/2}U^T UP^{1/2} & R^T P^{1/2}U^T UP^{1/2}R \end{bmatrix} \\ &= \begin{bmatrix} P & PR \\ R^T P & R^T PR \end{bmatrix} \\ &= \begin{bmatrix} V_{\mathcal{K}\mathcal{I}}^T V_{\mathcal{K}\mathcal{I}} & V_{\mathcal{K}\mathcal{I}}^T (V_{\mathcal{K}\mathcal{I}}R) \\ (V_{\mathcal{K}\mathcal{I}}R)^T V_{\mathcal{K}\mathcal{I}} & (V_{\mathcal{K}\mathcal{I}}R)^T (V_{\mathcal{K}\mathcal{I}}R) \end{bmatrix} = V_{\mathcal{K}\cdot}^T V_{\mathcal{K}\cdot} \end{aligned}$$

so the sufficient constraints in (4.8) are preserved. □

The above construction successfully reduces the problem from finding a feasible iterate W to modifying a subset of the rows by multiplying them with an appropriately constructed rotation matrix U. We find this U by moving along a *geodesic* in the Stiefel manifold. The starting point is given as $U_0 = V_{\mathcal{K}\mathcal{I}}P^{-1/2}$, for which $W(U_0) = V$. This U (and consequently each one after it) is then used to compute the next iterate, obtained by moving along the Stiefel manifold in a descent direction for the subproblem on U given by $\min_{U^TU=I} f(W(U))$.

Given the objective in (4.2), the derivative for one Laplacian L at V w.r.t. only $V_{\mathcal{K}\cdot}$ is $2L_{\mathcal{K}\mathcal{K}}V_{\mathcal{K}\cdot} + 2L_{\mathcal{K}\bar{\mathcal{K}}}V_{\bar{\mathcal{K}}\cdot}$, where $L_{\mathcal{K}\mathcal{K}}$ is the submatrix of L where both the rows and columns are in $\mathcal{K}$. The gradient of $h \circ W$ w.r.t. U will then be the sum over each L of

$$2\left(L_{\mathcal{K}\cdot}V_{\cdot\mathcal{I}} + (L_{\mathcal{K}\mathcal{K}}V_{\mathcal{K}\mathcal{I}}R + L_{\mathcal{K}\bar{\mathcal{K}}}V_{\bar{\mathcal{K}}\bar{\mathcal{I}}})R^T\right)P^{1/2}.$$

A similar term is used for the gradient of g that may be derived from $\frac{\partial}{\partial U}g(W(U))$.

The above transformation from updating V to finding an orthogonal matrix U is general and applies to methods such as [47], [195] that optimize within the Stiefel manifold by searching along *geodesics* or curves generated from the Cayley transformation.

4.4 Convergence

Generally, it is difficult to assess the convergence rate for non-convex optimization. However, in this case, we can show that under some mild conditions, the local convergence rate is $O(1/\sqrt{T})$, where T is the number of iterations. Note that the convergence rate analysis is not only useful as a performance measure but helps provide the optimal sampling strategy for our optimization method and also shows how the framework will behave with the addition of more cores, when available. To our knowledge, this is the first result of this kind for Spectral Clustering with regularization. Let $\Delta_t = \hat{L}_t - L$ where $\hat{L}_t$ is the sampled Laplacian at the t^{th} iteration. We first define:

$$\sigma^2 := \max_{V^TV=I,t} \mathbb{E}(\|\Delta_t V\|_F^2); \quad M := \max_{V^TV=I} \|LV\|_F; \quad N := \max_{V^TV=I} \|\partial g(V)\|_F.$$

Notice that M and N are constants decided by L and g respectively, while σ^2 directly depends on the sampling strategy. For notational convenience, we define a function Υ as follows:

$$\Upsilon(M, N, \sigma^2, T) := \sqrt{\frac{(M+N)^2 + \sigma^2}{T}}. \tag{4.12}$$

Our convergence result states:

Theorem 4.3. *Let V^* be a convergent point of the sequence $\{V_t\}$ generated from (4.6) that is in a small ball with radius δ, and denote $f(V^*)$ as f^*. Let ϕ be a positive value. If there exists a constant $\delta > 0$ such that $\mathcal{P}_\Omega\left(V_t - \gamma_t(\hat{L}_t V_t + \partial g(V_t))\right)$ is a non-expansive projection, we have:*

i) *If the step size is chosen as $\gamma_t = \frac{\phi\delta}{\sqrt{((M+N)^2+\sigma^2)T}}$ and $\bar{V}_T = (\sum_{t=1}^T \gamma_t)^{-1} \sum_{t=1}^T \gamma_t V_t$, then $\mathbb{E}\left(f(\bar{V}_T)\right) - f^* \leq (\phi + \phi^{-1})\frac{\delta}{2}\Upsilon$.*

ii) *If the step size is chosen as $\gamma_t = \theta_t \frac{f(V_t)-f^*}{(M+N)^2+\sigma^2}$, then $\mathbb{E}(f(\tilde{V}_T)) - f^* \leq \frac{\delta}{\sqrt{\theta_{\min}}}\Upsilon$ where $\tilde{V}_T = \frac{1}{T}\sum_{t=1}^T V_t$, $\theta_t \in (0,2)$ and $\theta_{\min} = \min_t 1 - (\theta_t - 1)^2$.*

Proof. Consider the expansion of $\|V_{t+1} - V^*\|_F^2$:

$$\begin{aligned}\|V_{t+1} - V^*\|_F^2 &= \|\mathcal{P}_\Omega(V_t - \gamma_t((L+\Delta_t)V_t + \partial g(V_t)) - \mathcal{P}_\Omega(V^*)\|_F^2 \\ &\quad \text{from the local non-expansive projection property,} \\ &\leq \|V_t - \gamma_t((L+\Delta_t)V_t + \partial g(V_t)) - V^*\|_F^2 \\ &\leq \|V_t - V^*\|^2 + \gamma_t^2 \underbrace{\|(L+\Delta_t)V_t + \partial g(V_t)\|_F^2}_{T_1} \\ &\quad - 2\gamma_t \underbrace{\langle (L+\Delta_t)V_t + \partial g(V_t), V_t - V^*\rangle}_{T_2}.\end{aligned} \tag{4.13}$$

Take the conditional expectation of T_1 and T_2 in terms of Δ_t given V_t:

$$\begin{aligned}\mathbb{E}(T_1) &= \|LV_t + \partial g(V_t)\|_F^2 + \mathbb{E}(\|\Delta_t V_t\|_F^2) + 2\mathbb{E}\langle LV_t + \partial g(V_t), \Delta_t V_t\rangle \\ &= \mathbb{E}(\|LV_t + \partial g(V_t)\|_F^2) + \mathbb{E}(\|\Delta_t V_t\|_F^2) \\ &\leq (M+N)^2 + \sigma^2\end{aligned} \tag{4.14}$$

$$\mathbb{E}(T_2) = \mathbb{E}\langle LV_t + \partial g(V_t), V_t - V^*\rangle \geq \mathbb{E}(f(V_t)) - f^*. \tag{4.15}$$

Take the expectation of both sides of (4.13) in terms of all random variables, together with (4.14) and (4.15), we have

$$2\gamma_t(\mathbb{E}(f(V_t)) - f^*) \leq \mathbb{E}\|V_t - V^*\|_F^2 - \mathbb{E}(\|V_{t+1} - V^*\|_F^2) + \gamma_t^2((M+N)^2 + \sigma^2), \tag{4.16}$$

which implies that

$$2\sum_{t=1}^{T}\gamma_t(\mathbb{E}(f(V_t))-f^*) \le \mathbb{E}\|V_1 - V^*\|_F^2 + ((M+N)^2+\sigma^2)\sum_{t=1}^{T}\gamma_t^2$$
$$\le \delta^2 + ((M+N)^2+\sigma^2)\sum_{t=1}^{T}\gamma_t^2.$$

Also note that

$$\sum_{t=1}^{T}\gamma_t = \frac{\phi\delta\sqrt{T}}{\sqrt{(M+N)^2+\sigma^2}} \quad \sum_{t=1}^{T}\gamma_t^2 = \frac{(\phi\delta)^2}{(M+N)^2+\sigma^2},$$

and

$$\frac{\sum_{t=1}^{T}\gamma_t\mathbb{E}(f(v_t))}{\sum_{t=1}^{T}\gamma_t} = \frac{\mathbb{E}\sum_{t=1}^{T}\gamma_t f(v_t)}{\sum_{t=1}^{T}\gamma_t} \le \mathbb{E}f(\bar{V}_t) \quad \text{(from the convexity of } f(V_t)\text{)}.$$

It follows that

$$\frac{\sum_{t=1}^{T}\gamma_t(\mathbb{E}(f(V_t))-f^*)}{\sum_{t=1}^{T}\gamma_t} \le \frac{\delta^2 + ((M+N)^2+\sigma^2)\sum_{t=1}^{T}\gamma_t^2}{2\sum_{t=1}^{T}\gamma_t}$$
$$\Rightarrow \mathbb{E}(f(\bar{V}_t)-f^*) \le \frac{\delta^2 + ((M+N)^2+\sigma^2)\sum_{t=1}^{T}\gamma_t^2}{2\sum_{t=1}^{T}\gamma_t}$$
$$= \frac{\delta^2 + ((M+N)^2+\sigma^2)\frac{(\phi\delta)^2}{(M+N)^2+\sigma^2}}{2\frac{\phi\delta\sqrt{T}}{\sqrt{(M+N)^2+\sigma^2}}}$$
$$= (\phi+\phi^{-1})\frac{\delta\sqrt{(M+N)^2+\sigma^2}}{2\sqrt{T}},$$

proving the first claim. Next we prove the second claim. From (4.13), (4.14), and (4.15), we have

$$\mathbb{E}(\|V_{t+1}-V^*\|_F^2) \le \|V_t - V^*\|_F^2 + \gamma_t^2((M+N)^2+\sigma^2) - 2\gamma_t(f(V_t)-f^*)$$
$$\le \|V_t - V^*\|_F^2 - \frac{(f(V_t)-f^*)^2}{(M+N)^2+\sigma^2} + ((M+N)^2+\sigma^2)\left(\gamma_t - \frac{f(V_t)-f^*}{(M+N)^2+\sigma^2}\right)^2$$
$$\le \|V_t - V^*\|_F^2 - \frac{(1-(1-\theta_t)^2)(f(V_t)-f^*)^2}{(M+N)^2+\sigma^2}$$
$$\le \|V_t - V^*\|_F^2 - \frac{\theta_{min}(f(V_t)-f^*)^2}{(M+N)^2+\sigma^2}.$$

It follows that

$$\frac{\theta_{min}}{(M+N)^2+\sigma^2}\mathbb{E}(f(V_t)-f^*)^2 \le \mathbb{E}(\|V_t - V^*\|_F^2) - \mathbb{E}(\|V_{t+1}-V^*\|_F^2). \tag{4.17}$$

Taking $t = 0, 1, \cdots, T-1$ in (4.17) respectively and summarizing all of them, we obtain

$$\frac{\theta_{min}}{(M+N)^2+\sigma^2}\sum_{t=1}^{T}\mathbb{E}(f(V_t)-f^*)^2 \le \mathbb{E}(\|V_1 - V^*\|_F^2) \le \delta^2$$
$$\Rightarrow \; T^{-1}\sum_{t=1}^{T}\mathbb{E}(f(V_t)-f^*)^2 \le \frac{\delta^2((M+N)^2+\sigma^2)}{T\theta_{min}}.$$

Taking these together, it show that

$$\begin{aligned} T^{-1}\sum_{t=1}^{T}\mathbb{E}(f(V_t)-f^*)^2 &\ge T^{-1}\sum_{t=1}^{T}(\mathbb{E}(f(V_t))-f^*)^2 \\ &\ge (T^{-1}\sum_{t=1}^{T}\mathbb{E}(f(V_t))-f^*)^2 \ge (\mathbb{E}(f(\tilde{V}_T))-f^*)^2. \end{aligned}$$

The last inequality uses Jensen's inequality—that is, $\mathbb{E}f(x) \ge f(\mathbb{E}(x))$ holds for any convex function. We prove the second claim. $\square$

From Theorem 4.3, it is clear that independent of how the step size is chosen, the local convergence rate is essentially bounded by $\Upsilon \in O(1/\sqrt{T})$. Next, we further investigate the behavior of σ^2, and introduce sampling strategies based on nodes and edges.

Similar convergence can also be achieved by the partial stochastic gradient projection method, that is,

$$V_{t+1} = \mathcal{P}_\Omega(V_t - \gamma_t \partial_{[t]} f(V_t)) \tag{4.18}$$

where $\partial f(V_t) := LV_t + \partial g(V_t)$ is the subgradient of $f(V)$ at V_t and $\partial_{[t]} f(V)$ is a vector with the same size as $\partial f(V)$ taking the same values on the set $[t]$ and setting the rest as 0.

4.4.1 Sampling

To meet the theoretical requirements for convergence of stochastic gradient descent, the randomly generated $\hat{L}_t$ should satisfy $\mathbb{E}(\hat{L}_t) = L$. To keep the notations and the presentation simple, I write the results and sampling strategy in the context of a single Laplacian. The following discusses a sampling strategy that only uniformly samples the nonzero elements in L. Note that nonzero elements in L correspond to edges in the graph. Define $\bar{L}_{ij} \in \mathbb{R}^{n\times n}$ to be an extended matrix with L_{ij} at the ij^{th} element and zeros at the rest. Generate the stochastic gradient at the current iteration as $\hat{L}_t = \frac{\|L\|_0}{|\mathcal{E}|}\sum_{ij\in\mathcal{E}}\bar{L}_{ij}$ where $\mathcal{E}$ is the set of randomly selected edges.

In order to simplify the following discussion, I assume that the sampling strategy chooses a fixed number of edges at each iteration. These assumptions imply that every nonzero element (edge) in L has equal probability to be chosen.

Let $\lambda_i(\Delta^T\Delta)$ denote the i^{th} largest eigenvalue of $\Delta^T\Delta$. From the definition of σ^2, we have

$$\sigma^2 = \mathbb{E}\left(\max_{V^TV=I}\|\Delta V\|_F^2\right) = \mathbb{E}\left(\sum_{i=1}^{p}\lambda_i(\Delta^T\Delta)\right) \le \mathbb{E}\left(\|\Delta\|_F^2\right) = \sum_{ij\in\mathcal{E}}\mathbb{E}\left(\Delta_{ij}^2\right) \tag{4.19}$$

One can easily verify that $\mathbb{E}(\Delta_{ij}^2) = \left(\frac{\|L\|_0}{|\mathcal{E}|}-1\right)L_{ij}^2$. Thus, from (4.19) we have $\sigma^2 \le \left(\frac{\|L\|_0}{|\mathcal{E}|}-1\right)\|L\|_F^2$. When the cardinality of L is large, $\|L\|_0 \gg |\mathcal{E}|$. In other words, σ^2 dominates the convergence rate. When $\sqrt{\frac{\|L\|_0}{|\mathcal{E}|}}$ is large, the bound is dominated by

$$O\left(\sqrt{\frac{\|L\|_0}{T|\mathcal{E}|}}\|L\|_F\right) = O((T|\mathcal{E}|)^{-1/2}). \tag{4.20}$$

Note that the size of $\mathcal{E}$ is proportional to the number of optimization workers. It means that the convergence can be sped up linearly by increasing the number of threads on different cores or computers—exactly the behavior one hopes to achieve in the ideal situation. In addition, this linear speedup property is also achieved by the *partial* stochastic gradient projection method.

This edge sampling strategy can be easily extended to the setting where multiple separable views live in a distributed environment. The basic change here is that one needs to sample edges across all views, satisfying the condition $\mathbb{E}(\hat{L}_t) = \sum_u L^{(u)}$. This condition is true for a sampling strategy that first chooses a single u with probability proportional to the number of edges in $L^{(u)}$ from which edges are sampled identically to the single-view case. Similar linear speedup properties can be obtained; the result above carries through with essentially mechanical changes. Besides sampling edges, one may sample nodes in the graph to generate the stochastic gradient.

Denote $[t]$ as a subset of coordinates of $V \in \mathbb{R}^{n\times p}$, which is randomly selected at iteration t. To make our following discussion simpler, we assume that the size of $[t]$ is a constant and denote the ratio $R := \frac{np}{|[t]|}$. Consider the following update for V_{t+1}:

$$V_{t+1} = \mathcal{P}_\Omega(V_t - \gamma_t \partial_{[t]} f(V_t)) \tag{4.21}$$

We can show a similar convergence property to Theorem 4.3 for this setting:

Theorem 4.4. *Let V^* be a convergent point of the sequence $\{V_t\}$ generated from* (4.21) *which is in a small ball with radius δ, and denote $f(V^*)$ as f^*. Let ϕ be a positive value. Let $\bar{\Upsilon} := \frac{(M+N)R}{\sqrt{T}}$. If there exists a constant $\delta > 0$ such that $\mathcal{P}_\Omega\left(V_t - \gamma_t \partial_{[t]} f(V_t))\right)$ is a non-expansive projection, we have:*

i) *If the step size is chosen as $\gamma_t = \frac{\phi\delta}{(M+N)\sqrt{T}}$ and $\bar{V}_T = (\sum_{t=1}^T \gamma_t)^{-1}\sum_{t=1}^T \gamma_t V_t$, then $\mathbb{E}\left(f(\bar{V}_T)\right) - f^* \leq (\phi + \phi^{-1})\frac{\delta}{2}\bar{\Upsilon}$.*

ii) *If the step size is chosen as $\gamma_t = \theta_t \frac{f(V_t)-f^*}{R(M+N)^2}$, then $\mathbb{E}(f(\tilde{V}_T)) - f^* \leq \frac{\delta}{\sqrt{\theta_{\min}}}\bar{\Upsilon}$ where $\tilde{V}_T = \frac{1}{T}\sum_{t=1}^T V_t$, $\theta_t \in (0,2)$ and $\theta_{\min} = \min_t 1 - (\theta_t - 1)^2$.*

This theorem basically shows the convergence rate for (4.21) is $O(1/\sqrt{T})$, which is the same as the full projection. The speedup property is also similar: both convergence rates are proportional to R. R is basically the inverse of the block size of $[t]$. Hence, when the block size increases x times, the required iterations to achieve the given accuracy decrease x times.

Proof. Consider the expansion of $\|V_{t+1} - V^*\|_F^2$:

$$\begin{aligned}\|V_{t+1} - V^*\|_F^2 &= \|\mathcal{P}_\Omega(V_t - \gamma_t \hat{\partial}_{[t]} f(V_t)) - \mathcal{P}_\Omega(V^*)\|_F^2 \\ &\leq \|V_t - \gamma_t \hat{\partial}_{[t]} f(V_t) - V^*\|_F^2 \quad \text{(from the local non-expansive projection property)} \\ &\leq \|V_t - V^*\|^2 + \gamma_t^2 \underbrace{\|\hat{\partial}_{[t]} f(V_t)\|_F^2}_{T_3} - 2\gamma_t \underbrace{\langle \hat{\partial}_{[t]} f(V_t), V_t - V^*\rangle}_{T_4}.\end{aligned} \tag{4.22}$$

Take the conditional expectation of T_1 and T_2 in terms of Δ_t given V_t:

$$\mathbb{E}(T_3) = \mathbb{E}(\|\partial_{[t]} f(V_t)\|_F^2) \leq \mathbb{E}\|\partial f(V_t)\|_F^2 = \mathbb{E}\|LV_t + \partial g(V_t)\|_F^2 \leq (M+N)^2 \tag{4.23}$$

$$\mathbb{E}(T_4) = \mathbb{E}\langle \partial_{[t]} f(V_t), V_t - V^* \rangle = \frac{1}{R}\mathbb{E}\langle \partial f(V_t), V_t - V^* \rangle \geq \frac{1}{R}(\mathbb{E}(f(V_t)) - f^*). \tag{4.24}$$

Take the expectation on both sides of (4.22) in terms of all random variables, and we have

$$2\gamma_t(\frac{1}{R}(\mathbb{E}f(V_t) - f^*)) \leq \mathbb{E}\|V_t - V^*\|^2 - \mathbb{E}\|V_{t+1} - V^*\|_F^2 + \gamma_t^2(M+N)^2.$$

The rest of the proof can follow the proof of Theorem 4.3 by simply treating "$\frac{1}{R}(\mathbb{E}f(V_t) - f^*)$" as "$\mathbb{E}f(V_t) - f^*$" in (4.16). □

Projection vs. Manifold Optimization. In order to realize the full benefits of parallelizing the optimization across multiple threads, we use the manifold optimization method of Section 4.3.1. This does not satisfy the conditions of a non-expansive projection P_Ω in Theorem 4.3. Rather, it has the properties of a block coordinate descent method and does not leave the feasible region at any time.

4.5 Generalized Orthogonality Constraints

A more general version of the model in (4.2), with applications in medical imaging, uses a more inclusive type of orthogonality constraints. In this, we consider a feasible set of matrices V that are orthogonal with respect to a positive definite matrix A. For example, one can solve a generalized eigenvalue problem $CV = AVD$ with the minimization

$$\begin{aligned} \min_{V \in \mathbb{R}^{n \times p}} \quad & \langle V, CV \rangle \\ \text{s.t.} \quad & V^T A V = I_{p \times p}. \end{aligned} \tag{4.25}$$

This operation constitutes the key computational phase of the Heat Kernel Smoothing procedure used in [163] to smooth signals over anatomical surfaces in 3D medical images.

The application of the above procedure to this problem allows a natural regularization in this domain. When smoothing over a surface, we can analyze the signal in the spherical harmonics domain [163]. We expect to find a more consistent smoothed signal if it is sparse on the spherical harmonic basis, especially if it primarily lies on the lower-frequency harmonics. Consequently, we may consider the regularizer

$$g(V) = \sum_l \lambda_l \|\Gamma_l V\|_1 \tag{4.26}$$

where Γ_l is the change of basis for V onto the spherical harmonics of degree l, and λ_l are constant multipliers increasing with l.

4.5.1 Optimization

The procedure used for ordinary orthogonality constraints may be extended to the generalized form $V^T AV = I$ with some modification. Consider a submatrix $V_{\mathcal{K}\cdot}$ consisting of rows of V. The constraints on $V_{\mathcal{K}\cdot}$ will be of the form

$$V_{\mathcal{K}\cdot}^T A_{\mathcal{K}\mathcal{K}} V_{\mathcal{K}\cdot} + V_{\bar{\mathcal{K}}\cdot}^T A_{\bar{\mathcal{K}}\mathcal{K}} V_{\mathcal{K}\cdot} + V_{\mathcal{K}\cdot}^T A_{\bar{\mathcal{K}}\mathcal{K}}^T V_{\bar{\mathcal{K}}\cdot} = P_1 \tag{4.27}$$

for some constant matrix P_1. Any change to $V_{\mathcal{K}\cdot}$ that preserves this constraint will produce a V which is also feasible. If we assume that $A_{\mathcal{K}\mathcal{K}}$ is full-rank, we find that this is equivalent to

$$\left(V_{\mathcal{K}\cdot} + A_{\mathcal{K}\mathcal{K}}^{-1} A_{\bar{\mathcal{K}}\mathcal{K}}^T V_{\bar{\mathcal{K}}\cdot}\right)^T A_{\mathcal{K}\mathcal{K}} \left(V_{\mathcal{K}\cdot} + A_{\mathcal{K}\mathcal{K}}^{-1} A_{\bar{\mathcal{K}}\mathcal{K}}^T V_{\bar{\mathcal{K}}\cdot}\right) = P \tag{4.28}$$

where the r.h.s. $P = P_1 + V_{\bar{\mathcal{K}}\cdot}^T A_{\bar{\mathcal{K}}\mathcal{K}} A_{\mathcal{K}\mathcal{K}}^{-1} A_{\mathcal{K}\bar{\mathcal{K}}} V_{\bar{\mathcal{K}}\cdot}$ is still constant w.r.t. $V_{\mathcal{K}\cdot}$.

In the following, we require that $A_{\mathcal{K}\mathcal{K}} \succ 0$, which will be true for all $\mathcal{K}$ if $A \succ 0$. In this case P will be full-rank if and only if the matrix $N = V_{\mathcal{K}\cdot} + A_{\mathcal{K}\mathcal{K}}^{-1} A_{\bar{\mathcal{K}}\mathcal{K}}^T V_{\bar{\mathcal{K}}\cdot}$ is full-rank. We thus further decompose the constraints by considering a full-rank submatrix. Let $\mathcal{I}$ be a maximal set of linearly independent columns of N; we can w.l.o.g. define a matrix R such that $N = [N_{\cdot\mathcal{I}}, N_{\cdot\mathcal{I}}R]$. The constraints in (4.28) can then be reduced to constraints only on the linearly independent columns,

$$\left(V_{\mathcal{K}\mathcal{I}} + A_{\mathcal{K}\mathcal{K}}^{-1} A_{\bar{\mathcal{K}}\mathcal{K}}^T V_{\bar{\mathcal{K}}\mathcal{I}}\right)^T A_{\mathcal{K}\mathcal{K}} \left(V_{\mathcal{K}\mathcal{I}} + A_{\mathcal{K}\mathcal{K}}^{-1} A_{\bar{\mathcal{K}}\mathcal{K}}^T V_{\bar{\mathcal{K}}\mathcal{I}}\right) = P_{\mathcal{I}\mathcal{I}}. \tag{4.29}$$

Given a feasible matrix V, we can define a mapping W by

$$\begin{aligned} W(U)_{\mathcal{K}\mathcal{I}} &= U - A_{\mathcal{K}\mathcal{K}}^{-1} A_{\bar{\mathcal{K}}\mathcal{K}}^T V_{\bar{\mathcal{K}}\mathcal{I}} \\ W(U)_{\mathcal{K}\bar{\mathcal{I}}} &= UR - A_{\mathcal{K}\mathcal{K}}^{-1} A_{\bar{\mathcal{K}}\mathcal{K}}^T V_{\bar{\mathcal{K}}\bar{\mathcal{I}}} \\ W(U)_{\cdot\bar{\mathcal{K}}} &= V_{\cdot\bar{\mathcal{K}}}. \end{aligned} \tag{4.30}$$

For any matrix U such that

$$U^T A_{\mathcal{K}\mathcal{K}} U = P_{\mathcal{I}\mathcal{I}}, \tag{4.31}$$

the matrix given by W will be feasible if and only if V is: $W(U)^T A W(U) = V^T A V$. Specifically, for the starting point $U_0 = N_{\cdot\mathcal{I}}$ we have that $U_0^T A_{\mathcal{K}\mathcal{K}} U_0 = I$ and $W(U_0) = V$.

In each iteration, we seek to decrease the objective f by our choice of U. The previous section has reduced the large $n \times p$ Stiefel manifold problem to a smaller $|\mathcal{K}| \times |\mathcal{I}|$ problem subject to constraints of the same form. Since the allowable U lies on a Stiefel manifold, this can be solved by manifold optimization methods such as [195] as in the previous case with simple orthogonality constraints.

4.5.2 Multi-threaded Constraint Maintenance

A key advantage of decomposing the problem into subsets $\mathcal{K}$ of the rows of V is that we may run multiple parallel threads, each optimizing on a different choice of $\mathcal{K}$. However, unlike in the case of ordinary orthogonality constraints, we expect some interaction between different rows of V. This is solved by requiring that if one thread is modifying rows $\mathcal{K}$ and another is simultaneously optimizing on rows $\mathcal{K}'$, then $A_{\mathcal{K}\mathcal{K}'} = 0$. Equivalently, we require that there is no edge in the graph of nonzeros of A between these two sets of vertices.

4.6 Experiments

The following describes a number of experiments to evaluate this method on several aspects: (a) performance w.r.t. to a variety of datasets with special emphasis on scalability as a function of size, (b) comparison with state of the art method [105] when appropriate, and (c) performance when incorporating high-level priors into the model. Though the focus was to show that the method is applicable for multi-view Spectral Clustering (with convex regularization) for larger computer vision datasets, for which few alternatives are available, I also performed some experiments on Machine Learning datasets as a sanity check, that matched reported results. The vision datasets cover Caltech101, Caltech256, LabelMe, and TinyImages. For experiments with very large datasets, I also used simulated datasets with hundreds of millions of examples. As a performance comparison measure, the following section reports on Normalized Mutual Information (NMI).

4.6.1 Multi-view ML Datasets

UCI digits: The first dataset is the handwritten digits (0-9) data from the UCI repository. The dataset consists of 2 000 examples, with six sets of features for each image from which I construct six views. These results are summarized in Table 4.1. Since my method depends on initialization, the experiments were repeated 10 times (different initializations) and I report on the best NMI value obtained and standard deviations. The authors of [105] provide an initialization in their code using eigenvectors of individual views.

Reuters Multilingual: In addition, I consider multi-view Spectral Clustering on a natural language dataset. We subsample the dataset in a manner consistent with [105]. Since the features for this dataset are sparse and high-dimensional, I first use Latent Semantic Analysis (LSA) [76] to reduce the dimensionality.

4.6.2 Multi-view Vision Datasets

Caltech101: I evaluated the method on Caltech101, a popular benchmark for object categorization with 102 categories of images (101 distinct objects and background), and 30 images per category. To generate the views, I use the UCSD-MKL dataset—a collection of kernels derived from various visual features (up to 25) for Caltech101 data. I used only the training class kernels in an unsupervised setting. Kernels for five random splits, with each split containing information regarding 1515 images (15 images for each of the 101 categories) is provided. I report also results randomly selecting subsets of the views. In each case, I

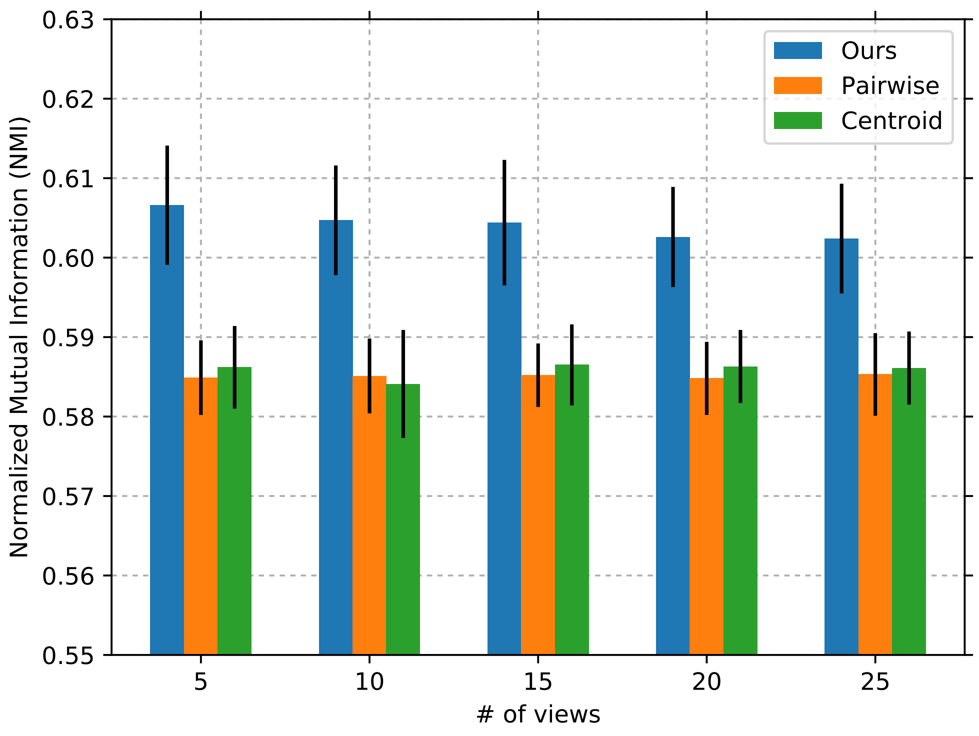


Figure 4.2: Comparing clustering results on the Caltech101 dataset, showing the NMI values for different choices of views for the Spectral Clustering model in this chapter.

	Digits	Reuters
Ours	0.798(0.03)	0.312(0.01)
[105] Pairwise	0.659	0.305
[105] Centroid	0.669	0.308
Best 1-view	0.641	0.288

Table 4.1: Comparison of clustering results on UCI Digits and Reuters, with mean (and s.d.) normalized mutual information (NMI).

report our summaries as well as [105] by averaging across all five splits. The results in Figure 4.2 suggest that the method compares well to [105].

Caltech256: A similar but bigger dataset is Caltech256, which contains 256 object classes and more than 30 000 images across all classes. I restrict my evaluations to three main features for each image—V1-like [165], SURF [6] and Region Covariance (RegCov) [183]—for generating views for this data. The Spectral Hashing method in [104] was then adapted to construct the graph for the Laplacians. Note that here I cannot perform comparisons with [105], since their method requires a dense kernel construction. Because of the nature of this dataset, the V1-like view alone yields a NMI of 0.267, amd SURF gives 0.207, whereas RegCov performs poorly at 0.088. Contrary to the results from other datasets above, here, the multi-view performance at 0.181 is close to but worse than the best view. With two views, SURF and V1-like, multi-view NMI is 0.22. There are two primary reasons. First, the views do not seem to be uncorrelated, and the best view, V1-like, seems to dominate the others. Since there are only a few feature types, we cannot expect an improvement over the single best view. Despite these issues, the evaluations suggest that solving multi-view Spectral Clustering for these sizes *is* feasible, if the features are assumed to be provided.

ImageNet: We can construct a dataset with similar properties to the above following a similar procedure in [116]. I use ILSVRC 2013 [41], an updated version of the challenge set that is the basis of the dataset in [116]. ImageNet categories consist of WordNet noun synsets, which precisely define the object in the image. From ILSVRC 2013 [41] I select 100 categories at random, with a total of 127 885 images selected. I use four views derived from Decaf [45], GIST [142], TinyImage [178], and SIFT [123] features. Each view considered separately produces NMIs of 0.198, 0.181, 0.181, and 0.184 respectively. The multi-view objective combining all four produces a labeling with an NMI of 0.203.

4.6.3 Jumbo-sized Datasets

I summarize my main experiments on very large datasets here. Note that there are significant implementation issues (e.g., memory management, data structures, queries) in successfully running a system on tens of millions of examples.

TinyImages: TinyImages [178] is a set of nearly 80 million 32×32 color images collected from internet searches. The dataset is distributed along with GIST features computed on each image, which were used as the basis of our clustering. Nearest neighbors were computed using [104], from which a weighted graph with 320 million edges was constructed. The dataset includes a keyword associated with each image; for 24 690 images the dataset authors evaluated the accuracy of this keyword, out of which 5660 images depicted the associated keyword. This keyword is the only form of label provided with the TinyImage dataset; no ground truth is available.

I split the entire TinyImages dataset into two clusters using Spectral Clustering with the manifold optimization method. With 34 CPU cores, the optimization averaged one iteration every 0.015 seconds. To qualitatively evaluate the clustering at a local scale, we look at how individual keywords are split between the clusters. While most keywords are split by this clustering, some keywords corresponding to more homogeneous sets of images are well separated into one cluster or the other. In Figure 4.3, I show a subset of the keywords sampled from both well-clustered and poorly clustered images.

ImageNet: One can also test a clustering task on the full ILSVRC2013 dataset. This full dataset has 1000 categories and 1 281 165 images. Since our optimization procedure considers a high-n low-p regime, I find a two-way split as in the TinyImages. The (non-normalized) MI of the two-way labelling versus the ground truth is 0.229.

Mixture model: To assess the scalability of my optimization scheme, independent of issues related to generating a diverse set of feature descriptors and side information on a large vision dataset, I performed experiments to evaluate if the method can reliably process a Gaussian Mixture Model. I ran the model on mixtures comprising of 10^6 and 10^8 examples distributed concurrently across (up to 36) heterogeneous CPU cores. For $|\mathcal{K}| = 1024$, this

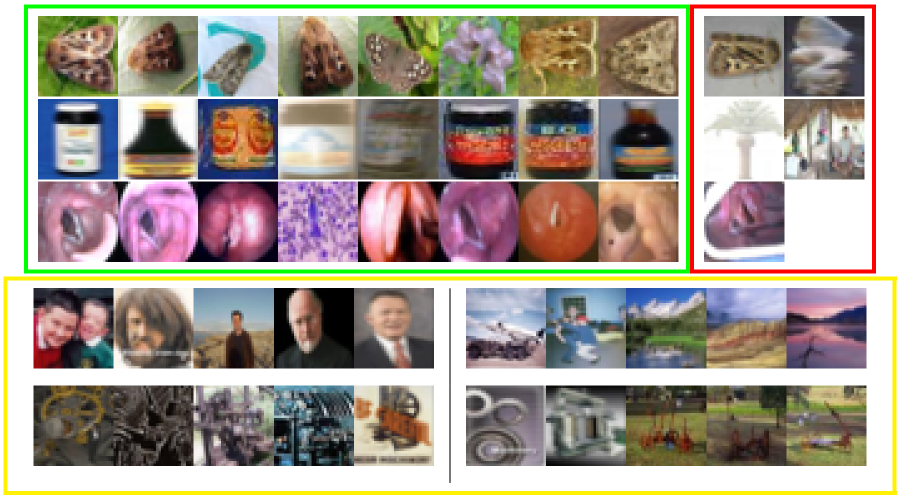


Figure 4.3: Results of my method applied to the TinyImages dataset, looking at how five selected keywords (top to bottom: `antler_moth`, `cassareep`, `true_vocal_cord`, `john`, and `machinery`) are split by the clustering. To the left of the divide is a sampling of images for which Spectral Clustering produces a "dominant" label for this keyword, and the rightmost columns are given the "wrong" label. Green and red boxes mark these groups for the three keywords shown for which the clustering achieved a good separation. The keywords in the yellow box do not have an informative cluster.

setup computed iterations at a rate of one iteration every 0.016 seconds and 0.034 seconds respectively. Within 50 000 iterations, the 10^6 case reaches an objective value of 0.054 with an NMI of 0.769 against the true label. The true label for each point is taken to be the choice of which Gaussian distribution from which that point it sampled. On the 10^8 case, an NMI of 0.683 was seen with the objective reduced to 2.685.

4.6.4 Model Characteristics

Varying $|\mathcal{K}|$ and Number of Iterations: The size of $\mathcal{K}$, the number of examples chosen in sampling in each iteration, is a key parameter in my approach. To show how this choice impacts the performance of the model, I use five kernels chosen from the Caltech101 experiments as our views. The kernels and the initialization are kept fixed in different runs, whereas $|\mathcal{K}|$ and the number of iterations are varied from 100 to 1000, and the objective is shown in Figure 4.5. The objective is progressively lower for increasing values of $|\mathcal{K}|$, and the iterations converge sooner with increasing values, as it approaches the full gradient. The rate of change in the objective as a function of iterations is similar for $|\mathcal{K}| \geq 300$, which suggests that a relatively small value should suffice.

Comparison of Projection and Manifold Optimization Techniques: I compare the manifold optimization of Section 4.3.1 and the method using projection on synthetic data. These use a single normalized Laplacian of a random graph over $n = 10^5$ points in four clusters. All three methods are applied to solve (4.2). As we can see from Figure 4.4, ordinary gradient descent converges in the fewest iterations due to using the entire matrix and $O(n^2)$ computations in *each* iteration. The manifold optimization method converges faster than projection in part because of heuristics used in selecting the step size. Further, the convergence rate increases when the sample size is increased. The Lanczos method (i.e., MATLAB's `eigs`) fails due to excessive memory requirements ($>$ 32GB).

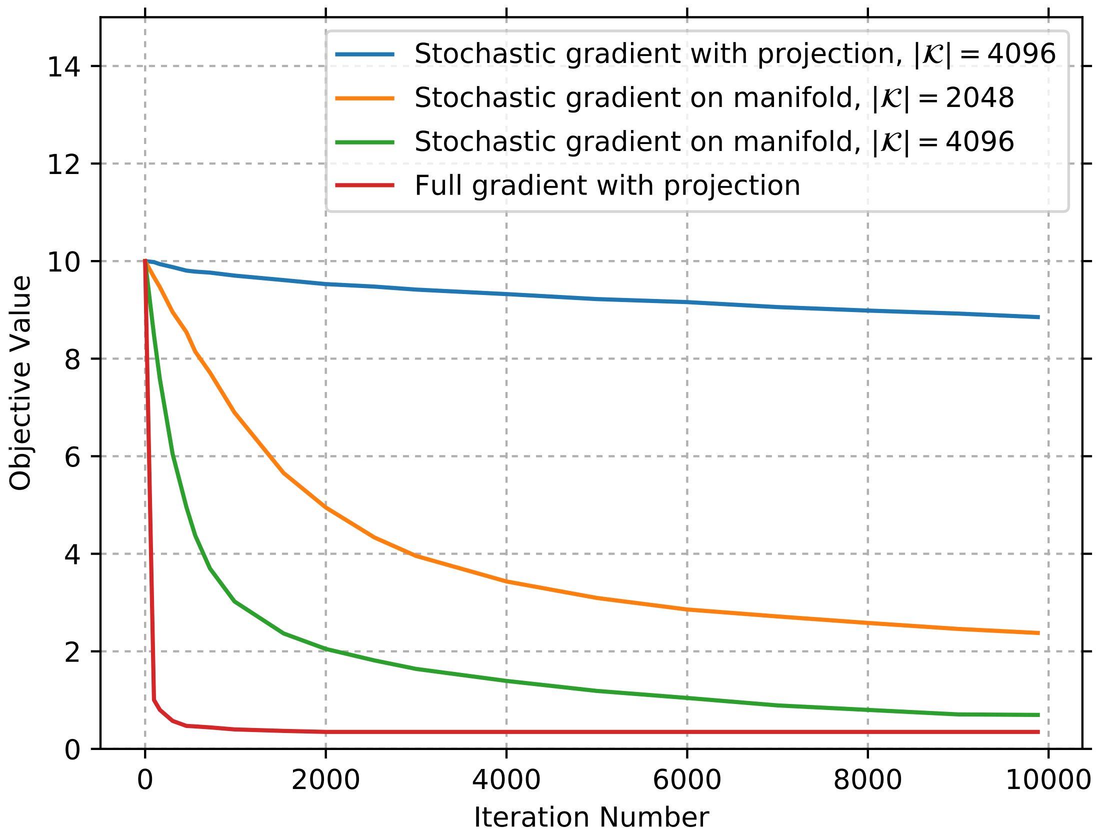


Figure 4.4: Plot showing the convergence rate of ordinary gradient descent with projection onto the Stiefel manifold, stochastic gradient descent with projection, and stochastic gradient descent using manifold optimization.

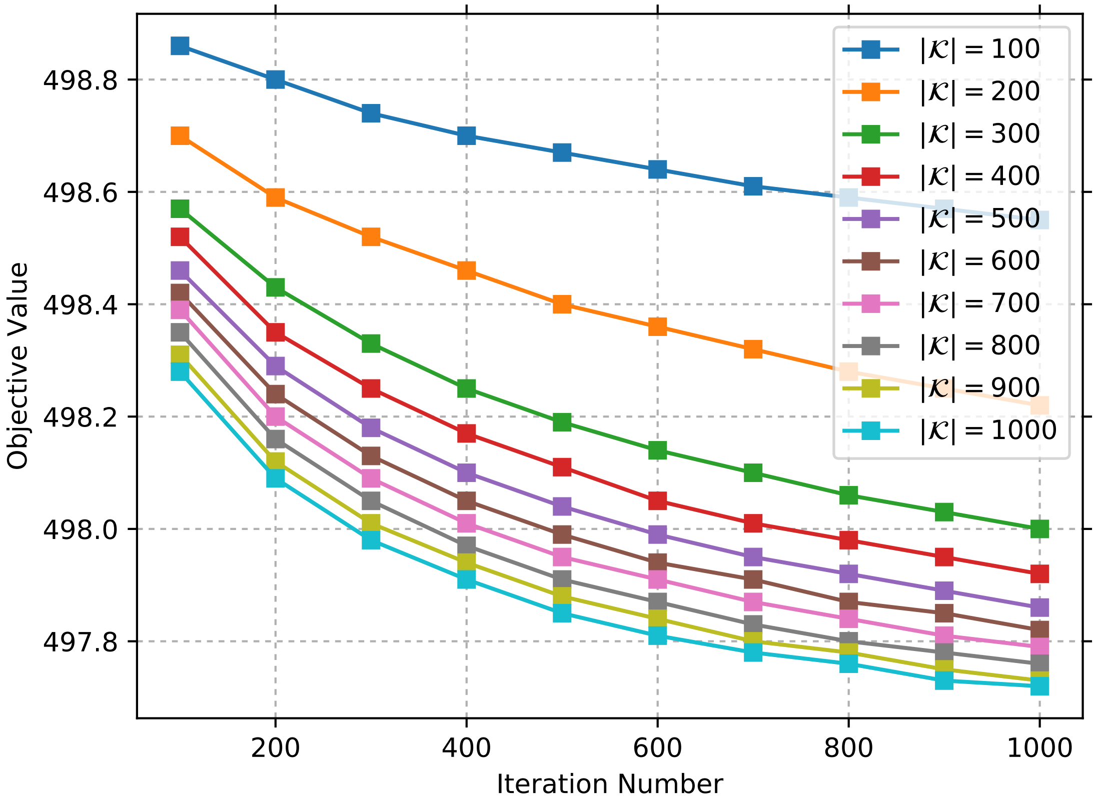


Figure 4.5: Plot showing the variation in objective with increasing iterations and different values of $|\mathcal{K}|$ from 100 to 1000. The objective value is drawn against iterations, demonstrating that quicker convergence is achieved with larger samples at the expense of increasing per-iteration cost.

Chapter 5

Memory-bounded CNNs

5.1 Introduction

Over the last few years, high capacity models such as deep Convolutional Neural Networks (CNNs) have been used to produce state-of-the-art results for a wide variety of vision problems including image classification [73], object detection [78], and segmentation [119]. This success has been in part attributed to the feasibility of training models with a large number of parameters and the availability of large training datasets [41], [121]. Training can scale this way due to greater computational power and refinements in the optimization algorithms used [14].

The need for techniques that can enable larger models is why the wave of recent work in CNNs was preceded by key improvements in the methods and software used to train Deep Learning models. These improvements made it possible to optimize objective functions defined over millions of parameters [40], [174]. These techniques, however, require careful tuning of the optimization and initialization hyperparameters to ensure that the training procedure arrives at a reasonable model [102]. In addition, simply performing the computations required for so many parameters has required leveraging high-performance parallel architectures such as GPUs [86], [102], or distributed clusters [40].

While the capacity of such deep models allows them to learn sophisticated mappings, it also introduces the need for good regularization techniques. Furthermore, they suffer from high memory cost at deployment time due to large model sizes. The earliest big deep models used regularization techniques like weight decay and DropOut [102]. However, these and other approaches [103] proposed in the literature do not address the issue of high memory cost, which is particularly problematic in models that rely on multiple fully connected layers. For example, one of the largest released models, the 19-layer VGG network, has 548MB of weights stored in the `caffemodel` format. Even the more recent 152-layer ResNet [73], which makes some model architecture decisions to be less wasteful in parameter count, still has a weight file of 231MB. The high memory cost becomes especially important when deploying in application in computationally constrained architectures like mobile devices [31], [77]. Accordingly, factorization [84], quantization [59], and other architecture and implementation changes are applied to reduce the parameter count and the cost of storing these parameters.

To address this problem, this chapter describes the use of sparsity-inducing regularizers for CNNs. These regularizers encourage that fewer connections in the convolution and fully connected layers take non-zero values and in effect result in sparse connectivity between hidden units in the deep network. In doing so, the regularizers not only restrict the model capacity but also reduce the runtime memory cost involved in deploying the learned CNNs. This is extended to group-sparse penalties that further reduce the overhead of sparse storage of the weights, while yielding roughly the same trade-off of memory cost vs training and test accuracy as unstructured sparsity.

This chapter will additionally show the performance of these regularizers as applied to networks trained on the MNIST, CIFAR, and ImageNet datasets. The results show that one can generate models that reduce the memory consumption at test time by a factor of three or more with minimal loss in accuracy. I then show how this can be used to improve the accuracy of vision classifiers using ensembles of deep networks *without* incurring the greater memory costs that would ordinarily result from having to store multiple models. Using sparsity regularization in this way significantly improves upon more typical ways researchers seek to limit model complexity, for example by changing the network topology by removing units or neurons. Finally, I show how the regularized training objectives can be efficiently optimized using stochastic gradient descent.

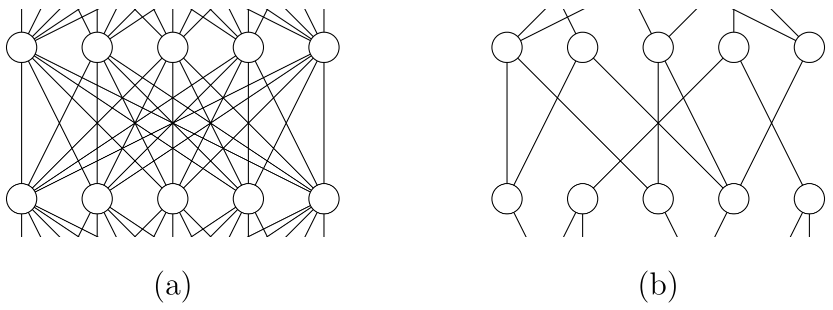


Figure 5.1: Illustration of the sparsity/pruning strategy to reduce the weight storage cost of a CNN. In the case of a densely connected neural network, as in (a), each node in the top layer connects to one at the bottom. Weights must be learned and stored for each connection. In a sparse network, as in (b), weights must only be stored for the connections that are present.

To summarize, this chapter will show a set of sparsity-inducing regularization functions, including group-sparse constraints, that I demonstrate are effective at reducing model complexity with no or minimal reduction in accuracy. Updates for these regularizations that are easily implemented within standard existing stochastic-gradient-based deep network training algorithms. This chapter concludes with empirical validation of the effect of sparsity on CNNs on CIFAR, MNIST, and ImageNet datasets.

5.1.1 Related Work

Regularization of Neural Networks. Weight decay was one of the first techniques for regularization of neural networks and was shown to significantly improve the generalization of neural networks by Krogh and Hertz [103]. It continues to be a key component of the training for state-of-the-art Deep Learning methods [102]. Weight decay works by reducing the magnitude of the weight vector of the network. It yields a simple additional term to the weight updates at each iteration of the learning procedure, assigning to each individual weight as $w_i \leftarrow w_i - \lambda w_i$. The update term used is the gradient of the squared ℓ_2-norm of the weights.

An interesting observation is that the training procedure for a linear perceptron with weight decay is equivalent to learning a linear SVM using stochastic gradient descent [166], where the weight decay is serving as the update to maximize the margin. Taking the typical SVM optimization over a collection of examples $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N) \in \mathbb{R}^d \times \{-1, 1\}$, it optimizes a linear separator $(\mathbf{w}, b)$:

$$\min_{\mathbf{w},b} C\|\mathbf{w}\|_2 + \frac{1}{N}\sum_{i=1}^{n}\left(1 - y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle - b)\right)_+ . \tag{5.1}$$

It can be shown (see [11]) that the smallest gap between two examples that will not be penalized by the loss term, twice the "margin," is $\frac{2}{\|\mathbf{w}\|}$. Hence the regularization $C\|\mathbf{w}\|_2$ serves to maximize this margin. Optimizing (5.1) by stochastic gradient descent, one can see that the gradient of this regularization is a scalar multiple of $\mathbf{w}$ itself, and subtracting this gradient will take the form $\mathbf{w} \leftarrow \mathbf{w} - \lambda\mathbf{w}$, precisely the update that is applied to a neural net in weight decay.

While not seen in CNNs or Computer Vision application, building sparse networks was considered for more classical neural networks. These were grouped into "pruning" methods [151], and include both regularization penalties and techniques for determining the importance of a given weight to the network's accuracy.

Hinton et al. [74] proposed a regularization technique known as *dropout*, which now forms a basis for many state-of-the-art deep neural network models [102], [120], [175]. During training, some portion of the units in the network is "dropped:" their output is fixed to zero and their weights are not updated during back-propagation. In each SGD iteration, a different random subset of the units is selected to be dropped. The outputs of these units are then multiplied by the dropout factor at test time. This is done in part as a computationally cheaper approximation to training an ensemble of sparse networks. The sparse networks implicitly created by dropout are simplified in a different way than the parameter regularization considered here. Dropout eliminates entire units or neurons, while I instead seek to reduce the number of parameters of those units. It also does so in a way such that the same number of weights still need to be stored for inference at test time, since this sparsification is only applied during training. Rather than having a smaller number of complex units, this work tries to train models that have a constant number of simpler units.

Comparison to Model Compression. A further method of constructing simpler models is the technique of *model compression* [23]. Model compression relies on the availability of large amounts of unlabeled data, using it to build smaller and computationally cheaper models by teaching the compressed model from the output of a larger and more parameter-heavy model that has been trained to fit the target task. This data is labelled using the original network and then used to train a smaller network. The idea behind this strategy is to make sure that the smaller network does not have to worry about regularization. Furthermore, unlike the method in this chapter where sparsity is enforced explicitly through the use of an ℓ_0 or ℓ_1 norm on a large number of hidden units, the model compression scheme simply trains a network with fewer hidden units. The method presented in the following chapter does, though, target a similar setting, in which plenty of computational resources are available at training time; however I wish to use these to train a model that is suitable for deployment in a more constrained environment.

Note that some recent works [43], [84], [209] have considered the same task but have adopted a different approach. They build networks with a set of *low rank* filters in each layer. These are built after training a network without this constraint, where the simpler network is selected to approximate the original full-rank network. Another approach to reducing the memory cost of the network parameters is to reduce the numerical precision of each parameter value. This can also be done during training, optimizing directly over the reduced-precision weights [64].

Sparse Autoencoders. Extensive work has also been done on training deep models that learn sparse *representations* of the data, while the learned parameters are themselves non-sparse. In an early work, Olshausen and Field [143] used a basic neural network structure inspired by the V1 layer of the visual cortex to learn sparse representations of a set of images. More recent work has used the term *sparse autoencoder* to describe this type of structure, and in Deep Learning, autoencoders have been used to initialize multi-layer networks as a method of layer-wise pre-training [190]. Indeed, much recent work on designing novel architectures for Deep Learning for vision application seeks to reduce or sparsify the parameter set. Convolutional layers themselves, in addition to enforcing an assumption of translation invariance, are meant in part to simply reduce the number of parameters [111]. A convolution *can* be represented by a fully connected layer, though it is unlikely to learn this mapping with a finite amount of training data and computation time. Some prior work has investigated automatically determining CNN structure, though instead by searching the space of layers and their sizes that may be used in place of simpler higher-parameter-count choices [170].

5.2 Convolutional Neural Nets

Convolutional Neural Nets are a variety of Deep Learning methods in which *convolutions* form the early layers. These have become the standard technique for performing Deep Learning for Computer Vision problems, as they explicitly deal with vision-based primitives. In the first of these layers, learned filters are convolved with the input image. The output of each of these kernels, composed with a nonlinear mapping, is then convolved with another set of filters in the next layer, and this is repeated for each subsequent convolution layer.

CNNs are frequently augmented with final "fully connected" layers that are more similar to classical neural networks, to provide a mapping from the features learned by the convolutional layers to the output labels. In most state-of-the-art networks, the fully connected layers are responsible for the majority of the parameter count of the network, though some recent work has considered models consisting entirely of convolutional layers [120] or has converted the fully connected layers into a convolution [164]. However, the state of the art, as the start of this work [72], does train with fully connected layers at the end of the network.

5.2.1 Optimization Perspective on CNN Training

The standard methods for training Deep Learning models may be expressed as stochastic gradient descent on a given loss function between the output prediction and training labels. The training procedure, given a set of inputs $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ and the corresponding ground truth labels $y_1, y_2, ..., y_n$, determines the choice of learned parameters W for all layers. Consider the network itself as a function f of the input data and W. The overall optimization problem on these weights may then be posed as follows:

$$\min_W \left(\mathcal{O}(W) := \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f(\mathbf{x}_i, W)) + \lambda r(W) \right). \tag{5.2}$$

Here, $\mathcal{L}$ is a loss function between the true labels y_i and the predictions of the network $f(\mathbf{x}_i, W)$ on the i^{th} training example. Taking the sum over the training data gives the empirical risk. The function r is the *regularization* term, with a weighting hyperparameter λ, which seeks to reduce the hypothesis space. For ordinary weight decay, this will be a squared ℓ_2 norm: $r(W) = \|W\|_2^2$.

The objective in (5.2), along with its gradients, is very expensive to compute for practical problems. The state of the art in Computer Vision considers very large networks with up to 1001 layers [73] and 60 million parameters [102]. Further, these models are trained on large datasets such as ImageNet, with the 2012-2014 classification challenge set having 1.2 million training images. The optimization of this objective is made far more computationally tractable by using *stochastic gradient descent* [14] or stochastic variants of adaptive and accelerated gradient methods [174].

5.3 Regularization Updates

I propose, as the central contribution of this chapter, to encourage sparsity in the networks by applying simple updates to the set of weights in each layer during training. First, though, consider the regularization functions themselves. Regularization often uses a norm function on the parameters of the model. Define the ℓ_p vector norms as $\|x\|_p = \left(\sum_i |x_i|^p\right)^{1/p}$. This

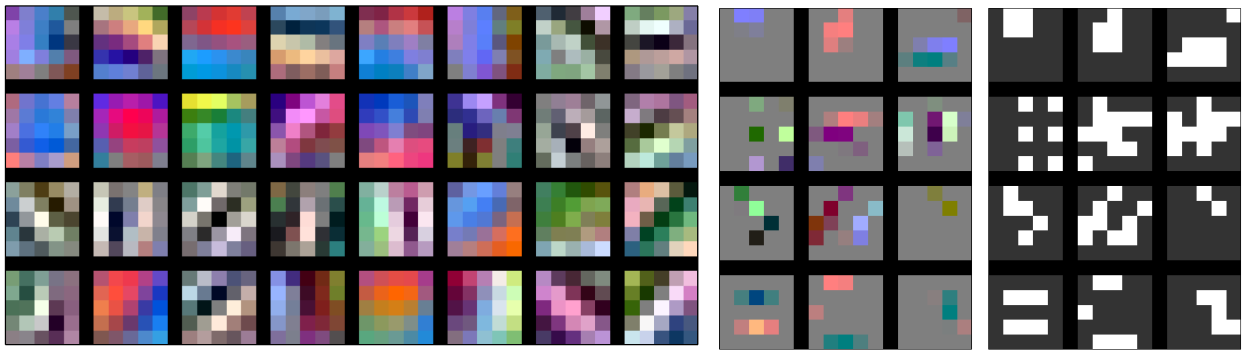


Figure 5.2: Convolution filters learned on the CIFAR-10 classification task. The non-sparse kernels on the left come from a baseline model using classical weight decay as regularization. Incorporating a sparsity-inducing ℓ_1 shrinkage during training yields the sparse filters on the middle right and 20 all-zero filters not shown. The pixel-wise nonzero pattern of the sparse filters is shown on the far right.

definition is typically extended to ℓ_0 and ℓ_∞ norms by taking limits on p. Of special interest to sparsity regularization is the ℓ_0 "norm," the count of the number of nonzeros of x.

5.3.1 ℓ_1 Regularization and the Shrinkage Operator

The most common classical technique for learning sparse models in Machine Learning is ℓ_1 regularization [177]. The ℓ_1 norm of a vector is the tightest convex relaxation of the ℓ_0 norm. It has been shown for some classes of Machine Learning models that regularization terms consisting of an ℓ_1 norm can provide a provably tight approximation, or even an exact solution, to a corresponding ℓ_0 regularization that directly penalizes or constrains the number of nonzero parameters of the model [46].

One can optimize an ℓ_1 regularization term by updating the weights along a *subgradient* of the ℓ_1 norm. A negative multiple of a subgradient gives a descent direction, and updating the weights a sufficiently small distance along this ray will reduce the regularization term. Its sum with the gradient of the loss terms in (5.2) gives a subgradient of the whole objective. Since the ℓ_1 norm is differentiable almost everywhere, and therefore the subdifferential is a singleton set, it is not even usually necessary in practice in CNN training to choose a particular subgradient. For comparison, the ReLU operator [124], one of the most commonly used components in current deep neural networks (DNNs), tends to be implemented in most codes by simply taking a fixed choice of gradient at the nondifferentiable point. Therefore, the choice of subgradient I consider below is simply the sign operator, applied element-wise to the vector of weights. It can be seen that $\text{sign}(W) \in \partial r(W)$ for $r(W) = \|W\|_1$. The resulting update is therefore, for some $\delta > 0$, the element-wise operator:

$$W_i \leftarrow W_i - \delta \text{sign}(W_i). \tag{5.3}$$

This ℓ_1 update is currently implemented in Caffe [86] as a type of weight decay that can be applied to the whole network.

This update has the shortcoming that, while it will produce a large number of weights that are very *near* zero, it will almost never output weights that are exactly zero. Since later layers can still receive and learn to magnify input based on the resulting small nonzero weights, and earlier layers will receive back-propagated gradients along these weights, these near-zero parameters cannot be ignored. When seeking to construct a sparse model, the natural technique is to try to threshold away these very small weights that an optimal ℓ_1 solution will have set to zero. Finding a good threshold for this heuristic requires some attention to the schedule of learning weights and the level of regularization relative to the gradients near the end of training.

A regularization step with better empirical and theoretical properties is the shrinkage operator [177]:

$$W_i \leftarrow (|W_i| - \delta)_+ \text{sign}(W_i). \tag{5.4}$$

Here the notation x_+ refers to the positive component of a scalar x. The shrinkage operator is among the oldest techniques for sparsity-inducing regularization. A key property for deep network training is that this operator will not allow weights to change sign and "overshoot" zero in an update. The operator will output zero weights rather than small weights oscillating in sign at each iteration. For a suitable choice of δ, it eliminates the need to consider thresholding. Neighboring layers will get zero input/back-propagation along that connection, so they can learn on the zero weight.

This shrinkage update is an example of a *proximal mapping*, also called a "proximal operator," and alternating this with descent steps along the gradient of the loss would yield a *proximal gradient* method. These methods generalize and extend the LASSO, and have been applied to group sparsity regularizations [200]. Proximal mappings are also effective when using approximate or stochastic gradients; Schmidt et al. [160] provide convergence guarantees in the case of a convex problem.

5.3.2 Projection to ℓ_0 Balls

While ℓ_1 regularization is known to induce or encourage sparsity in common Machine Learning models, the direct way to construct sparse models is to consider the ℓ_0 norm. This, importantly, deals directly with the count of the number of nonzeros. That count determines the memory cost when using sparse formats, regardless of the magnitude or value of the nonzero elements.

A simple regularization operator can be used to train models under ℓ_0 norm constraints. This update, in every n iterations, will set to zero all but the t largest-magnitude elements of the parameter vector. This imposes the hard constraint that $\|W\|_0 \leq t$ for some integer t. This operator can be seen as a projection onto an ℓ_0 ball, namely:

$$\mathcal{P}_0(W,t) = \begin{pmatrix} \arg\min\limits_{W'} & \|W - W'\|_2^2 \\ \text{s.t.} & \|W'\|_0 \leq t \end{pmatrix}. \tag{5.5}$$

This "projection" matches the operator of hard thresholding all but the t least-magnitude elements, as the quantity the projection seeks to minimize in (5.5) is the total squared magnitude of the elements that are zero in the optimal W' but not in W. In compressed sensing and M-estimators, this procedure is called "Iterative Hard Thresholding" [12], [85].

This ℓ_0-regularization update can be seen as an instance of "projected gradient" methods frequently used in convex optimization, and the extension to group sparsity below is derived similarly. A theoretical analysis of projected gradient algorithms can be gleaned

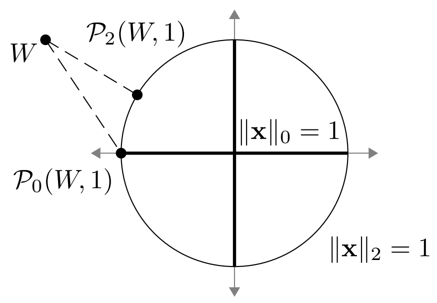
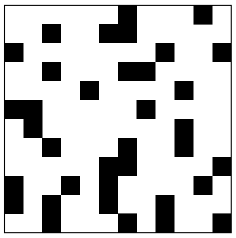


Figure 5.3: Illustration of the "ℓ_0 projection" done by hard thresholding. The projection operator is defined in (5.5). This suggests an analogy between iterative hard thresholding and projected stochastic gradient descent (SGD).

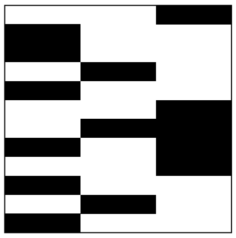
from the more general analysis on proximal operators. Projection onto a convex set P is the proximal operator for an extended-value function that maps P to 0 and its complement to ∞. Therefore, projection onto a convex feasible set can be seen as a special case of proximal stochastic gradient methods. See [150] for discussion specifically on projection within stochastic gradient methods.

5.3.3 Group Sparsity

The increased complexity of sparse computation can lead to a number of computational drawbacks. Computing the layer responses in an implementation that stores the parameters in a sparse storage format will suffer from poor memory locality and be less easily parallelizable. Typically, sparse computations will be limited by memory bandwidth [9]. Improved computational performance can be achieved by performing computation on *blocks* of parameters that correspond to operations on entries in the response map that are adjacent in memory or can be executed in parallel. This is referred to in [82] as "blocking," cited in [9] as achieving greater performance even in GPU-parallelized sparse matrix computations. One can also see that given the same number of total nonzeros, a block-sparse representation will require less overhead for sparse storage, as one only need store the indices of the nonzero blocks alongside the weights rather than the indices of each individual nonzero weight.



(a)



(b)

Figure 5.4: A matrix with nonzero pattern given by the shaded entries in (a) has scattered sparsity. The matrix in (b) shows *group* sparsity. Group sparsity can better leverage SIMD hardware for both efficient and memory-bound computation.

In a CNN setting, rather than simply filling in the zeros within a block, the network can be allowed to use all the weights in the blocks, learning nonzero values for the whole block. This can be done by creating the blocks of nonzeros using a *group* sparse regularization [205] with groups corresponding to computationally efficient blocks. "Groups" here uses terminology from group-sparse regularization, and is distinct from the grouping of convolutional layers done to achieve cross-GPU parallelization in Krizhevsky et al. [102].

Let $W = [w_{fcij}] \in \mathbb{R}^{F \times C \times K_1 \times K_2}$ be the 4D tensor of weights in a convolutional layer of the network. This parameterizes a set of $K_1 \times K_2$ convolutions, over C input channels producing F output channels. A good accuracy vs. sparsity trade-off can be achieved by grouping over the spatial dimensions. This is achieved by imposing, during training, a constraint on the number of groups that contain a nonzero:

$$\left(\sum_{f=1}^{F} \sum_{c=1}^{C} \mathbb{I}[w_{fcij} \neq 0 \text{ for any } i = 1, ..., K_1 \text{ and } j = 1, ..., K_2] \right) \leq t. \tag{5.6}$$

The corresponding update can also be derived as a projection onto the set of weights feasible w.r.t. to this constraint. By the same reasoning as the iterative hard thresholding update, here the update will set to zero all entries in W except those in the t groups with greatest ℓ_2 norm.

5.4 Implementation

I implemented the regularization updates presented above as a modification to Caffe [86]. The ℓ_1 update matches the one already present, though that implementation was extended to allow mixing different types of regularization. For example, in Figure 5.6, one can retain ordinary weight decay for all but select portions of the model in order to analyze the effect of sparsity on those particular parts. The experiments below use the baseline networks provided with that distribution, LeNet [111] and CIFAR-10 Quick [170], as a target for regularization.

For larger-scale experiments, this chapter demonstrates the usefulness of these regularizers on a network developed for the ImageNet classification challenge [102]. This network has five convolutional layers and three fully connected layers, with a total of 60 million parameters. Upon the publication of [102], it was one of the largest neural networks described in the literature. Due to the much larger scale of both the dataset and the network, we perform experiments using a computationally cheaper "fine-tuning" procedure in which we use an existing non-sparse network as the initialization for sparsity-regularized training. The starting point for this training was a Caffe-based [86] duplication of the "AlexNet" network. The network was progressively sparsified in stages over 200 000 iterations, taking approximately one week on a GeForce GTX TITAN. Thresholding/ℓ_0 projection was done every 100 iterations. The number of nonzeros remaining after the thresholding was manually reduced in steps, tightening the sparsity constraint each time the ℓ_0-constrained training converged on a network with comparable accuracy to the original dense weights.

Optimization with the ℓ_0-norm projection was seen to be fairly robust, and it posed little additional difficulty to find settings for which the ℓ_0-regularized training would converge. I did not observe any case where a network with baseline regularization was successfully trained but an ℓ_0-regularized variant of the model failed to find a model reasonably close the optimum w.r.t. the training loss. The high test accuracies for very sparse models seen in this work's other experiments also suggest that the optimization was successful in finding a reasonable fit to the training data in those cases.

5.4.1 Layer-wise Distribution of Sparsity

It was observed on standard experimental networks that each layer of the network performs very differently when varying the level of sparsity-inducing regularization. Figure 5.6 shows that significant decreases in accuracy are seen when sparsifying the first convolutional layer at nonzero ratios for which the fully connected layers are still able to describe the target concept. A key quantity to analyze, as is done on the right plot of Figure 5.5, is this nonzero ratio defined as the ratio between the number of nonzero weights in a layer and the total number of parameters. The distribution of sparsity between different layers is the hyperparameter with the greatest effect on the performance of sparse deep models. The procedure below allows us to automatically determine sparsity hyperparameters incorporating some of the intuitions given by the layer-specific results in Figure 5.6.

The choice of which layers to sparsify was made by doing a greedy search, progressively reducing the number of nonzeros while maximizing the accuracy on a validation set. The procedure is to repeat the following two steps:

1. For each layer, reduce the number of nonzeros by 20%, and train a network.
2. Take the network from the previous step that produces the best validation accuracy, and use this network as the start of the next iteration.

For a baseline CIFAR-10 network this procedure yielded nonzero distributions skewed towards sparsifying the later layers of the network. The level of sparsity, and the number of nonzero parameters, chosen by this procedure for each layer of the best candidate networks is shown in Figure 5.5. From the normalized plot on the right of Figure 5.5, it is clear that the tightest sparsity constraints were imposed on the final convolution layer and the fully connected layers, while the first two convolution layers were relatively untouched. In terms of the number of parameters set to zero, the greatest reduction in weights was in the final convolution and first fully connected layers. These networks significantly outperformed baselines that imposed sparsity constraints uniformly across all layers.

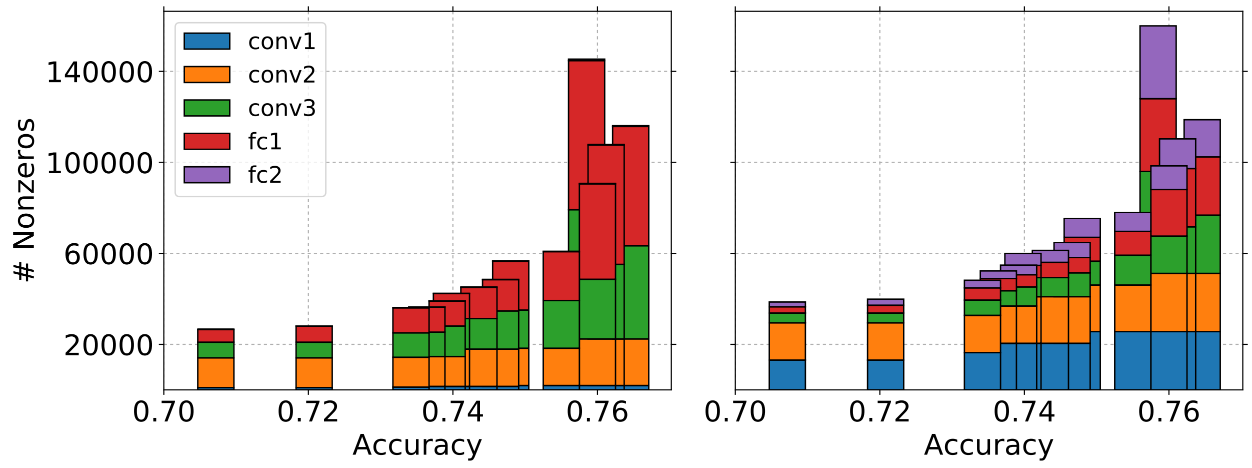


Figure 5.5: These plots show the distribution of nonzero parameters determined by a greedy procedure seeking to maximize the accuracy of sparse networks for CIFAR-10. Each stack of boxes corresponds to a single network and is centered on the accuracy for that network. The plot on the left directly counts the number of nonzeros in the layer. The plot on the right shows the same networks, but normalizes the heights such that each layer's box would be the same height for a dense network. "Conv1-3" are the convolution layers, while "fc1-2" are the fully connected layers.

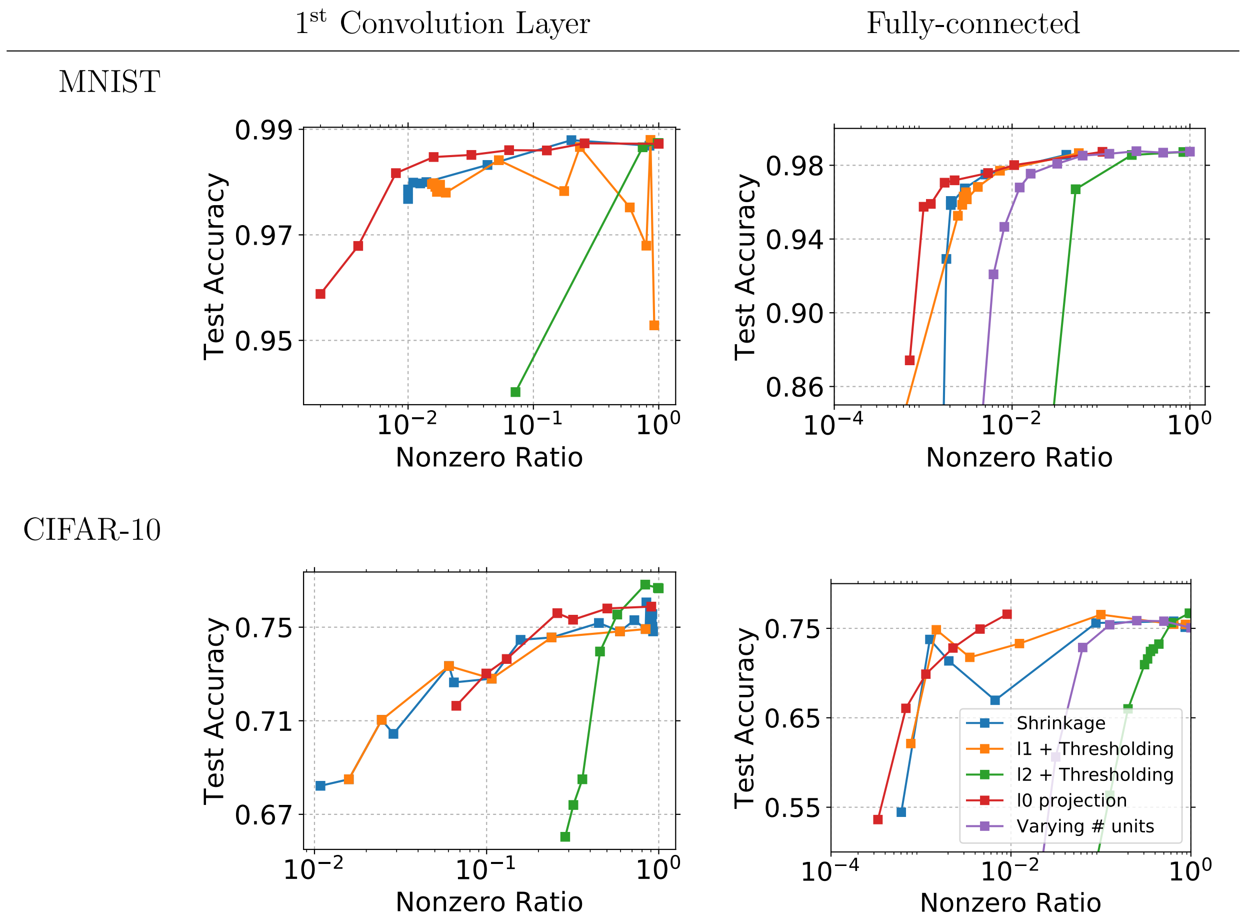


Figure 5.6: Sparsity vs. accuracy when sparsity regularizations are imposed only on specific layers of the MNIST and CIFAR-10 baseline networks. As baselines, the plots also show the results of both thresholding the weights learned under ℓ_2-regularization and reducing the number of hidden units in the first fully connected layer.

5.5 Experiments

The experiments below use MNIST [111], CIFAR-10 [101], and ImageNet [41] as benchmarks and show the effect of sparsity on widely-distributed baseline CNNs for these datasets.

MNIST is a set of handwritten digits, with 60 000 training examples and 10 000 test examples. Each image is 28×28 with a single channel. MNIST is among the most commonly used datasets in Machine Learning, Computer Vision, and deep networks.

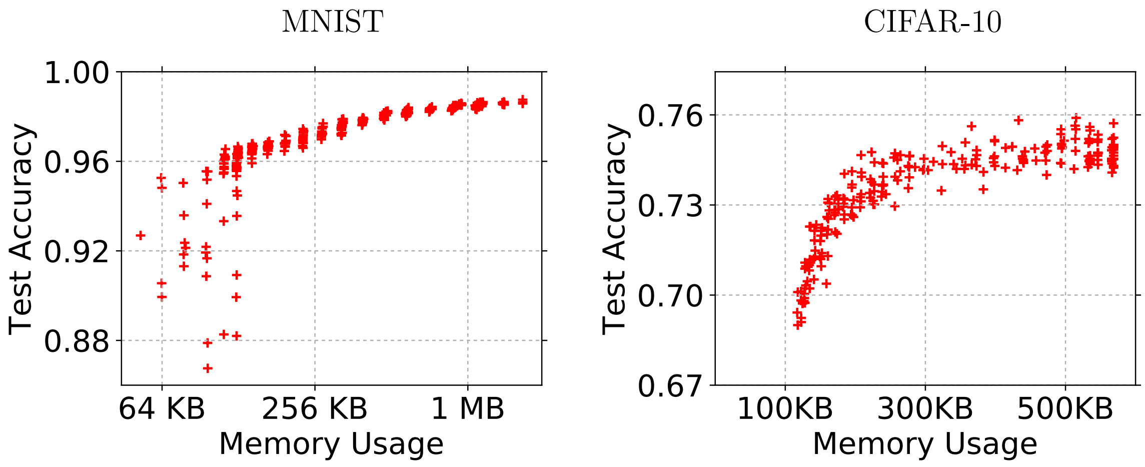
CIFAR-10 is a subset of the "80 million tiny images" dataset [178] where ground truth class labels have been provided. There are 10 classes, with 5 000 training images and 1 000 test images per class. Each image is RGB, with 32×32 pixels. Other than subtracting the mean of the training set, this implementation did not consider whitening or data augmentation, obtaining higher error rates but allowing comparison against an easily reproducible baseline with standard codes.

The AlexNet fine-tuning experiments used the ILSVRC 2012 training set. Simple data augmentations using random crops and mirroring were used in the training phase, along with subtracting the mean image from the training set.

5.5.1 Accuracy and Regularization Updates

MNIST and CIFAR-10. A key empirical result of this work is that sparse models achieve surprisingly high accuracies even as the number of nonzero parameters gets quite small. Figure 5.6 shows limited experiments where particular layers of two baseline MNIST and CIFAR-10 models are sparsified with various penalties. It also shows, as baselines, schemes that either apply a threshold to ℓ_2-regularized models, or vary the network structure to directly reduce the number of parameters.

Note that, while the ℓ_0 projection is the regularization that imposes sparsity most directly, the ℓ_1-based regularizations perform comparably well for a "middle" range of regularization strength. This is as would be expected, based on the literature for sparsity-inducing regularization for shallow models. The ℓ_0 projection tends to outperform the ℓ_1 subgradient and shrinkage updates as the number of nonzeros begins to approach within an order of



ImageNet

Model	**Top-1**	**Top-5**	**Memory**
[102]	59.3%	81.8%	-
BVLC AlexNet [86]	57.28%	80.44%	233MB
Sparse (ours)	55.60%	80.40%	58MB

Figure 5.7: Exploring the trade-off between accuracy and model size achieved by our method for different problems. In the first row, each point plotted is a candidate network considered by the greedy search in Section 5.4.1. The x axis shows the memory required to store the weights of that network, using the best choice among common sparse storage formats. Below is listed the same for a network trained on ILSVRC 2012, as described in Section 5.4.

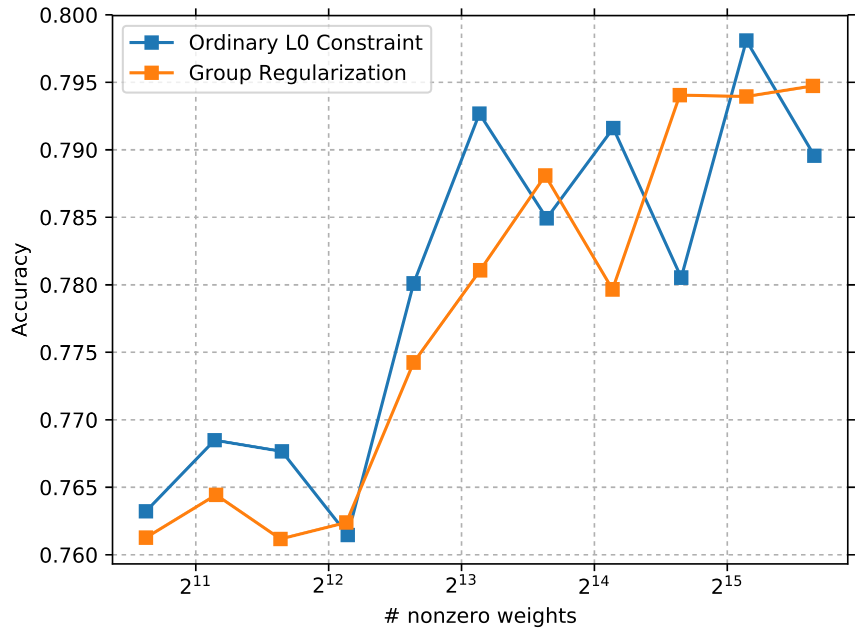
magnitude of the dense model. This can be explained as the result of the ℓ_1-based updates' effect on the *magnitude* of the regularized weights, in contrast to the ℓ_0 projection that does not at all modify the largest-magnitude weights. This distinction becomes more important as we have more smaller-magnitude weights in less sparse models. It is also less visible in MNIST as all models quickly reached a "ceiling" accuracy. In the other direction, the hard thresholding/ℓ_0 update seems to produce models of moderate accuracy at higher levels of sparsity. In this range, for higher regularization strength, the ℓ_1 updates also frequently fail to produce very sparse models at all, due to a discontinuity in the regularization path between the sparser models shown in 5.6 and models that are all zero.

Figure 5.8 also shows the comparison between this ℓ_0 update and its group sparse variant described in Section 5.3.3. Notably, this grouping achieves the same trade-off in accuracy vs. sparsity as the ungrouped method despite restricting the network to a particular sparsity pattern. This also shows that the ordinary thresholding update without grouping produces a network that is not necessarily sparse with respect to the number of nonzero groups; even very sparse lower-accuracy networks still have nonzeros in most of the blocks.

Most of these experiments primarily use the ℓ_0 projection because it requires far less intensive hyperparameter tuning. The different sparsity levels seen for the ℓ_1 norms are only seen in a very narrow range of values for the regularization multiplier. Outside this range one will get either dense models or models that are completely zero. By contrast, good models are produced for nearly any choice of the number of nonzeros imposed by an ℓ_0 constraint.

ImageNet. Therefore, the experiments with AlexNet, as applied to ImageNet, focus on the case of applying the ℓ_0-projection regularization, and only on the fully connected layers, as these are together responsible for 96% of the total number of weights in the network.

The top-one and top-five validation accuracies of the original networks, the Caffe duplication, and our sparse version of the Caffe duplication are shown in the table in Figure 5.7. Note that the Caffe duplication achieves slightly lower validation accuracies than the original



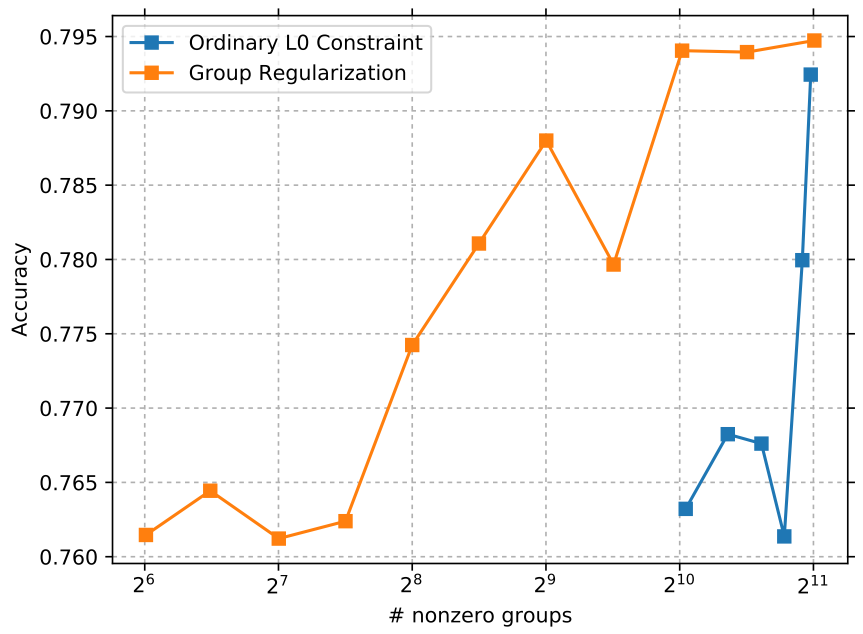


Figure 5.8: Comparing group sparsity and an ordinary ℓ_0 constraint on the third convolution layer of CIFAR-10 Quick. The top figure plots the accuracy against the total number of nonzero weights. Each point is a network trained with different constraints. The bottom figure plots the accuracy for the same networks against the number of nonzero weight *blocks*.

networks trained by Krizhevsky et al. Initially, only the final fully connected layer was sparsified and reduced to 400 000 nonzero weights out of 4 096 000 in the original dense network (plus 1 000 per-neuron biases). This is *ten times smaller* than the original parameter set of this layer in the dense network. In the second stage, the other two fully connected layers were also regularized to produce a network with 3 million parameters in each of these two layers (for 6 million total). This is from a total of 54.5 million weights in these two layers of the original dense network. Overall, the sparse network had all but 14% of the weights of the network set to zero. To compare, I also directly thresholded the network without additional training to have the same number of nonzeros in each layer. This yielded a network that achieves only 38.92% top-one validation accuracy and 63.44% top-five validation accuracy. The accuracies reported here are done without test time oversampling or any similar test time data augmentation methods.

5.5.2 Ensembles

The above results suggest that the very high parameter counts in Deep Learning models include a number of redundant weights. Much of the computational resources and model complexity incurred by these very large models is spent to yield relatively little benefit in terms of test time accuracy. Given a fixed budget of some computational resource such as memory, this is much better spent on more effective ways to increase accuracy. A good method for improving test time accuracy is building *ensembles* that combine the output of multiple models.

To test this, I constructed ensembles using *bagging* [21], where each member of the ensemble is trained on a random resampling of the training data. Table 5.1 shows the resulting accuracy as we grow the ensemble under a *parameter budget*. Each ensemble was constructed to maintain a set of nonzero weights that does not grow in size even when considering greater numbers of predictors in the ensemble. This can be done by sparsifying the individual elements of the ensemble with the regularizations presented above. This targets a setting such as a mobile device with limited memory, for which building ensembles

Nets	**Nonzeros/Net**	**Total**	**Test Accuracy**
1	145 578	145 578	75.85% ± 0.559%
2	71 770 (49.3%)	143 540	77.40% ± 0.192%
3	46 023 (31.6%)	138 069	77.18% ± 0.215%
4	36 333 (25.0%)	145 332	75.96% ± 0.116%
5	28 249 (19.4%)	141 245	74.60% ± 0.155%

Table 5.1: Building ensembles of sparse models under a parameter budget. The 1-model case was trained on the original dataset as a baseline, all others on bagged resampling. Accuracies given are means with standard deviations across multiple trials with different models and bagged datasets in each trial.

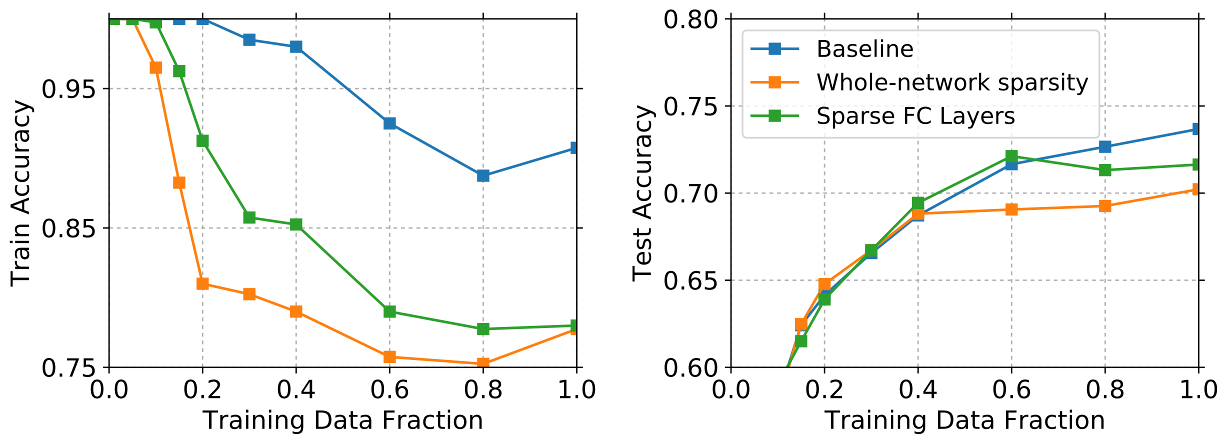


Figure 5.9: Training and test accuracy for dense and sparsity-regularized CNNs when varying the size of the training set by randomly subsampling CIFAR-10 as in Section 5.5.3.

with the full model is prohibitively expensive, which can be true quite quickly for CNNs. Using this as a proxy for model capacity and power, this experiment further allows trade-offs in how to build predictors, and where model capacity should be spent to get the best performance at deployment. To train an ensemble with n members, this procedure repeats the following steps n times:

1. Re-sample the training data, with replacement, to get another training set of the same size.

2. Take a layer-wise distribution of nonzeros given by the method in Section 5.4.1 that has $\leq 1/n$ total nonzeros.

3. Train a model on the re-sampled data, with each layer having an ℓ_0 constraint to enforce this distribution.

At test time, the above experiments ran each CNN as normal over the test dataset. To produce an end prediction from the whole ensemble, I average the output layers and predict the class corresponding to the largest average output.

The results show a significant increase in accuracy for smaller ensembles. As the size of the ensemble grows and the models become sparser, however, this yields diminishing returns, as the individual models can no longer sufficiently approximate the target task.

5.5.3 Reduced Training Data

A key property of regularization is its ability to improve the *generalizability* of a learned model. If there is insufficient training data to properly estimate the true underlying distribution, the minimizer of the empirical risk will be different from the best model for the expected risk under the true distribution. The model will then "over-fit" the training data rather than correctly learning the target concept.

The experiment with result shown in Figure 5.9 tested this assertion using models trained with sparse regularization and with ℓ_2 weight decay. I randomly subsampled the CIFAR-10 training dataset to produce a series of smaller datasets with increasingly insufficient training data. Two results can be seen from this experiment that match what Machine Learning theory predicts. First, the differences between simpler and more complex models become narrower with less training data, on the left-hand side of the plots. Indeed, the simpler models begin to outperform the more parameter-heavy dense models in some cases when little training data is available. Secondly, in all models, as more training data becomes available we see that the training accuracy decreases while the test accuracy on unseen data increases. The change in accuracy on both sets is much less pronounced on simpler models.

5.6 Memory Usage

To describe what the sparsity levels mean in concrete computation terms, consider three storage formats, each of which allows for fast computation with sparse networks. When calculating the memory usage of sparse networks, assume the optimal choice among the following formats:

1. **Dense:** One can ignore any sparsity present in the network and store it as with the original dense weights. For very sparse networks, a great deal of storage will be filled

with the number zero, but for networks with few zero weights, this will still be cheaper due to the additional overhead for required sparse data structures.

2. **Bit-mask:** One can also use a simple bit-mask with a number of bits equal to the number of total parameters. If and only if the bit corresponding to a parameter is one, then there the value of that parameter is nonzero. The nonzero parameters are then stored in a flat array. Additional indices into the flat array, depending on how computation is done with these weights, can allow this format to still be used directly in CNNs at runtime.

3. **Indexed:** Finally, one can use a scheme where only the nonzero parameters are stored, each one as a pair consisting of an index into the original weight array alongside the weight value. This is the traditional way to handle sparse vectors.

In some types of layers, more specific sparse formats such as Compressed Sparse Row (CSR) matrices [158] may also be suitable for computation. These will have a memory cost approximately equal to the "indexed" case.

The previous memory estimates assumed that the weights are stored as single-precision (32-bit/4-byte) floating-point values. The overhead introduced by the sparse data structures is relatively smaller for double-precision (64-bit/8-bytes) floating-point values. Additional formats, such as half-precision floats or fixed-point weights are not supported by standard hardware or CNN codes, but are increasingly seeing support in specialized mobile frameworks.

Using these formats, we plot in Figure 5.7 the memory required to store the weights of sparsified forms of the baseline test networks. Each point is a candidate network considered by a greedy search over the per-layer distribution of nonzeros. In the table in the same figure, we give the memory used for the "indexed" format on a sparse CNN for ImageNet.

5.7 Computation Time

As described earlier in this chapter, sparse computations can be slower than the corresponding dense computation. In highly optimized CNN implementations such as Caffe [86] and cuda-convnet [102], parallel hardware can be used to execute the large blocks of arithmetic in parallel. When the parameters of a convolution or inner product are sparse, however, the corresponding operations may be on elements of the layer's input or output responses that are no longer adjacent or regularly spaced in memory. As a result, the performance is instead bounded by random-access I/O rather than the number of floating-point operations required. With the additional overhead of sparse matrices, the result can be slower even if fewer operations are required.

Fortunately, the basic operations most commonly used in vision networks can still be made efficient for deployment on modern hardware, even when doing sparse computation. With the right implementation choices, the innermost loop of the computation in operations such as convolution will still be dense vector computations on responses adjacent in memory. This was demonstrated on a modified implementation of Caffe, that natively stores layer parameters as sparse coordinate list (COO) or compressed sparse row (CSR) tensors or matrices. The experimental implementation also does computations directly using these sparse parameters.

The core computation in Caffe's convolution implementation is a matrix-matrix multiplication between the convolution filters and a matrix constructed from the input image or response map. Denote this operation with $B \leftarrow WA$. A is the input response map, where each row is a channel of the input with some spatial offset. The matrix W contains the learned convolution filter bank, where each row is one filter in the bank, vectorized over the spatial and channel dimensions of the filter. Finally, B is the output response map. At test time, we take W to be a sparse matrix and compute the multiplication as:

```
$B \leftarrow 0$
**for** $i$, $j$ such that $W_{ij}$ is nonzero **do**
    $B_{i\cdot} \leftarrow B_{i\cdot} + W_{ij} A_{j\cdot}$
**end for**.
```

The inner computation here multiplies the j^{th} row of A by the scalar element W_{ij}, and adds the result to the i^{th} row of B.

Since A and B are still dense matrices, this is a dense vector operation (`axpy` in BLAS), and can be done efficiently in parallel, e.g. exploiting optimizations such as SIMD instructions. Therefore, even when using sparse weights in the network at test time, the innermost computation is done using dense linear algebra operations. The loss of parallelizability due to the sparsity of W incurs much less cost than a naïve implementation of sparse convolution.

Ultimately, though, the relative computation time between sparse and dense networks will depend on details of the network such as the sizes of the response maps at each layer, the size of the filter banks, and the number of images in a batch at test time. It will also vary significantly between architectures, as CPUs, GPUs, and lower-power processors will have differences in memory models and efficiency of vector processing. The results in this section were produced using CPU Caffe on an Intel Xeon E5520 at 2.27 GHz.

Figure 5.10 shows the computation time for a baseline CIFAR network used in the memory experiments. The experiment shown in that figure takes a number of networks trained with varying regularization parameters to achieve varying levels of sparsity. For very sparse

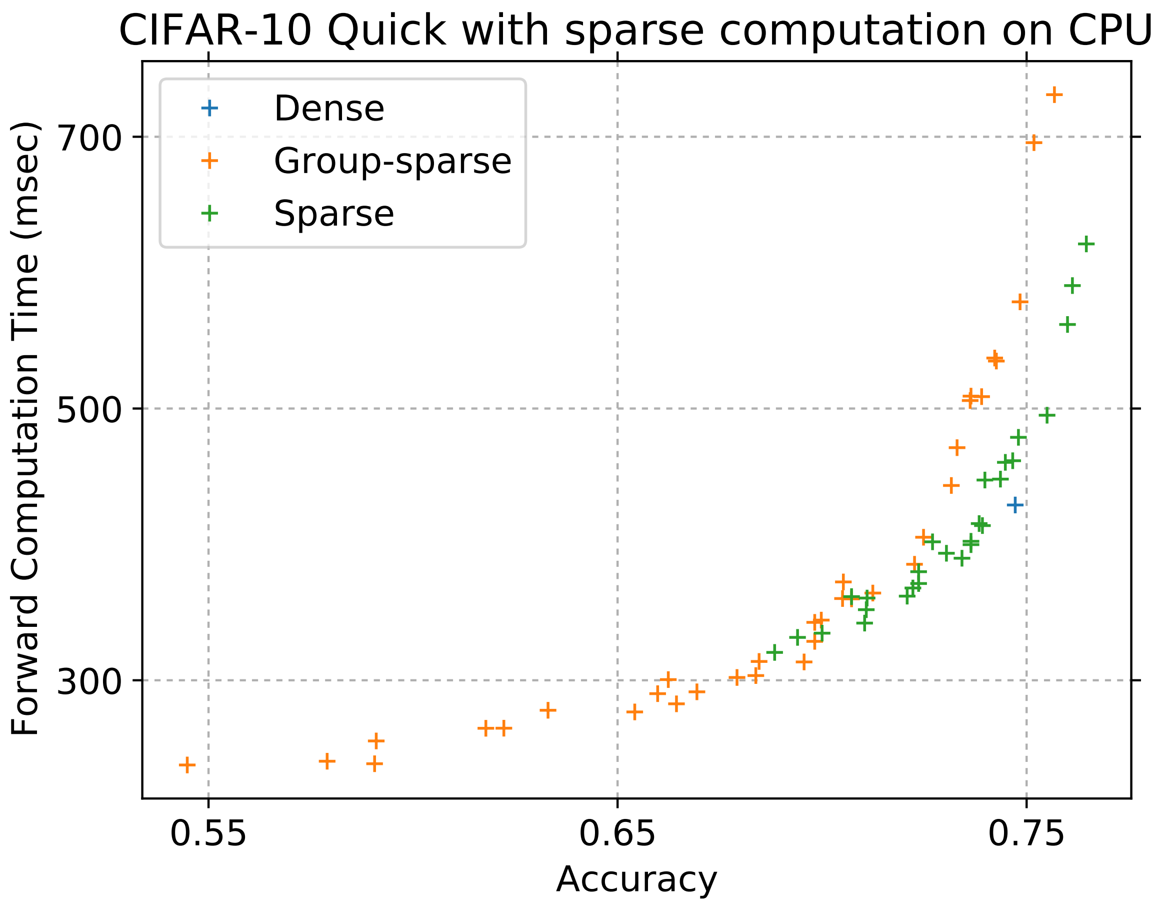


Figure 5.10: Showing the trade-off in computation time vs accuracy for CNNs trained with differing sparsity regularization. Each green or red marker is a network trained with differing sparsity constraints, while in blue is the accuracy and computation time for the baseline network trained without sparsity constraints and using ordinary dense computations.

networks, the results show a decrease in accuracy while requiring less computation time. Performing sparse computation on networks that are mostly dense, while achieving greater accuracy, proves to be slower than performing dense matrix multiplication after filling the parameter matrix with zeros. With a CPU implementation, roughly comparable performance can be achieved between the sparse and dense implementations. It is important to note, however, that the networks that perform most similarly to the dense network in this comparison are more *memory* efficient.

5.8 Summary

This chapter presents a scheme for pruning the weights of a deep neural network during training so that it uses significantly less memory during inference. The chapter presents and compares several different types of regularization and updates, including a hard-thresholding scheme that provides definite guarantees on the number of nonzeros in the resulting network. Using a group-wise sparsity pattern, additional overhead due to memory access patterns is reduced, allowing for networks that are compressed essentially for free.

These updates are also demonstrated as part of two meta-training schemes that use sparse neural networks as a building block. First, a greedy parameter search finds the optimal distribution of nonzero parameters layer by layer, providing a principled way to do a sort of architecture search over how connections are distributed throughout the neural network. Secondly, a scheme for building ensembles of sparse neural networks is shown to give better generalization and higher test time accuracy for a given choice of memory usage.

Since the appearance of [33], ℓ_0-based iterative thresholding for neural networks has been extended and improved upon by multiple other authors [68], [87]. Other related sparsity regularizations and constraints have also shown similar benefits [87], [203]. Han et al. showed that the use of sparsity in conjunction with other neural network compression techniques [67] yields yet greater reductions in deploy-time parameter storage costs; they show compression rates of $30 - 50\times$ with even smaller or no reduction in the resulting network's accuracy. Compressing and reducing the computational costs of neural networks has received special attention from the Deep Learning community in the past three years. As these models grow yet deeper and training datasets grow larger, attention to the practicalities of deploying and running them is a necessary part of making Deep Learning useful.

Chapter 6

Conclusion

The process of extending a Computer Vision model to include side information and additional constraints is key to making these models useful. While a central body of image understanding research focuses on the most general problems, such as image classification, object detection, and semantic segmentation, applying the resulting work to specialized tasks frequently requires taking the special structure of that task into account in the core model. Real-world deployments of Artificial Intelligence will inevitably make use of additional available signals including tags, context, and user interaction. In Vision, systems leveraging the full set of signals will advance significantly beyond what can be deduced from single images in isolation.

Chapter 3 presented a model for the cosegmentation problem. It included an efficient, GPU-parallelizable algorithm for the core optimization problem. Further, I proved theoretical optimality guarantees for a quasiconvex problem that used a histogram prior that was invariant to the scale of the foreground object. The original work in [35] used then-common features and Vision building blocks that were constructed by hand such as pixel color distances, textons, and SIFT. There are a number of clear possible extensions to this cosegmentation model and optimization scheme. Both employing more complex recent feature constructions and considering a more general class of related segmentation models could yield new advantages and superior accuracy.

One of more fundamental limitations to this model is that the histogram assignment matrix, defined in (3.5), is assumed to be a linear operator in much of the later discussion. Work such as [27] shows that it can be strongly advantageous, in identifying an object against its background, to use more complex and non-linear global models of an object's appearance. Significantly similar algorithms could also be used, though, to optimize a cosegmentation model that uses a more general class of object appearance descriptors. In this setting one would replace the product Hx with some nonlinear mapping $H(x)$ that maps the segmentation potentials to a descriptor vector. The normalized histogram

$$\frac{Hx}{\|Hx\|} \tag{6.1}$$

discussed in Section 3.4 indeed is also a nonlinear mapping that serves this function. More complex descriptors, derived for instance from Deep Learning [208], have the potential to produce much more accurate segmentation results with much less user interaction.

An arbitrary descriptor function would likely lead to a non-quasiconvex model, but a number of ways to incorporate complex segmentation priors have empirically been shown to produce good results [26], [157]. Since the Box-QP optimization described in Section 3.3 works only with derivatives of the objective, it could apply to any differentiable variant of the problem in (3.6), and it would take the form:

$$\begin{aligned} \min_{x_i, \bar{h}} \quad & \sum_i x_i^T L_i x_i + \lambda \|H(x_i) - \bar{h}\|_2^2 \\ \text{s.t.} \quad & x_i \in [0,1]^{n_i} \\ & x_i^{(s)} = m_i^{(s)} \qquad\qquad i = 1...m. \end{aligned} \tag{6.2}$$

This will not yield a convex optimization problem if the choice of descriptor function H is not itself convex. It is reasonable to expect this would converge on a reasonable segmentation, though, if given the right choice of parameters for a box-constrained optimization. For instance, a loss of optimality guarantees could be mitigated with an appropriate initialization, such as independent segmentations of each image.

Additionally, in the intervening time since the work done in [34], solutions to the related *instance segmentation* problem have advanced significantly. An instance segmentation solver can label the pixels in an image belonging to an object, or a set of objects, using only the knowledge about that object class learned from a labelled training set. The recent Mask-RCNN [71], for instance, shows an ability to segment objects belong to one of 80 classes [121] with an Average Precision (AP), averaged over thresholds on the intersection over union (IoU) ratio of the instance segmentations, of 37.1. This result does require a large training set with detailed segmentation labels, in their case more than 400 000 captions on over 80 000 images. This dataset is time-consuming and expensive to build, but after this it requires zero additional input at test time. The conjunction of this sort of segmentation with cosegmentation, though, could provide a balance between the two, as a smaller training set may be needed for new objects if at test time additional seeds are given as a hint.

Cosegmentation also suggests a model that would significantly reduce the effort required to build the training set for this kind of instance segmentation model. Using detailed ground-truth segmentations to find the learned descriptor h, given a hypothetical powerful enough feature construction to encompass the variation in a label class, the cosegmentation model will eventually become sufficient alone to select out a reasonable segmentation. Even given the histogram features used in this thesis, seen as in Figure 3.15, a powerful enough appearance model can be learned to segment instances of the same object in different poses. Taken to the fullest extent, this points towards cosegmentation, with the sort of general and fast optimization technique presented in Chapter 3, being a component of a few-shot [181] procedure for learning segmentation models.

Chapter 4 described a scalable stochastic optimization approach for multi-view Spectral Clustering with a convex regularizer. A useful feature of this approach is that at any given step, the gradient is computed only for a subset of the examples—the direct consequence being that, with an increase in the number of examples, the optimization can still make progress without having to compute the full gradient at each step. That chapter provided a detailed analysis of the clustering optimization's convergence properties, which sheds light on how adding a large number of processors in a distributed environment will affect its performance. Finally, the chapter discussed how high-level priors can be easily leveraged within this framework. The highly scalable implementation provided by this work is particularly useful in applications where one would want to effectively leverage such meta knowledge within inference, which remains difficult in alternatives based on Nyström extension. My empirical evaluations on several ML, vision, and synthetic datasets suggest that the model is scalable and efficient, and matches the performance of other existing multi-view Spectral Clustering models.

Machine Learning, as applied to the combination of images and text, has recently become one of the most popular topics in Computer Vision research. One of the more exciting fields is the prediction of relevant human-readable captions for an image [191], [202]. Starting with the initial publication of a large dataset suitable for training more complex models [121], the state of the art has advanced from an "M1 score," defined as the portion of generated captions that are evaluated as better than a human caption on the test set, of 0.166 [91] to 0.273 [191]. These captions, however, are much more focused and involved than the short tags and contextual text that frequently accompany images in the wild. Models that work with both text and images as input for other tasks have also shown promising results including using simple tags or highly unstructured text [171]. Tasks such as 3D semantic parsing [98], object detection [204], and zero-shot classification [48] can leverage text as an additional view to improve image and video understanding.

The work in [34] used one of the earliest useful feature embeddings from Deep Learning [45] to build one of the clustering views. Advances in this field in Computer Vision have, in the intervening time, produced models that are vastly superior to those available in 2014. In addition, CNNs have been applied to the full range of possible inferences one can make from an image, giving a vast array of specialized models that could each yield a feature to serve as a "view" within the multi-view clustering framework. Models have also been trained, using a *triplet loss*, to directly learn measures of distance and similarity on appropriately constructed training sets [106], [161]. These learned distance measures give a manifold embedding for a target task, meaning they are a type of distance that would be especially suited to spectral clustering.

Generalizing the procedure described in Chapter 4 also leads to a class of algorithms that optimize non-smooth objectives when the feasible set is a manifold. As long as the manifold allows for the kind of decomposition presented in this thesis, it is suitable for coordinate descent methods. The generalization [80], [81], as applied to eigenvalue problems, can be used to investigate population differences in brain connectivity data. Given the popularity of manifold optimization for tasks such as cosegmentation [88], matrix completion [185], and face recognition [79], allowing these techniques to scale to large problems is important. Techniques such as stochastic gradient descent [13] and coordinate descent, as adapted to these problems, provide one way to make their solvers fully scalable.

Building Deep Learning models that can be run in resource-constrained environments, as considered in Chapter 5, has been a major focus of Machine Learning research in the past few years. Work has appeared on ways to reduce the computation time [84], reduce the size of parameters [28], and model architectures built specifically for mobile computing [77]. This has been complemented by the construction of computing hardware designed specifically to execute neural networks efficiently [89].

Some of the most efficient networks have been built by combining many different techniques that all reduce the size and cost of inference. The work by Han et al. in [67] uses three methods for reducing the necessary storage for parameters. These include reducing the vast majority of the weights to zero using an iterative hard thresholding scheme, and performing inference with the resulting weights by representing them as a Compressed Sparse Row (CSR) matrix. They further reduce the storage needed for the nonzero weights by quantizing and compressing them. They first reduce the number of distinct weights that appear in the network, as done in [59], by performing k-means clustering, replacing each weight with the centroid of the resulting cluster to which it is assigned. Finally, they use Huffman coding to reduce the total bits necessary to represent these quantized weights.

Work specifically on sparse neural networks has also expanded upon the ways to make the networks more efficient. The work in [110] also enforces group sparsity, but where the groups are over weights in the same spatial position. This eliminates the need to even use sparse matrices to perform the convolution, with the resulting loss of memory locality, and instead reduces the elements necessary to construct the convolution as a matrix multiplication. The work in [115] uses grouped sparsity, with groups given by the convolutional filters. The connection between sparsifying the network and learning network architectures has also been explored [194], where a sparsification of a larger network serves as the solution for searching the space of smaller networks.

Reduced CNNs have been shown to be effective on the full range of Vision tasks, extending the initial results on efficiency beyond classification. Results have been shown for object detection [78], semantic segmentation [146], and on recurrent models [134]. Work on efficient neural nets has also expanded beyond convolutional models and Computer Vision tasks, including machine translation [162] and audio understanding [108].

All of these problem domains show rapid development of new ways to incorporate side information and modify the underlying Machine Learning models. This type of work leads to a unique class of design choices in the optimization formulations that drive most of the work in the literature. Each extension to the models poses challenges in how to make the optimization efficient, and in how to ensure a good solution is found. The work in this thesis shows a number of key tools that can be used to produce efficient, distributed, and robust Computer Vision and Machine Learning methods.

LIST OF REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467, 2016.

[2] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds.* Princeton, NJ: Princeton University Press, 2009.

[3] P.-A. Absil and J. Malick. Projection-like retractions on matrix manifolds. *SIAM Journal on Optimizations*, 22(1):135–158, 2012.

[4] L. Balzano, R. Nowak, and B. Recht. Online identification and tracking of subspaces from highly incomplete information. In *48th Annual Allerton Conference on Communication, Control, and Computing (CoCC)*, Monticello, IL, 2010. pp. 704–711.

[5] D. Batra, A. Kowdle, D. Parikh, J. Luo, and T. Chen. Interactive co-segmentation with intelligent scribble guidance. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, CA, 2010. pp. 3169–3176.

[6] H. Bay, T. Tuytelaars, and L. V. Gool. SURF: Speeded up robust features. In *Proc. of the European Conference on Computer Vision (ECCV)*, Graz, Austria, 2006. Springer, pp. 404–417.

[7] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms.* Hoboken, NJ: John Wiley & Sons, Inc., 2003.

[8] N. Bell, S. Dalton, and L. N. Olson. Exposing fine-grained parallelism in algebraic multigrid methods. *SIAM Journal on Scientific Computing*, 34(4):C123–C152, 2012.

[9] N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proc. of the Conference on High Performance Computing Networking, Storage and Analysis*, Portland, OR, 2009. ACM, p. 18.

[10] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. arXiv:1306.6709, 2013.

[11] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

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

[13] S. Bonnabel. Stochastic gradient descent on Riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229, 2013.

[14] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT*, Paris, France, 2010. pp. 177–186.

[15] S. Boughorbel, J.-P. Tarel, and F. Fleuret. Non-Mercer kernels for SVM object recognition. In *Proc. of the British Machine Vision Conference (BMVC)*, London, U.K., 2004. pp. 1–10.

[16] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15:1455–1459, 2014.

[17] O. Bousquet and L. Bottou. The tradeoffs of large scale learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Vancouver, Canada, 2008. Curran Associates, Inc., pp. 161–168.

[18] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge, U.K.: Cambridge University Press, 2004.

[19] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, 2001, pp. 105–112.

[20] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 23(11):1222–1239, 2001.

[21] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[22] P. Brémaud. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*, volume 31. New York, NY: Springer Science & Business Media, 2013.

[23] C. Bucilă, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, PA, 2006. pp. 535–541.

[24] S. Burer and J. Chen. Relaxing the optimality conditions of box QP. *Computational Optimization and Applications*, 48(3):653–673, 2011.

[25] X. Cai, F. Nie, W. Cai, and H. Huang. Heterogeneous image features integration via multi-modal semi-supervised learning model. In *Proc. of the International Conference on Computer Vision (ICCV)*, Sydney, Australia, Dec 2013. pp. 1737–1744.

[26] Y. Chai, E. Rahtu, V. Lempitsky, L. Van Gool, and A. Zisserman. Tricos: A tri-level class-discriminative co-segmentation method for image classification. In *Proc. of the European Conference on Computer Vision (ECCV)*, Florence, Italy, 2012. Springer, pp. 794–807.

[27] F. Chen, H. Yu, R. Hu, and X. Zeng. Deep learning shape priors for object segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, OH, 2013. pp. 1870–1877.

[28] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *Proc. of the International Conference on Machine Learning (ICML)*, Lille, France, 2015. pp. 2285–2294.

[29] W.-Y. Chen, Y. Song, H. Bai, C.-J. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33(3):568–586, 2011.

[30] X. Chen and D. Cai. Large scale spectral clustering with landmark-based representation. In *Proc. of the AAAI Conference on Artificial Intelligence*, San Francisco, CA, 2011.

[31] F. Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv:1610.02357, 2016.

[32] W.-S. Chu, C.-P. Chen, and C.-S. Chen. MOMI-cosegmentation: Simultaneous segmentation of multiple objects among multiple images. In *Proc. of the Asian Conference on Computer Vision (ACCV)*, Queenstown, New Zealand, 2010. pp. 355–368.

[33] M. D. Collins and P. Kohli. Memory bounded deep convolutional networks. arXiv:1412.1442, 2014.

[34] M. D. Collins, J. Liu, J. Xu, L. Mukherjee, and V. Singh. Spectral clustering with a convex regularizer on millions of images. In *Proc. of the European Conference on Computer Vision (ECCV)*, Zürich, Switzerland, 2014. Springer, pp. 282–298.

[35] M. D. Collins, J. Xu, L. Grady, and V. Singh. Random walks based multi-image segmentation: Quasiconvexity results and GPU-based solutions. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Providence, Rhode Island, 2012. pp. 1656–1663.

[36] C. Couprie, L. Grady, L. Najman, and H. Talbot. Power watersheds: A unifying graph-based optimization framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33(7):1384–1399, 2011.

[37] J. Cui, Q. Yang, F. Wen, Q. Wu, C. Zhang, L. V. Gool, and X. Tang. Transductive object cutout. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, AK, 2008. pp. 1–8.

[38] G. B. Dantzig. Origins of the simplex method. In S. G. Nash, Ed., *A History of Scientific Computing*. New York, NY: ACM, 1990, pp. 141–151.

[39] C. Darken and J. Moody. Towards faster stochastic gradient search. In J. Cowan, G. Tesauro, and J. Alspector, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Denver, CO, 1993. Curran Associates, Inc., pp. 1009–1009.

[40] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS).* Stateline, NV: Curran Associates, Inc., 2012, pp. 1223–1231.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, FL, 2009. pp. 248–255.

[42] J. Deng, S. Satheesh, A. C. Berg, and L. Fei-Fei. Fast and balanced: Efficient label tree learning for large scale object recognition. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Grenada, Spain, 2011. Curran Associates, Inc., pp. 567–575.

[43] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS).* Montréal, Canada: Curran Associates, Inc., 2014, pp. 1269–1277.

[44] M. P. a. Do Carmo and J. F. Francis. *Riemannian Geometry.* Boston, MA: Birkhäuser, 1992.

[45] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. arXiv:1310.1531, 2013.

[46] D. L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[47] A. Edelman, T. A. Arias, and S. T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2):303–353, 1998.

[48] M. Elhoseiny, Y. Zhu, H. Zhang, and A. Elgammal. Link the head to the "beak": Zero shot learning from noisy text description at part precision. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017. pp. 5640–5649.

[49] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[50] D. Eynard, A. Kovnatsky, M. M. Bronstein, K. Glashoff, and A. M. Bronstein. Multimodal manifold analysis by simultaneous diagonalization of Laplacians. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 37(12):2505–2517, 2015.

[51] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient belief propagation for early vision. *International Journal of Computer Vision (IJCV)*, 70(1):41–54, 2006.

[52] P. F. Felzenszwalb and R. Zabih. Dynamic programming and graph algorithms in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33(4):721–740, 2011.

[53] M. C. Ferris, O. L. Mangasarian, and S. J. Wright. *Linear programming with MATLAB*. Philadelphia, PA: SIAM, 2007.

[54] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.

[55] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the Nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(2):214–225, Feburary 2004.

[56] R. Frank, J. Schneid, and C. W. Ueberhuber. Stability properties of implicit Runge-Kutta methods. *SIAM Journal on Numerical Analysis*, 22(3):497–514, 1985.

[57] C. Galleguillos, A. Rabinovich, and S. Belongie. Object categorization using co-occurrence, location and appearance. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, AK, 2008. pp. 1–8.

[58] R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle points — online stochastic gradient for tensor decomposition. In *Proc. of The 28th Conference on Learning Theory*, volume 40 of *Proceedings of Machine Learning Research*, Paris, France, 03–06 Jul 2015. pp. 797–842.

[59] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. arXiv:1412.6115, 2014.

[60] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[61] L. Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(11):1768–1783, 2006.

[62] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx, Mar. 2014.

[63] M. Grundmann, V. Kwatra, M. Han, and I. Essa. Efficient hierarchical graph-based video segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 2141–2148.

[64] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. arXiv:1502.02551, 2015.

[65] C. Hage and M. Kleinsteuber. Robust PCA and subspace tracking from incomplete observations using ℓ_0-surrogates. *Computational Statistics*, 29(3-4):467–487, 2014.

[66] L. Hagen and A. B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 11(9):1074–1085, 1992.

[67] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv:1510.00149, 2015.

[68] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., *Advances in Neural Information Processing Systems (NIPS)*. Montréal, Canada: Curran Associates, Inc., 2015, pp. 1135–1143.

[69] M. T. Harandi, M. Salzmann, and R. Hartley. From manifold to manifold: Geometry-aware dimensionality reduction for SPD matrices. In *Proc. of the European Conference on Computer Vision (ECCV)*, Zürich, Switzerland, 2014. Springer, pp. 17–32.

[70] M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *Proc. of the International Conference on Machine Learning (ICML)*, New York, NY, 2016. pp. 1225–1234.

[71] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. arXiv:1703.06870, 2017.

[72] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. arXiv:1502.01852, 2015.

[73] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *Proc. of the European Conference on Computer Vision (ECCV)*, Amsterdam, The Netherlands, 2016. Springer, pp. 630–645.

[74] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580, 2012.

[75] D. S. Hochbaum and V. Singh. An efficient algorithm for co-segmentation. In *Proc. of the International Conference on Computer Vision (ICCV)*, Kyoto, Japan, 2009. pp. 269–276.

[76] T. Hofmann. Probabilistic latent semantic analysis. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, Stockholm, Sweden, 1999. pp. 289–296.

[77] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861, 2017.

[78] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017.

[79] Z. Huang, R. Wang, S. Shan, and X. Chen. Projection metric learning on Grassmann manifold with application to video based face recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015. pp. 140–149.

[80] S. J. Hwang, N. Adluru, M. D. Collins, S. N. Ravi, B. B. Bendlin, S. C. Johnson, and V. Singh. Coupled harmonic bases for longitudinal characterization of brain networks. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016. pp. 2517–2525.

[81] S. J. Hwang, M. D. Collins, S. N. Ravi, V. K. Ithapu, N. Adluru, S. C. Johnson, and V. Singh. A projection free method for generalized eigenvalue problem with a nonsmooth regularizer. In *Proc. of the International Conference on Computer Vision (ICCV)*, Santiago, Chile, 2015. pp. 1841–1849.

[82] E.-J. Im, K. Yelick, and R. Vuduc. Sparsity: Optimization framework for sparse matrix kernels. *International Journal of High Performance Computing Applications*, 18(1):135–158, 2004.

[83] H. Ishikawa. Exact optimization for Markov random fields with convex priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 25(10):1333–1336, 2003.

[84] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proc. of the British Machine Vision Conference (BMVC)*, Nottingham, U.K., 2014. pp. 1–13.

[85] P. Jain, A. Tewari, and P. Kar. On iterative hard thresholding methods for high-dimensional M-estimation. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS)*. Montréal, Canada: Curran Associates, Inc., 2014, pp. 685–693.

[86] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, Orlando, FL, 2014. pp. 675–678.

[87] X. Jin, X. Yuan, J. Feng, and S. Yan. Training skinny deep neural networks with iterative hard thresholding methods. arXiv:1607.05423, 2016.

[88] A. Joulin, F. Bach, and J. Ponce. Discriminative clustering for image co-segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 1943–1950.

[89] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. of the 44th Annual International Symposium on Computer Architecture*, Toronto, Canada, 2017. ACM, pp. 1–12.

[90] M. Kan, S. Shan, H. Zhang, S. Lao, and X. Chen. Multi-view discriminant analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 38(1):188–194, 2016.

[91] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015. pp. 3128–3137.

[92] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision (IJCV)*, 1(4):321–331, 1988.

[93] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv:1609.04836, 2016.

[94] G. Kim, E. P. Xing, L. Fei-Fei, and T. Kanade. Distributed cosegmentation via submodular optimization on anisotropic diffusion. In *Proc. of the International Conference on Computer Vision (ICCV)*, Barcelona, Spain, 2011. pp. 169–176.

[95] D. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv:1412.6980, 2014.

[96] N. Koep and S. Weichwald. Pymanopt: A python toolbox for optimization on manifolds using automatic differentiation. *Journal of Machine Learning Research*, 17:1–5, 2016.

[97] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(2):147–159, 2004.

[98] C. Kong, D. Lin, M. Bansal, R. Urtasun, and S. Fidler. What are you talking about? Text-to-image coreference. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, OH, 2014. pp. 3558–3565.

[99] A. Kowdle, D. Batra, W.-C. Chen, and T. Chen. iModel: Interactive co-segmentation for object of interest 3D modeling. In *European Conference on Computer Vision (ECCV) Workshops*, Hersonissos, Greece, 2010. Springer, pp. 211–224.

[100] A. Krishnamurthy, S. Balakrishnan, M. Xu, and A. Singh. Efficient active algorithms for hierarchical clustering. In *Proc. of the International Conference on Machine Learning (ICML)*, Edinburgh, Scotland, 2012. pp. 267–274.

[101] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Computer Science Department, University of Toronto, 2009.

[102] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Stateline, NV, 2012. Curran Associates, Inc., pp. 1097–1105.

[103] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In S. Hanson, J. Cowan, and C. Giles, Eds., *Advances in Neural Information Processing Systems (NIPS)*. Denver, CO: Curran Associates, Inc., 1992, pp. 950–957.

[104] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proc. of the International Conference on Computer Vision (ICCV)*, Kyoto, Japan, 2009. pp. 2130–2137.

[105] A. Kumar, P. Rai, and H. Daume. Co-regularized multi-view spectral clustering. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Granada, Spain, 2011. Curran Associates, Inc., pp. 1413–1421.

[106] V. B. G. Kumar, G. Carneiro, and I. Reid. Learning local image descriptors with deep siamese and triplet convolutional networks by minimising global loss functions. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016. pp. 5385–5394.

[107] L. Ladicky, C. Russell, P. Kohli, and P. H. S. Torr. Graph cut based inference with co-occurrence statistics. In *Proc. of the European Conference on Computer Vision (ECCV)*, Hersonissos, Greece, 2010. Springer, pp. 239–253.

[108] N. D. Lane, P. Georgiev, and L. Qendro. DeepEar: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proc. of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, Osaka, Japan, 2015. ACM, pp. 283–294.

[109] J. Langford, A. J. Smola, and M. Zinkevich. Slow learners are fast. volume 22, Vancouver, Canada, 2009. Curran Associates, Inc., pp. 2331–2339.

[110] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016. pp. 2554–2564.

[111] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[112] S. Lee and S. J. Wright. Implementing algorithms for signal and image reconstruction on graphical processing units. Technical report, University of Wisconsin Madison, 2008.

[113] Y. J. Lee and K. Grauman. CollectCut: Segmentation with top-down cues discovered in multi-object images. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, CA, 2010. pp. 3185–3192.

[114] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, volume 6. 1998.

[115] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *Proc. of the International Conference on Learning Representations (ICLR)*, Toulon, France, 2017.

[116] M. Li, X.-C. Lian, J. T. Kwok, and B.-L. Lu. Time and space efficient spectral clustering via column sampling. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Colorado Springs, CO, 2011. pp. 2297–2304.

[117] S. Z. Li. Markov random field models in computer vision. In *Proc. of the European Conference on Computer Vision (ECCV)*, Stockholm, Sweden, 1994. Springer, pp. 361–370.

[118] Y. Li and S. Osher. Coordinate descent optimization for ℓ^1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problems and Imaging*, 3(3):487–503, 2009.

[119] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei. Fully convolutional instance-aware semantic segmentation. arXiv:1611.07709, 2016.

[120] M. Lin, Q. Chen, and S. Yan. Network in network. arXiv:1312.4400, 2013.

[121] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *Proc. of the European Conference on Computer Vision (ECCV)*, Zürich, Switzerland, 2014. Springer, pp. 740–755.

[122] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *Journal of Machine Learning Research*, 16(1):285–322, 2015.

[123] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004.

[124] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 30, Atlanta, GA, 2013.

[125] T. Meltzer, C. Yanover, and Y. Weiss. Globally optimal solutions for energy minimization in stereo vision using reweighted belief propagation. In *Proc. of the International Conference on Computer Vision (ICCV)*, volume 1, Beijing, China, 2005. pp. 428–435.

[126] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 27(10):1615–1630, 2005.

[127] J. J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, 1(1):93–113, 1991.

[128] E. N. Mortensen and W. A. Barrett. Interactive segmentation with intelligent scissors. *Graphical Models and Image Processing*, 60(5):349–384, 1998.

[129] L. Mukherjee, V. Singh, and C. R. Dyer. Half-integrality based algorithms for cosegmentation of images. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Miami, FL, 2009. pp. 2028–2035.

[130] L. Mukherjee, V. Singh, and J. Peng. Scale invariant cosegmentation for image groups. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Colorado Springs, CO, June 2011. pp. 1881–1888.

[131] L. Mukherjee, V. Singh, J. Xu, and M. Collins. Analyzing the subspace structure of related images: Concurrent segmentation of image sets. In *Proc. of the European Conference on Computer Vision (ECCV)*, volume 7575 of *Lecture Notes in Computer Science*. Florence, Italy: Springer Berlin Heidelberg, 2012, pp. 128–142.

[132] N. Murata. A statistical study of on-line learning. In *Online Learning and Neural Networks*. Cambridge, U.K.: Cambridge University Press, 1998.

[133] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, Stockholm, Sweden, 1999. Morgan Kaufmann Publishers Inc., pp. 467–475.

[134] S. Narang, G. Diamos, S. Sengupta, and E. Elsen. Exploring sparsity in recurrent neural networks. arXiv:1704.05119, 2017.

[135] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, 1983, pp. 372–376.

[136] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[137] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In S. Becker, S. Thrun, and K. Obermayer, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Vancouver, Canada, 2002. Curran Associates, Inc., pp. 849–856.

[138] A. Nitanda. Stochastic proximal gradient descent with acceleration techniques. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Montréal, Canada, 2014. Curran Associates, Inc., pp. 1574–1582.

[139] F. Niu, B. Recht, C. Ré, and S. J. Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Granada, Spain, 2011. Curran Associates, Inc., pp. 693–701.

[140] J. Nocedal and S. J. Wright. *Numerical Optimization*. New York, NY: Springer, 2nd edition, 2006.

[141] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 24(7):971–987, 2002.

[142] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision (IJCV)*, 42(3):145–175, 2001.

[143] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311 – 3325, 1997.

[144] D. Pachauri, M. Collins, R. Kondor, and V. Singh. Incorporating domain knowledge in matching problems via harmonic analysis. In *Proc. of the International Conference on Machine Learning (ICML)*, Edinburgh, Scotland, 2012. p. 1271.

[145] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014.

[146] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. arXiv:1606.02147, 2016.

[147] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann, 1988.

[148] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov. Real-time computer vision with OpenCV. *Communications of the ACM*, 55(6):61–69, 2012.

[149] A. Rabinovich, A. Vedaldi, C. Galleguillos, E. Wiewiora, and S. Belongie. Objects in context. In *Proc. of the International Conference on Computer Vision (ICCV)*, Rio de Janeiro, Brazil, 2007. pp. 1–8.

[150] B. Recht and C. Ré. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*, 5(2):201–226, 2013.

[151] R. Reed. Pruning algorithms–a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[152] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014.

[153] W. Ring and B. Wirth. Optimization methods on Riemannian manifolds and their application to shape space. *SIAM Journal on Optimization*, 22(2):596–627, 2012.

[154] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[155] S. Roth, V. Lempitsky, and C. Rother. Discrete-continuous optimization for optical flow estimation. In D. Cremers, B. Rosenhahn, A. L. Yuille, and F. R. Schmidt, Eds., *Statistical and Geometrical Approaches to Visual Motion Analysis.* Berlin, Germany: Springer, 2009, pp. 1–22.

[156] C. Rother, V. Kolmogorov, T. Minka, and A. Blake. Cosegmentation of image pairs by histogram matching - Incorporating a global constraint in MRFs. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, New York, NY, 2006. pp. 993–1000.

[157] J. C. Rubio, J. Serrat, A. López, and N. Paragios. Unsupervised co-segmentation through region matching. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Providence, RI, 2012. pp. 749–756.

[158] Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations. Technical Report 90.20, RIACS, NASA Ames Research Center, 1990.

[159] A. Saha and A. Tewari. On the finite time convergence of cyclic coordinate descent methods. arXiv:1005.2146, 2010.

[160] M. Schmidt, N. L. Roux, and F. R. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS)*. Grenada, Spain: Curran Associates, Inc., 2011, pp. 1458–1466.

[161] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015. pp. 815–823.

[162] A. See, M.-T. Luong, and C. D. Manning. Compression of neural machine translation models via pruning. arXiv:1606.09274, 2016.

[163] S. Seo, M. K. Chung, and H. K. Vorperian. Heat kernel smoothing using Laplace-Beltrami eigenfunctions. In *Proc. of Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, Beijing, China, 2010. Springer, pp. 505–512.

[164] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv:1312.6229, 2014.

[165] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, San Diego, CA, 2005. pp. 994–1000.

[166] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical programming*, 127(1):3–30, 2011.

[167] S. Shalev-Shwartz and A. Tewari. Stochastic methods for l1-regularized loss minimization. *Journal of Machine Learning Research*, 12:1865–1892, 2011.

[168] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(8):888–905, Aug 2000.

[169] J. Shotton, J. Winn, C. Rother, and A. Criminisi. TextonBoost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Proc. of the European Conference on Computer Vision (ECCV)*, Graz, Austria, 2006. Springer, pp. 1–15.

[170] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Stateline, NV, 2012. Curran Associates, Inc., pp. 2951–2959.

[171] N. Srivastava and R. R. Salakhutdinov. Multimodal learning with deep boltzmann machines. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., *Advances in Neural Information Processing Systems (NIPS)*. Curran Associates, Inc., 2012, pp. 2222–2230.

[172] A. Strehl and J. Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

[173] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4):625–653, 1999.

[174] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 28, Atlanta, GA, 2013. pp. 1139–1147.

[175] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. arXiv:1409.4842, 2014.

[176] C. J. Taylor and D. J. Kriegman. Minimization on the lie group $SO(3)$ and related manifolds. Technical Report 9405, Yale University, 1994.

[177] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):pp. 267–288, 1996.

[178] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(11):1958–1970, 2008.

[179] A. Toshev, B. Taskar, and K. Daniilidis. Object detection via boundary structure segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, CA, 2010. pp. 950–957.

[180] T. Toyoda and O. Hasegawa. Random field model for integration of local information and global information. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(8):1483–1489, 2008.

[181] E. Triantafillou, R. Zemel, and R. Urtasun. Few-shot learning through an information retrieval lens. arXiv:1707.02610, 2017.

[182] D. Tsai, M. Flagg, and J. M. Rehg. Motion coherent tracking with multi-label MRF optimization. In *Proc. of the British Machine Vision Conference (BMVC)*, Aberystwyth, Wales, 2010. pp. 190–202.

[183] O. Tuzel, F. Porikli, and P. Meer. Region covariance: A fast descriptor for detection and classification. In *Proc. of the European Conference on Computer Vision (ECCV)*, Graz, Austria, 2006. Springer, pp. 589–600.

[184] M. O. Ulfarsson and V. Solo. Sparse variable PCA using geodesic steepest descent. *IEEE Transactions on Signal Processing*, 56(12):5823–5832, 2008.

[185] B. Vandereycken. Low-rank matrix completion by Riemannian optimization. *SIAM Journal on Optimization*, 23(2):1214–1236, 2013.

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

[187] S. Vicente, V. Kolmogorov, and C. Rother. Graph cut based image segmentation with connectivity priors. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, AK, 2008. pp. 1–8.

[188] S. Vicente, V. Kolmogorov, and C. Rother. Cosegmentation revisited: Models & optimization. In *Proc. of the European Conference on Computer Vision (ECCV)*, Hersonissos, Greece, 2010. Springer, pp. 465–479.

[189] S. Vicente, C. Rother, and V. Kolmogorov. Object cosegmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Colorado Springs, CO, 2011. pp. 2217–2224.

[190] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proc. of the International Conference on Machine Learning (ICML)*, Helsinki, Finland, 2008. pp. 1096–1103.

[191] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015. pp. 3156–3164.

[192] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[193] Y. Weiss and W. T. Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744, 2001.

[194] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Barcelona, Spain, 2016. Curran Associates, Inc., pp. 2074–2082.

[195] Z. Wen and W. Yin. A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 142(1-2):1–38, 2012.

[196] W. Wiegerinck, A. Komoda, and T. Heskes. Stochastic dynamics of learning with momentum in neural networks. *Journal of Physics A: Mathematical and General*, 27(13):4425, 1994.

[197] J. Winn, A. Criminisi, and T. Minka. Object categorization by learned universal visual dictionary. In *Proc. of the International Conference on Computer Vision (ICCV)*, volume 2, Beijing, China, 2005. pp. 1800–1807.

[198] P. Wolfe. The simplex method for quadratic programming. *Econometrica: Journal of the Econometric Society*, 27(3):382–398, 1959.

[199] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[200] S. J. Wright, R. D. Nowak, and M. A. T. Figueiredo. Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing*, 57(7):2479–2493, 2009.

[201] L. Xiao and T. Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

[202] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. of the International Conference on Machine Learning (ICML)*, Lille, France, 2015. pp. 2048–2057.

[203] J. Yoon and S. J. Hwang. Combined group and exclusive sparsity for deep neural networks. In *Proc. of the International Conference on Machine Learning (ICML)*, Sydney, Australia, 2017. pp. 3958–3966.

[204] H. Yu and J. M. Siskind. Sentence directed video object codetection. arXiv:1506.02059, 2015.

[205] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 68(1):pp. 49–67, 2006.

[206] M. D. Zeiler. Adadelta: an adaptive learning rate method. arXiv:1212.5701, 2012.

[207] Y. Zeng, D. Samaras, W. Chen, and Q. Peng. Topology cuts: A novel min-cut/max-flow algorithm for topology preserving segmentation in N–D images. *Computer Vision and Image Understanding*, 112(1):81–90, 2008.

[208] D. Zhang, J. Han, C. Li, and J. Wang. Co-saliency detection via looking deep and wide. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015. pp. 2994–3002.

[209] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. Efficient and accurate approximations of nonlinear convolutional networks. arXiv:1411.4229, 2014.

[210] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., *Advances in Neural Information Processing Systems (NIPS)*, Vancouver, Canada, 2010. Curran Associates, Inc., pp. 2595–2603.