TASK-SPECIFIC AND INTERPRETABLE FEATURE LEARNING

BY

ZHANGYANG WANG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Doctoral Committee:

Professor Emeritus Thomas S. Huang, Chair
Professor Zhi-Pei Liang
Professor Mark Hasegawa-Johnson
Associate Professor Florin Dolcos
Dr. Jianchao Yang, Snapchat Inc.

ABSTRACT

Deep learning models have had tremendous impacts in recent years, while a question has been raised by many: Is deep learning just a triumph of empiricism? There has been emerging interest in reducing the gap between the theoretical soundness and interpretability, and the empirical success of deep models. This dissertation provides a comprehensive discussion on bridging traditional model-based learning approaches that emphasize problem-specific reasoning, and deep models that allow for larger learning capacity. The overall goal is to devise the next-generation feature learning architectures that are: 1) task-specific, namely, optimizing the entire pipeline from end to end while taking advantage of available prior knowledge and domain expertise; and 2) interpretable, namely, being able to learn a representation consisting of semantically sensible variables, and to display predictable behaviors.

This dissertation starts by showing how the classical sparse coding models could be improved in a task-specific way, by formulating the entire pipeline as bi-level optimization. Then, it mainly illustrates how to incorporate the structure of classical learning models, e.g., sparse coding, into the design of deep architectures. A few concrete model examples are presented, ranging from the ℓ_0 and ℓ_1 sparse approximation models, to the ℓ_∞ constrained model and the dual-sparsity model. The analytic tools in the optimization problems can be translated to guide the architecture design and performance analysis of deep models. As a result, those customized deep models demonstrate improved performance, intuitive interpretation, and efficient parameter initialization. On the other hand, deep networks are shown to be analogous to brain mechanisms. They exhibit the ability to describe semantic content from the primitive level to the abstract level. This dissertation thus also presents a preliminary investigation of the synergy between feature learning with cognitive science and neuroscience. Two novel application domains, image aesthetics assessment and brain encoding, are explored, with promising preliminary results achieved.

To my parents, and my wife Li

ACKNOWLEDGMENTS

Foremost, I would like to express sincere gratitude to my adviser, Professor Thomas Huang, for sharing with me his invaluable expertise and insight during my entire Ph.D. study, without which neither this dissertation nor my career would have been possible. I am also truly grateful to Professor Mark Hasegawa-Johnson, Professor Zhi-Pei Liang, and Professor Florin Dolcos, for serving on my doctoral committee and offering valued suggestions in the preparation of this dissertation.

I would like to extend my thanks to my previous internship mentors: Dr. Jin Li, Dr. Lei Zhang and Dr. Yuxiao Hu (Microsoft Research); Dr. Hailin Jin and Dr. Jianchao Yang (Adobe Research); and Dr. Nasser Nasrabadi (US Army Research Lab). The experiences with them were both productive and enjoyable, and made up indispensable parts of this dissertation.

Over the past years, it has been my great honor to collaborate with many outstanding researchers, including Professor Chang Wen Chen (University of Buffalo), Professor Qing Ling (University of Science and Technology of China), Professor Shuai Huang (University of Washington), Professor Jiayu Zhou (Michigan State University), Professor Xia Ben Hu (Texas A&M University), and many others. Every one of them has helped sharpen my skills and broaden my perspective.

There are also other fellow students having directly or indirectly contributed to this dissertation, especially Dr. Zhaowen Wang, Dr. Shiyu Chang, Ding Liu, Wei Han and Xian-Ming Liu, to all of whom I owe my sincere gratitude. My thanks also go to many other friends, in particular to all the former and current members of the Image Formation and Processing (IFP) group, whose continuing support and encouragement have made my Ph.D. career an unforgettable journey.

Lastly and most importantly, I want to thank my parents and my wife for their boundless love, understanding, and support in all the ways in my life.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Seeking meaningful feature representations is one of the fundamental problems in signal processing, machine learning and pattern recognition. Feature learning is a set of techniques that automatically learn a transformation of raw data input to an effective representation for subsequent tasks. Many traditional model-based feature learning approaches, such as Principal Component Analysis (PCA) [16], support vector machine (SVM) [40] and sparse coding [96], rely on simple priors and concise architectures, and have proven effective in solving many well-constrained and well-formulated problems. Despite their extensive theoretical and algorithm results, the limited representation power of traditional methods appears insufficient for modeling the emerging *Big Data*. In addition, most of their inference algorithms have to rely on inefficient iterative solutions.

Lately, deep learning [89][1] has attracted great attention for its tremendous representation power. A deep feed-forward network adopts multiple layers of nonlinear feature transformations, and could be naturally tuned with a task-driven loss. However, generic deep architectures, sometimes referred to as *"black-box"* methods, largely ignore the problem-specific formulations and prior knowledge. Instead, they rely on stacking somewhat ad-hoc modules and an explosive amount of parameters.[2] Despite a few hypotheses and intuitions, it remains difficult to understand and interpret the working mechanism of deep models, as well as to relate them with prior wisdom of traditional models.

The thesis recognizes the existing gap between deep models with large representation capacity, and traditional models that emphasize problem-specific prior and interpretability. To reduce the gap, rather than applying off-the-shelf models as "black boxes", we aim to devise novel feature learning architectures that are:

[1] Throughout the thesis, "deep network" and "deep model" are interchangeably used, both of which refer to deep feed-forward neural networks by default.

[2] An interesting reference as reported: a Google search for "neural network black box" yields 2,410,000 results. By comparison, "logistic regression black box" yields 600,000 results. Most of the latter articles seem to position logistic regression as opposite to neural networks.

- **Task-specific**, namely, optimizing the entire pipeline from end to end while taking advantage of available prior knowledge and domain expertise.
- **Interpretable**, namely, being able to learn a representation consisting of semantically sensible variables, and to display predictable behaviors.

This dissertation starts with the discussion on sparse coding, a representative example of traditional model-based feature learning approaches, in Chapter 2. Sparse coding is believed, by many researchers, to have an outstanding interpretability. The sensory processing in the brain suggests a sparse coding strategy over a highly over-complete basis, when finding stimuli that effectively activate the neurons [126]. Given a proper basis, sparse linear models are also well known to be a feature selection tool (a.k.a., LASSO). In particular, this dissertation illustrates how the interpretable sparse coding model could be further improved in a task-specific way, by formulating the end-to-end pipeline as bi-level optimization.

In Chapter 3, we demonstrate how to incorporate the structure of classical problems, e.g., sparse coding, into the design of deep architectures. Despite the remarkable effectiveness and interpretability of sparse coding, the bottlenecks in both efficiency and scalability limit its practical usage. We exploit deep networks as fast trainable regressors to approximate the sparse coding inference, and perform the end-to-end training with the specific loss function. Such deep architectures benefit from their problem-specific regularizations and interpretable pipelines, and achieve improved performance. The resulting framework is also capable of interpreting the empirical success of many deep learning techniques.

On the other hand, deep networks are shown to be analogous to brain mechanisms. They also exhibit the ability to describe semantic content from the primitive level to the abstract level. Chapter 4 presents a preliminary investigation on the synergy between feature learning with cognitive science and neuroscience. Deep models can be customized by the knowledge of the cognitive and neural underpinnings of human perception. Those models describe and estimate human preference better, and may in turn help understand the complex neural mechanisms underlying human perception. Finally, Chapter 6 concludes this dissertation with a summary and future directions.

CHAPTER 2

TASK-SPECIFIC SPARSE CODING

In this chapter, we refer to two concrete examples: 1) hyperspectral image classification; 2) sparse coding-based image clustering as a concrete example, to demonstrate how to develop task-specific sparse coding models, by formulating the end-to-end pipeline as bi-level optimization.

2.1 Bi-Level Sparse Coding for Hyperspectral Image Classification

We present a semi-supervised method for single pixel classification of hyperspectral images. The proposed method is designed to address the special problematic characteristics of hyperspectral images, namely, high dimensionality of hyperspectral pixels, lack of labeled samples, and spatial variability of spectral signatures. To alleviate these problems, the proposed method features the following components. First, being a semi-supervised approach, it exploits the wealth of unlabeled samples in the image by evaluating the confidence probability of the predicted labels, for each unlabeled sample. Second, we propose to jointly optimize the classifier parameters and the dictionary atoms by a *task-specific* formulation, in order to ensure that the learned features (sparse codes) are optimal for the trained classifier. Finally, it incorporates spatial information through adding a Laplacian smoothness regularization to the output of the classifier, rather than the sparse codes, making the spatial constraint more flexible. The proposed method is compared to a few comparable methods for classification of several popular datasets, and it produces significantly better classification results.

2.1.1 Introduction

The spectral information contained in hyperspectral imagery allows characterization, identification, and classification of land-covers with improved accuracy and robustness. However, several critical issues should be addressed in the classification of hyperspectral data [48, 129]: 1) small amount of available labeled data; 2) high dimensionality of each spectral sample; 3) spatial variability of spectral signatures; 4) high cost of sample labeling. In particular, the large number of spectral channels and small number of labeled training samples pose the problem of the curse of dimensionality and as a consequence result in the risk of overfitting the training data. For these reasons, desirable properties of a hyperspectral image classifier should be its ability to produce accurate land-cover maps when working within a high-dimensional feature space, low-sized training datasets, and high levels of spatial spectral signature variability.

Many supervised and unsupervised classifiers have been developed to tackle the hyperspectral data classification problem [132]. Classical supervised methods, such as artificial neural networks [15, 175] and support vector machines (SVMs) [42, 138, 40, 8], were readily revealed to be inefficient when dealing with a high number of spectral bands and lack of labeled data. In [66], SVM was regularized with an unnormalized graph Laplacian, thus leading to the Laplacian SVM (LapSVM) that adopts the manifold assumption for semi-supervised classification. Another framework based on neural networks was presented in [131]. It consists of adding a flexible embedding regularizer to the loss function used for training neural networks, and leads to improvements in both classification accuracy and scalability on several hyperspectral image classification problems. In recent years, kernel-based methods have often been adopted for hyperspectral image classification [25, 77, 24, 23]. They are certainly able to handle efficiently the high-dimensional input feature space and deal with the noisy samples in a robust way [140]. More recently, sparse representation has been increasingly popular for image classification. The sparse representation-based classification (SRC) [170] is mainly based on the observation that despite the high dimensionality of natural signals, signals belonging to the same class usually lie in a low-dimensional subspace. In [33], an SRC-based algorithm for hyperspectral classification was presented that utilizes the sparsity of the input sample with respect to a given overcomplete training dictionary. It is based on a sparsity model where a test spectral pixel is approximately represented by a few training samples (atoms) among the

entire atoms from a dictionary. The weightings associated with the atoms are called the sparse code. The class label of the test pixel is then determined by the characteristics of the recovered sparse code. Experimental results show remarkable improvements in discriminative effects. However, the main difficulty with all supervised methods is that the learning process heavily depends on the quality of the training dataset. Even worse, labeled hyperspectral training samples are only available in a very limited number due to the cost of sample labeling. On the other hand, unsupervised methods are not sensitive to the number of labeled samples since they operate on the whole dataset, but the relationships between clusters and class labels are not ensured [13]. Moreover, typically in hyperspectral classification, a preliminary feature selection/extraction step is undertaken to reduce the high input space dimensionality, which is time-consuming, scenario-dependent, and needs prior knowledge.

As a trade-off, semi-supervised classification methods become a natural alternative to yield better performance. In semi-supervised learning literature, the algorithms are provided with some available supervised information in the form of labeled data in addition to the wealth of unlabeled data. Such a framework has recently attracted a considerable amount of research in remote sensing, such as the Laplacian SVM (LapSVM) [8, 66], transductive SVM [22], biased-SVM [120] and graph-based methods [71]. Even though the above mentioned algorithms exhibit good performance in classifying hyperspectral images, most of them are based on the assumption that spectrally similar instances should share the same label. However in practice, we may have very different spectra corresponding to the same material, which sometimes makes the above strict assumption no longer valid. Moreover, in most recent hyperspectral classification approaches [99, 117], the spatial information is exploited together with the spectral features, encouraging pixels in the local neighborhood to have similar labels. The spatial smoothness assumption holds well in the homogeneous regions of hyperspectral images. However, conventional approaches often fail to capture the spatial variability of spectral signatures, e.g., on the border of regions of different classes.

In this section, we introduce a hyperspectral image classification method, tackling the problems imposed by the special characteristics of hyperspectral images, namely, high-input dimension of pixels, low number of labeled samples, and spatial variability of the spectral signatures. To this end, the proposed method has the following characteristics and technical contributions:

- **Semi-supervised:** Extending the task-specific dictionary learning formulation in [113] to the semi-supervised framework for hyperspectral classification, the huge number of unlabeled samples in the image are exploited together with a limited amount of labeled samples to improve the classification performance in a task-specific setting.

- **Joint optimization of feature extraction and classification**: Almost all prior research on hyperspectral classifier design can be viewed as the combinations of two independent parts: 1) extraction of features; 2) a training procedure for designing the classifier. Although in some prior work raw spectral pixels are used directly, it is widely recognized that features extracted from the input pixels, such as the sparse code, often promote a more discriminative and robust classification [170]. However, to consider the two stages separately typically leads to a sub-optimal performance, because the extracted features are not optimized for the best performance of the following classification step. In this section, we jointly optimize the classifier parameters and dictionary atoms. This is different from the classical data-driven feature extraction approach [33] that only tries to reconstruct the training samples well. Our joint task-specific formulation ensures that the learned sparse code features are optimal for the classifier.

- **Incorporation of spatial information**: We incorporate spatial information by adding a spatial Laplacian regularization [8] to the probabilistic outputs of the classifier, i.e., the likelihood of the predicted labels. This is more flexible than the popular "naive" Laplacian smoothness constraint, that simply forces all pixels in a local window to have similar learned features.

A novel formulation of bi-level optimization is designed to meet our requirements [39, 176], which is solved by a stochastic gradient descent algorithm [21]. The proposed method is then evaluated on three popular datasets and we see an impressive improvement in performance on all of them. Even for quite ill-posed classification problems, i.e., very small number of high dimensional labeled samples, the proposed method gains a remarkable and stable improvement in performance over comparable methods.

Section 2.1.2 manifests a step-by-step construction of our formulation in detail, followed by the optimization algorithm to solve it. Section 2.1.3 discusses the classification results of the proposed method in comparison to several other

competitive methods, with a wide range of available labeled samples. It also investigates the influences of both the unlabeled samples and dictionary atoms on the classifier's performance, as well as the discriminability of the obtained dictionary. Section 2.1.4 includes some concluding remarks and the future work.

2.1.2 Formulation and Algorithm

Notations

Consider a hyperspectral image $\mathbf{X} \in R^{m \times n}$ of n pixels, each consisting of an m-dimensional spectral vector. Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]$ denote the pixel set in a hyperspectral image, with each spectral pixel $\mathbf{x}_i \in R^{m \times 1}, i = 1, 2, \cdots, n$. For all the corresponding labels $\mathbf{y} = [y_1, y_2, \cdots, y_n]$, we assume l labels $[y_1, y_2, \cdots, y_l]$ are known, constituting a labeled training set $\mathbf{X}_l = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_l]$, while making $\mathbf{X}_u = [\mathbf{x}_{l+1}, \mathbf{x}_{l+2}, \cdots, \mathbf{x}_n]$ the unlabeled training set with $u = n - l$. We assume that the number of labeled samples is uniformly selected for each class. This means for a K-class classification, each class has $l_c = \frac{l}{K}$ labeled samples.

Without loss of generality, we let all $y_i \in \{-1, 1\}$ to focus on discussing a binary classification. However, the proposed classifier can be naturally extended to a multi-class case, by either replacing the binary classifier with the multi-class classifier (e.g., soft-max classifier [58]), or adopting the well-known one-versus-one or one-versus-all strategy.

Our goal is to jointly learn a dictionary $\mathbf{D}$ consisting of a set of basis for extracting the sparse code (feature vector), and the classification parameter $\mathbf{w}$ for a binary classifier applied to the extracted feature vector, while guaranteeing them to be optimal to each other.

Joint Feature Extraction and Classification

Sparse Coding for Feature Extraction In [33], the authors suggest that the spectral signatures of pixels belonging to the same class are assumed to approximately lie in a low-dimensional subspace. Pixels can be compactly represented by only a few sparse coefficients (sparse code). In this part, we adopt the sparse code as the input features, since extensive literature has examined the outstanding effect of SRC for a more discriminative and robust classification [170].

We assume that all the data samples $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n], \mathbf{x}_i \in R^{m\times 1}, i = 1, 2, \cdots, n$, are encoded into their corresponding sparse codes $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n]$, $\mathbf{a}_i \in R^{p\times 1}, i = 1, 2, \cdots, n$, using a learned dictionary $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_p]$, where $\mathbf{d}_i \in R^{m\times 1}, i = 1, 2, \cdots, p$ are the learned atoms. It should be noted that the initial dictionary is generated by assigning equal number of atoms to each class. That means for a K-class classification, there are $p_c = \frac{p}{K}$ atoms assigned to each class in a dictionary consisting of p atoms.

The sparse representation is obtained by the following convex optimization

$$\mathbf{A} = \arg\min_{\mathbf{A}} \tfrac{1}{2}||\mathbf{X} - \mathbf{D}\mathbf{A}||_F^2 + \lambda_1 \textstyle\sum_i ||\mathbf{a}_i||_1 + \lambda_2||\mathbf{A}||_F^2, \tag{2.1}$$

or rewritten in a separate form for each $\mathbf{x}_i$

$$\mathbf{a}_i = \arg\min_{\mathbf{a}_i} \tfrac{1}{2}||\mathbf{x}_i - \mathbf{D}\mathbf{a}_i||_2^2 + \lambda_1||\mathbf{a}_i||_1 + \lambda_2||\mathbf{a}_i||_2^2. \tag{2.2}$$

Note $\lambda_2 > 0$ is necessary for proving the differentiability of the objective function (see [2.1.1] in Section 2.1.5). However, setting $\lambda_2 = 0$ work well in practice [113].

Obviously, the effect of sparse coding (3.13) largely depends on the quality of dictionary $\mathbf{D}$. The authors in [33] suggest constructing the dictionary by directly selecting atoms from the training samples. More sophisticated methods are widely used in SRC literature, discussing how to learn a more compact and effective dictionary from a given training dataset, e.g., the K-SVD algorithm [3].

We recognize that many structured sparsity constraints (priors) [33, 152] can also be considered for dictionary learning. They usually exploit the correlations among the neighboring pixels or their features. For example, the SRC dictionary has an inherent group-structured property since it is composed of several class-wise sub-dictionaries, i.e., the atoms belonging to the same class are grouped together to form a sub-dictionary. Therefore, it would be reasonable to enforce each pixel to be compactly represented by groups of atoms instead of individual ones. This could be accomplished by encouraging coefficients of only certain groups to be active, like the Group Lasso [144]. While the performance may be improved by enforcing structured sparsity priors, the algorithm will be considerably more complicated. Therefore, we do not take into account any structured sparsity prior here, and leave them for our future study.

Task-Specific Loss Functions for Classification Classical loss functions in SRC are often defined by the reconstruction error of data samples [33, 96]. The performances of such learned classifiers highly hinge on the quality of the input features, which is only sub-optimal without the joint optimization with classifier parameters. In [128], the authors study a straightforward joint representation and classification framework, by adding a penalty term to the classification error in addition to the reconstruction error. The authors in [80, 182] propose to enhance the dictionary's representative and discriminative power by integrating both the discriminative sparse-code error and the classification error into a single objective function. The approach jointly learns a single dictionary and a predictive linear classifier. However, being a semi-supervised method, the unlabeled data does not contribute much to promoting the discriminative effect in [182], as only the reconstruction error is considered on the unlabeled set except for an "expansion" strategy applied to a small set of highly-confident unlabeled samples.

In order to obtain an optimal classifier with regard to the input feature, we exploit a task-specific formulation which aims to minimize a classification-oriented loss [113]. We incorporate the sparse codes $\mathbf{a}_i$, which are dependent on the atoms of the dictionary $\mathbf{D}$ that are to be learned, into the training of the classifier parameter $\mathbf{w}$. The logistic loss is used in the objective function for the classifier. We recognize that the proposed formulation can be easily extended to other classifiers, e.g., SVM. The loss function for the labeled samples is directly defined by the logistic loss

$$L(\mathbf{A}, \mathbf{w}, \mathbf{x}_i, y_i) = \sum_{i=1}^{l} \log(1 + e^{-y_i \mathbf{w}^T \mathbf{a}_i}). \tag{2.3}$$

For unlabeled samples, the label of each $\mathbf{x_i}$ is unknown. We propose to introduce the predicted confidence probability p_{ij} that sample $\mathbf{x_i}$ has label y_j (y_j=1 or -1), which is naturally set as the likelihood of the logistic regression

$$p_{ij} = p(y_j|\mathbf{w}, \mathbf{a}_i, \mathbf{x}_i) = \frac{1}{1+e^{-y_j \mathbf{w}^T \mathbf{a}_i}}, \quad y_j = 1 \quad \text{or} quad - 1. \tag{2.4}$$

The loss function for the unlabeled samples then turns into an entropy-like form

$$U(\mathbf{A}, \mathbf{w}, \mathbf{x}_i) = \sum_{i=l+1}^{l+u} \sum_{y_j} p_{ij} L(\mathbf{a}_i, \mathbf{w}, \mathbf{x}_i, y_j), \tag{2.5}$$

which is a weighted sum of loss under different classification outputs y_j.

Furthermore, we can similarly define p_{ij} for the labeled sample $\mathbf{x}_i$, that is 1

when y_j is the given correct label y_i and 0 elsewhere. The joint loss functions for all the training samples can thus be written into a unified form

$$T(\mathbf{A}, \mathbf{w}) = \sum_{i=1}^{l+u} \sum_{y_j} p_{ij} L(\mathbf{a}_i, \mathbf{w}, \mathbf{x}_i, y_j). \tag{2.6}$$

A semi-supervised task-specific formulation has also been proposed in [113]. However, it is posed as a naive combination of supervised and unsupervised steps. The unlabeled data are only used to minimize the reconstruction loss, without contributing to promoting the discriminative effect. In contrast, our formulation (2.6) clearly distinguishes itself by assigning an adaptive confidence weight (2.25) to each unlabeled sample, and minimizes a classification-oriented loss over both labeled and unlabeled samples. By doing so, unlabeled samples also contribute to improving the discriminability of learned features and classifier, jointly with the labeled samples, rather than only optimized for reconstruction loss.

Spatial Laplacian Regularization We first introduce the weighting matrix $\mathbf{G}$, where G_{ik} characterizes the similarity between a pair of pixels $\mathbf{x}_i$ and $\mathbf{x}_k$. We define G_{ik} in the form of shift-invariant bilateral Gaussian filtering [155] (with controlling parameters σ_d and σ_s)

$$G_{ik} = \exp(-\tfrac{d(\mathbf{x}_i, \mathbf{x}_k)}{2\sigma_d^2}) \cdot \exp(-\tfrac{||\mathbf{x}_i - \mathbf{x}_k||_2^2}{2\sigma_s^2}), \tag{2.7}$$

which measures both the spatial Euclidean distance ($d(\mathbf{x}_i, \mathbf{x}_k)$) and the spectral similarity between an arbitrary pair of pixels in a hyperspectral image. Larger G_{ik} represents higher similarity and vice versa. Further, rather than simply enforcing pixels within a local window to share the same label, G_{ik} is defined over the whole image and encourages both spatially neighboring and spectrally similar pixels to have similar classification outputs. It makes our spatial constraints much more flexible and effective. Using the above similarity weights, we define the spatial Laplacian regularization function

$$S(\mathbf{A}, \mathbf{w}) = \sum_{i=1}^{l+u} \sum_{y_j} \sum_{k}^{l+u} G_{ik} ||p_{ij} - p_{kj}||_2^2). \tag{2.8}$$

Bi-level Optimization Formulation

Finally, the objective cost function for the joint minimization formulation can be expressed by the following bi-level optimization (the quadratic term of $\mathbf{w}$ is to

avoid overfitting):

$$\begin{aligned} \min_{\mathbf{D},\mathbf{w}} \quad & T(\mathbf{A},\mathbf{w}) + S(\mathbf{A},\mathbf{w}) + \tfrac{\lambda}{2}||\mathbf{w}||_2^2 \\ s.t. \quad & \mathbf{A} = \arg\min_{\mathbf{A}} \tfrac{1}{2}||\mathbf{X} - \mathbf{D}\mathbf{A}||_F^2 + \lambda_1 \textstyle\sum_i ||\mathbf{a}_i||_1 + \lambda_2 ||\mathbf{A}||_F^2. \end{aligned} \tag{2.9}$$

Bi-level optimization [39] has been investigated in both theory and application sides. In [176], the authors propose a general bi-level sparse coding model for learning dictionaries across coupled signal spaces. Another similar formulation has been studied in [113] for general regression tasks.

In the testing stage, each test sample is first represented by solving (2.2) over the learned $\mathbf{D}$. The resulting sparse coefficients are fed to the trained logistic classifier with the previously learned $\mathbf{w}$. The test sample is classified into the class of the highest output probability (2.25).

Algorithm

Built on the methodologies similar to [113] and [176], we solve (2.9) using a projected first order stochastic gradient descent (SGD) algorithm, whose detailed steps are outlined in Algorithm 1. At a high level overview, it consists of an outer stochastic gradient descent loop that incrementally samples the training data. It uses each sample to approximate gradients with respect to the classifier parameter $\mathbf{w}$ and the dictionary $\mathbf{D}$, which are then used to update them. Next, we briefly explain a few key technical points of the Algorithm 1.

Stochastic Gradient Descent The stochastic gradient descent (SGD) algorithm [21] is an iterative, "on-line" approach for optimizing an objective function, based on a sequence of approximate gradients obtained by randomly sampling from the training data set. In the simplest case, SGD estimates the objective function gradient on the basis of a single randomly selected example $\mathbf{x}_t$

$$w_{t+1} = w_t - \rho_t \nabla_w F(\mathbf{x}_t, w_t), \tag{2.10}$$

where F is a loss function, w is a weight being optimized and ρ_t is a step size known as the "learning rate". The stochastic process $\{w_t, t = 1, \cdots\}$ depends upon the sequence of randomly selected examples $\mathbf{x}_t$ from the training data. It thus optimizes the empirical cost, as a good proxy for the expected cost.

Following the derivations in [113], we can show that the objective function in (2.9), denoted as $B(\mathbf{A}, \mathbf{w})$ for simplicity, is differentiable on $\mathbf{D} \times \mathbf{w}$, and that

$$\begin{aligned} \nabla_{\mathbf{w}} B(\mathbf{A}, \mathbf{w}) &= \mathbb{E}_{\mathbf{x},y}[\nabla_{\mathbf{w}} T(\mathbf{A}, \mathbf{w}) + \nabla_{\mathbf{w}} S(\mathbf{A}, \mathbf{w}) + \lambda \mathbf{w}] \\ \nabla_{\mathbf{D}} B(\mathbf{A}, \mathbf{w}) &= \mathbb{E}_{\mathbf{x},y}[-\mathbf{D}\boldsymbol{\beta}^* \mathbf{A}^{\mathbf{T}} + (\mathbf{X}_t - \mathbf{D}\mathbf{A})\boldsymbol{\beta}^{*T}], \end{aligned} \tag{2.11}$$

where $\boldsymbol{\beta}^*$ is a vector defined by the following property:

$$\begin{aligned} \boldsymbol{\beta}^*_{S^C} &= 0 \\ \boldsymbol{\beta}^*_S &= (\mathbf{D}_S^T \mathbf{D}_S + \lambda_2 \mathbf{I})^{-1} \nabla_{\mathbf{A}_S} [T(\mathbf{A}, \mathbf{w}) + S(\mathbf{A}, \mathbf{w})], \end{aligned} \tag{2.12}$$

and S are the indices of the nonzero coefficients of $\mathbf{A}$. The proof of the above equations is given in the Section 2.1.5.

Sparse Reconstruction The most computationally intensive step in Algorithm 1 is solving the sparse coding (step 3). We adopt the Feature-Sign algorithm [96] for efficiently solving the exact solution to sparse coding.

Algorithm 1 Stochastic gradient descent algorithm for solving (2.9)

Require: $\mathbf{X}, \mathbf{Y}$; λ, λ_1, λ_2, σ_d and σ_s; $\mathbf{D}_0$ and $\mathbf{w}_0$ (initial dictionary and classifier parameter); ITER (number of iterations); t_0, ρ (learning rate)

1: FOR t=1 to ITER DO
2: Draw a subset $(\mathbf{X}_t, \mathbf{Y}_t)$ from $(\mathbf{X}, \mathbf{Y})$
3: Sparse coding: computer $\mathbf{A}^*$ using Feature-Sign algorithm:
$\mathbf{A}^* = \arg\min_{\mathbf{A}} \frac{1}{2}||\mathbf{X}_t - \mathbf{D}\mathbf{A}||_2^2 + \lambda_1 \sum_i ||\mathbf{a}_i||_1 + \frac{\lambda_2}{2}||\mathbf{A}||_2^2$
4: Compute the active set S (the nonzero support of $\mathbf{A}$)
5: Compute $\boldsymbol{\beta}^*$: Set $\boldsymbol{\beta}^*_{S^C} = 0$ and $\boldsymbol{\beta}^*_S = (\mathbf{D}_S^T \mathbf{D}_S + \lambda_2 \mathbf{I})^{-1} \nabla_{\mathbf{A_S}} [T(\mathbf{A}, \mathbf{w}) + S(\mathbf{A}, \mathbf{w})]$
6: Choose the learning rate $\rho_t = \min(\rho, \rho \frac{t_0}{t})$
7: Update $\mathbf{D}$ and $\mathbf{W}$ by a projected gradient step:
$\mathbf{w} = \prod_{\mathbf{w}} [\mathbf{w} - \rho_t (\nabla_{\mathbf{w}} T(\mathbf{A}, \mathbf{w}) + \nabla_{\mathbf{w}} S(\mathbf{A}, \mathbf{w}) + \lambda \mathbf{w})]$
$\mathbf{D} = \prod_{\mathbf{D}} [\mathbf{D} - \rho_t (\nabla_{\mathbf{D}} (-\mathbf{D}\boldsymbol{\beta}^* \mathbf{A}^T + (\mathbf{X}_t - \mathbf{D}\mathbf{A})\boldsymbol{\beta}^{*T})]$
where $\prod_{\mathbf{w}}$ and $\prod_{\mathbf{D}}$ are respectively orthogonal projections on the embedding spaces of $\mathbf{w}$ and $\mathbf{D}$.
8: END FOR

Ensure: $\mathbf{D}$ and $\mathbf{w}$

Remark on SGD convergence and sampling strategy: The convergence proof of SGD [137] for non-convex problems indeed assumes three times differentiable cost functions. As a typical case in machine learning, we use SGD in a setting where it is not guaranteed to converge in theory, but behaves well in practice.

SGD algorithms are typically designed to minimize functions whose gradients have the form of an expectation. While an i.i.d. (independent and identically distributed) sampling process is required, it cannot be computed in a batch mode. In our algorithm, instead of sampling one per iteration, we adopt a mini-batch strategy by drawing more samples at a time. Authors in [113] further pointed out that solving multiple elastic-net problems with the same dictionary $\mathbf{D}$ can be accelerated by the pre-computation of the matrix $\mathbf{D}^T\mathbf{D}$. In practice, we draw a set of 200 samples in each iteration, which produces steadily good results in all our experiments under universal settings.

Strictly speaking, drawing samples from the distribution of training data should be made i.i.d. (step 2 in Algorithm 1). However, this is practically difficult since the distribution itself is typically unknown. As an approximation, samples are instead drawn by iterating over random permutations of the training set [137].

2.1.3 Experiments

In this part, we evaluate the proposed method on three popular datasets, and compare it with some related approaches in the literature, including:

- Laplacian Support Vector Machine (LapSVM) [8, 66], that is a semi-supervised extension of the SVM and applies the spatial manifold assumption to SVM. The classification is directly executed on raw pixels without any feature extraction, which follows the original setting in [10].
- Semi-supervised Classification (SSC) approach [161] that employs a modified clustering assumption.
- Semi-supervised hyperspectral image segmentation that adopts Multinomial Logistic Regression with Active Learning (MLR-AL) [98].

Regarding parameter choices of the three methods, we try our best to follow the settings in their original papers. For LapSVM, the regularization parameters γ_1, γ_2 are selected from $[10^{-5}, 10^5]$ according to a five-fold cross-validation procedure. In SSC, the width parameter of the Gaussian function is tuned using a five-fold cross-validation procedure. The parameter setting in MLR-AL follows that of the original paper [98].

Besides the above mentioned three algorithms, we also include the following algorithms in the comparison, in order to illustrate the merits of both joint optimization and spatial Laplacian regularization on the classifier outputs:

- Non-joint optimization of feature extraction and classification (Non-Joint). It refers to conducting the following two stages sequentially:

$$\begin{aligned} &\textbf{1. Feature extraction:} \\ &\mathbf{A} = \arg\min_{\mathbf{A}} \tfrac{1}{2}||\mathbf{X} - \mathbf{D}\mathbf{A}||_F^2 + \lambda_1 \textstyle\sum_i ||\mathbf{a_i}||_1 + \lambda_2 ||\mathbf{A}||_F^2. \\ &\textbf{2. Learning a classifier:} \\ &\min_{\mathbf{w}} \quad T(\mathbf{A}, \mathbf{w}) + \tfrac{\lambda}{2}||\mathbf{w}||_2^2. \end{aligned} \tag{2.13}$$

  The training of $\mathbf{D}$ is independent of the learning of the classifier parameter $\mathbf{w}$. This is different from the joint optimization of the dictionary and classifier as is done in (2.9) by the task-specific formulation.

- Non-joint optimization of feature extraction and classification, with spatial Laplacian regularization (Non-Joint + Laplacian). It is the same as the Non-Joint method except for adding a spatial Laplacian regularization term $S(\mathbf{A}, \mathbf{w})$ to the second subproblem:

$$\begin{aligned} &\textbf{1. Feature extraction:} \\ &\mathbf{A} = \arg\min_{\mathbf{A}} \tfrac{1}{2}||\mathbf{X} - \mathbf{D}\mathbf{A}||_F^2 + \lambda_1 \textstyle\sum_i ||\mathbf{a}_i||_1 + \lambda_2 ||\mathbf{A}||_F^2. \\ &\textbf{2. Learning a classifier:} \\ &\min_{\mathbf{w}} \quad T(\mathbf{A}, \mathbf{w}) + S(\mathbf{A}, \mathbf{w}) + \tfrac{\lambda}{2}||\mathbf{w}||_2^2. \end{aligned} \tag{2.14}$$

- The proposed joint method without spatial Laplacian regularization (Joint), which is done by dropping the $S(\mathbf{A}, \mathbf{W})$ term in (2.9)

$$\begin{aligned} &\min_{\mathbf{D},\mathbf{W}} \quad T(\mathbf{A}, \mathbf{w}) + \tfrac{\lambda}{2}||\mathbf{w}||_2^2 \\ &s.t. \quad \mathbf{A} = \arg\min_{\mathbf{A}} \tfrac{1}{2}||\mathbf{X} - \mathbf{D}\mathbf{A}||_F^2 + \lambda_1 \textstyle\sum_i ||\mathbf{a}_i||_1 + \lambda_2 ||\mathbf{A}||_F^2. \end{aligned} \tag{2.15}$$

- The proposed joint method with spatial Laplacian regularization (Joint + Laplacian), by minimizing our proposed bi-level formulation (2.9).

Parameter Setting For the proposed method, the regularization parameter λ in (2.9) is fixed to be 10^{-2}, and λ_2 in (2.2) is set to 0 to exploit sparsity. The elastic-net parameter λ_1 in (2.2) is generated by a cross-validation procedure, which is

similar to the one in [113]. The values of λ_1 are 0.225, 0.25, and 0.15 for the first three experiments, respectively. σ_d and σ_s in (2.7) are fixed as 3 and 3,000, respectively. The learning rate ρ is set as 1, and maximum number ITER is set as 1000 for all. Although we have verified that these choices of parameters work well in extensive experiments, we recognize that a finer tuning may further improve the performance.

In particular, we would like to mention the initializations of $\mathbf{D}$ and $\mathbf{w}$. For the two non-joint methods, $\mathbf{D}$ is initialized by solving the first subproblem (feature extraction) in (2.13) or (2.14). In this subproblem, for each class, we initialize its sub-dictionary atoms randomly. We then employ several iterations of K-SVD using only the available labeled data for that class, and finally combine all the output class-wise sub-dictionaries into a single initial dictionary $\mathbf{D}$. Next, we solve $\mathbf{A}$ based on $\mathbf{D}$, and continue to feed $\mathbf{A}$ into the second subproblems (learning a classifier) in (2.13) and (2.14) for good initializations of $\mathbf{w}$, for Non-Joint and Non-Joint + Laplacian, respectively. For the two joint methods, we use the results of Non-Joint, and Non-Joint + Laplacian, to initialize $\mathbf{D}$ and $\mathbf{w}$ of Joint and Joint + Laplacian, respectively.

The one-versus-all strategy is adopted for addressing multi-class problems, which means that we train K different binary logistic classifiers with K corresponding dictionaries for a K-class problem. For each test sample, the classifier with the maximum score will provide the class label. When the class number is large, this one-versus-all approach is more scalable than learning a single large dictionary with a multi-class loss [113], and provides better results.

For the two joint methods, we assign only five dictionary atoms per class to initialize the dictionary, which means for a K-class problem we have $p_c = 5$ and $p = 5K$ for the total dictionary size. For the two non-joint methods, fifty dictionary atoms ($p_c = 50$) are assigned per class in the first subproblems of (2.13) and (2.14), respectively. The choices of dictionary sizes for both joint and non-joint methods will be illustrated in the fourth experiment. We use the term "atoms per class" for two reasons: 1) We initialize our dictionary by first applying KSVD to each class to obtain a class-wise sub-dictionary. This helps to improve the class discriminability of the learned dictionary more than just applying KSVD to the whole data. Therefore, we need to specify how many atoms are assigned per class in the initialization stage. Note that when Algorithm 1 starts, the atoms become all entangled, and further it is impossible to identify how many (and which) atoms are representing a specific class in the final learned dictionary. 2) Each dataset

Table 2.1: Overall classification results (%) for the AVIRIS Indiana Pines data with different numbers of labeled samples per class (u = ALL, $\lambda = 10^{-2}$, λ_1 = 0.225, λ_2 = 0, ρ = 1, σ_d = 3, σ_s = 3,000)

l_c	2	3	4	5	6	7	8	9	10
LapSVM [8]	57.80	61.32	63.1	66.39	68.27	69.00	70.15	70.04	70.73
SSC [161]	44.61	56.98	58.27	60.56	60.79	64.19	66.81	69.40	70.50
MLR+AL [98]	52.34	56.16	59.21	61.47	65.16	69.21	72.14	73.89	74.43
Non-Joint ($p_c = 50$)	63.72	69.21	71.87	76.88	79.04	81.81	85.23	87.77	88.54
Non-Joint + L ($p_c = 50$)	66.89	72.37	75.33	78.78	81.21	84.98	87.25	88.61	89.88
Joint ($p_c = 5$)	69.81	76.03	80.42	82.91	84.81	85.76	86.95	87.54	89.31
Joint + L ($p_c = 5$)	**76.55**	**80.63**	**84.28**	**86.33**	**88.27**	**90.68**	**91.87**	**92.53**	**93.11**

has a different number of classes, and empirically, more classes demand more dictionary atoms to represent. Note, however, if we assign atoms in proportion to the number of samples per class, some minor classes will tend to be severely underrepresented.

In the first three experiments, we use all the unlabeled pixels (denoted as "ALL") from each hyperspectral image for semi-supervised training. Later, we discuss how unlabeled samples u will influence the classification accuracy. The last experiment will provide a visualized example to manifest that the proposed method indeed leads to a more discriminative dictionary that contributes to a higher classification accuracy.

Classification Performance on AVIRIS Indiana Pines Data

The AVIRIS sensor generates 220 bands across the spectral range from 0.2 to 2.4 μm. In the experiment, the number of bands is reduced to 200 by removing 20 water absorption bands. The AVIRIS Indiana Pines hyperspectral image has the spatial resolution of 20m and 145×145 pixels. It contains 16 classes, most of which are different types of crops (e.g., corn, soybeans, and wheat). The ground-truth classification map is shown in Figure 2.1 (a).

Table 2.1 evaluates the influence of the number of labeled samples per class l_c on the classification of AVIRIS Indiana Pines data, with l_c varying from 2 to 10. The dictionary consists of only p = 80 atoms to represent all the 16 classes for the joint methods, and p = 800 for the non-joint methods. The bold value in each column indicates the best result among all the seven methods. As can be seen from the table, the classification results improve for all the algorithms with the increase in the number of labeled samples. The last two methods, i.e., Joint and Joint +

Laplacian, outperform the other five methods in terms of overall accuracy (OA) significantly. It is also observed that the Joint + Laplacian method obtains further improvement over the Joint method, showing the advantage of spatial Laplacian regularization. Amazingly, we notice that even when there are as few as three samples per class, the OA of the proposed method (Joint + Laplacian) is still higher than 80%.

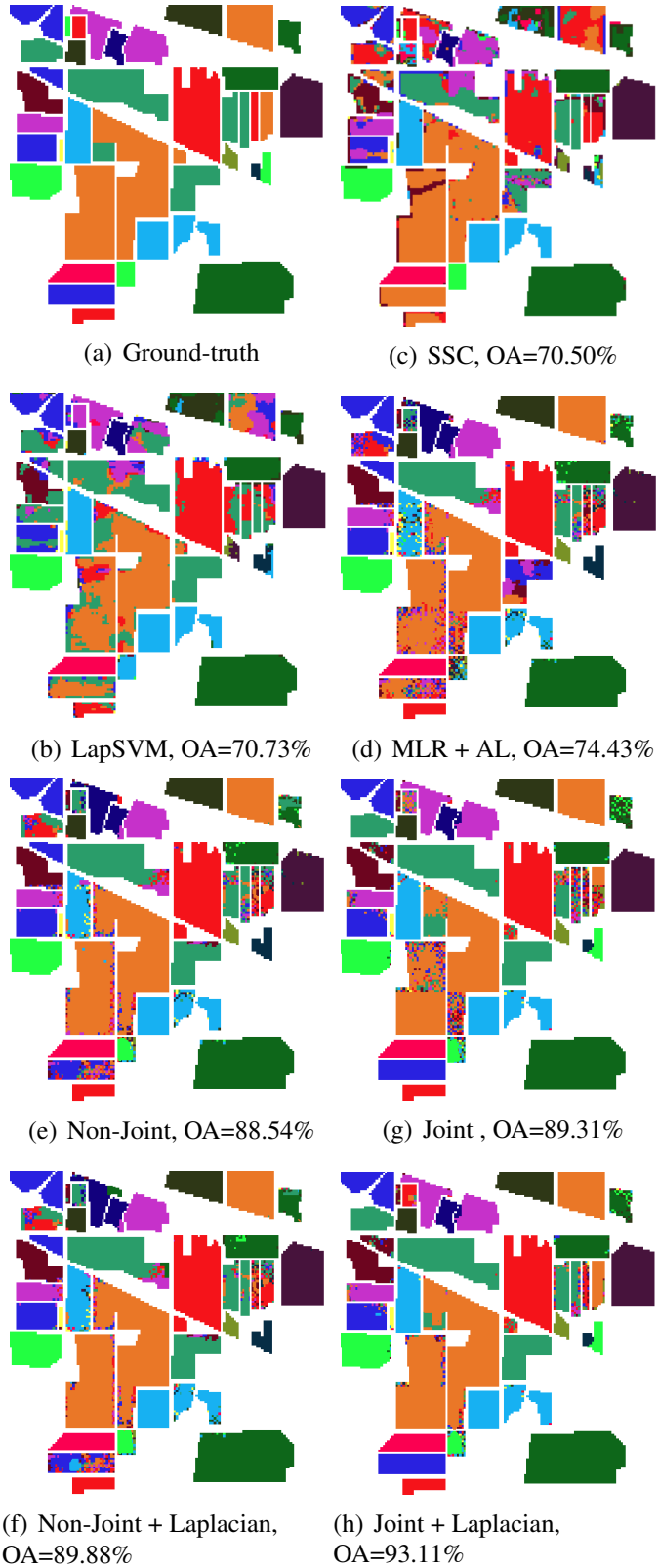
Figure 2.1 demonstrates the classification maps obtained by all the methods when 10 labeled samples are used per class. The proposed method, either with or without spatial regularization, obtains fewer misclassifications compared with the other methods. What is more, the homogeneous areas in (h) are significantly better preserved than that in (g), which again confirms the effectiveness of the spatial Laplacian regularization on the output of the classifier. Figure 2.2 visually demonstrates that along with the increase of l_c, the classification results gradually improve, and both the regional and scattered misclassifications are reduced dramatically.

Classification Performance on AVIRIS Salinas Data

This dataset is collected over the Valley of Salinas, Southern California, in 1998. This hyperspectral image is of size 217×512, with 16 different classes of objects in total. In our experiment, a nine-class subset is considered, including vegetables, bare soils, and vineyard fields. The ground-truth classification map is shown in Figure 2.3 (a). As AVIRIS Salinas is recognized to be easier for classification than AVIRIS Indian Pines, all methods obtain high OAs as listed in Table 2.2, while the Joint + Laplacian method marginally stands out. When we turn to Figure 2.3 (b)-(h) for the comparison in classification maps, however, the Joint + Laplacian method is visually much superior in reducing scattered misclassifications.

Classification Performance on University of Pavia Data

The ROSIS sensor collected this data during a flight campaign over the Pavia district in north Italy. 103 spectral bands were used for data acquisition in this dataset, comprising of 610×610 pixel images with a geometric resolution of 1.3m. A few samples contain no information and were discarded before the classification. The ground truth data shows a total of nine distinct classes, and has been portrayed visually in Figure 2.4 (a). Similar conclusions can be attained from both Table



(a) Ground-truth

(c) SSC, OA=70.50%

(b) LapSVM, OA=70.73%

(d) MLR + AL, OA=74.43%

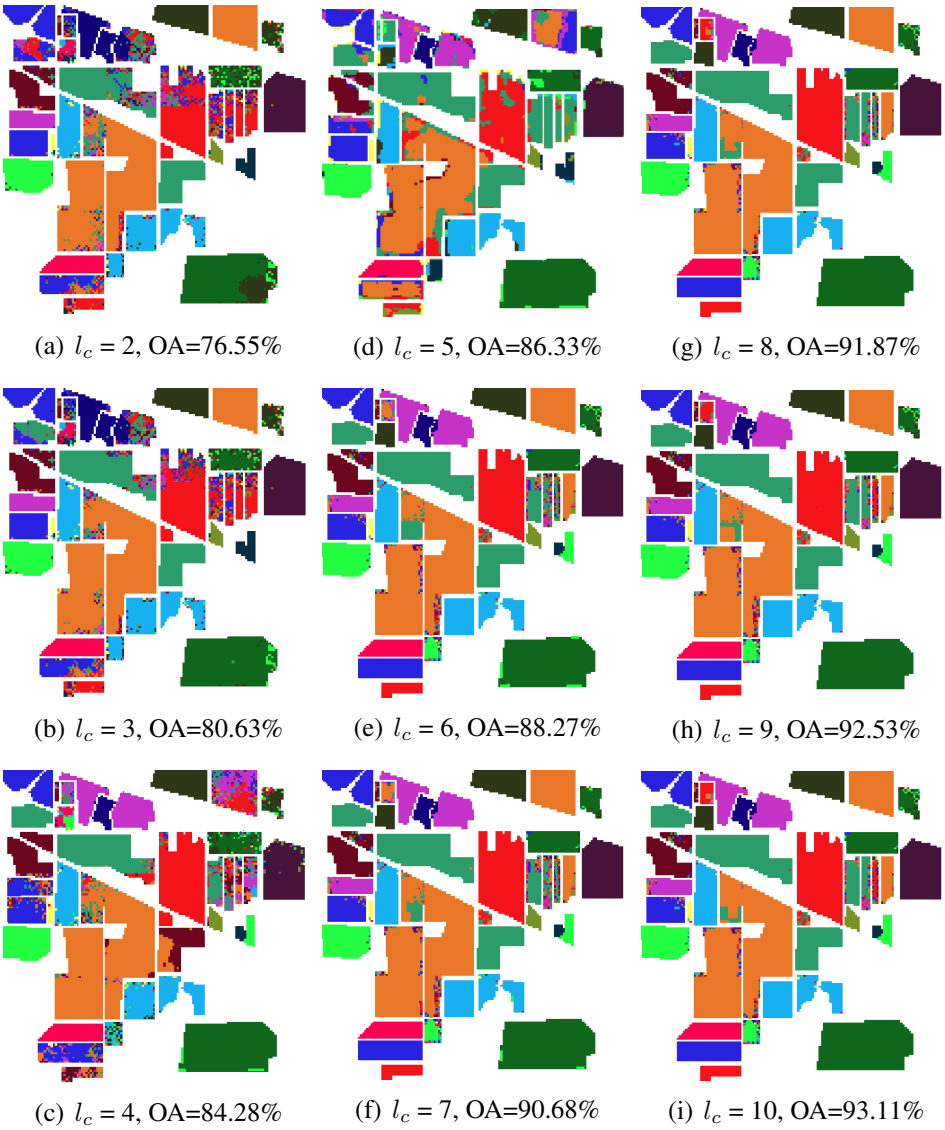
(e) Non-Joint, OA=88.54%

(g) Joint , OA=89.31%

(f) Non-Joint + Laplacian, OA=89.88%

(h) Joint + Laplacian, OA=93.11%

Figure 2.1: Classification maps for the AVIRIS Indian Pines scene using different methods with 10 labeled samples per class.

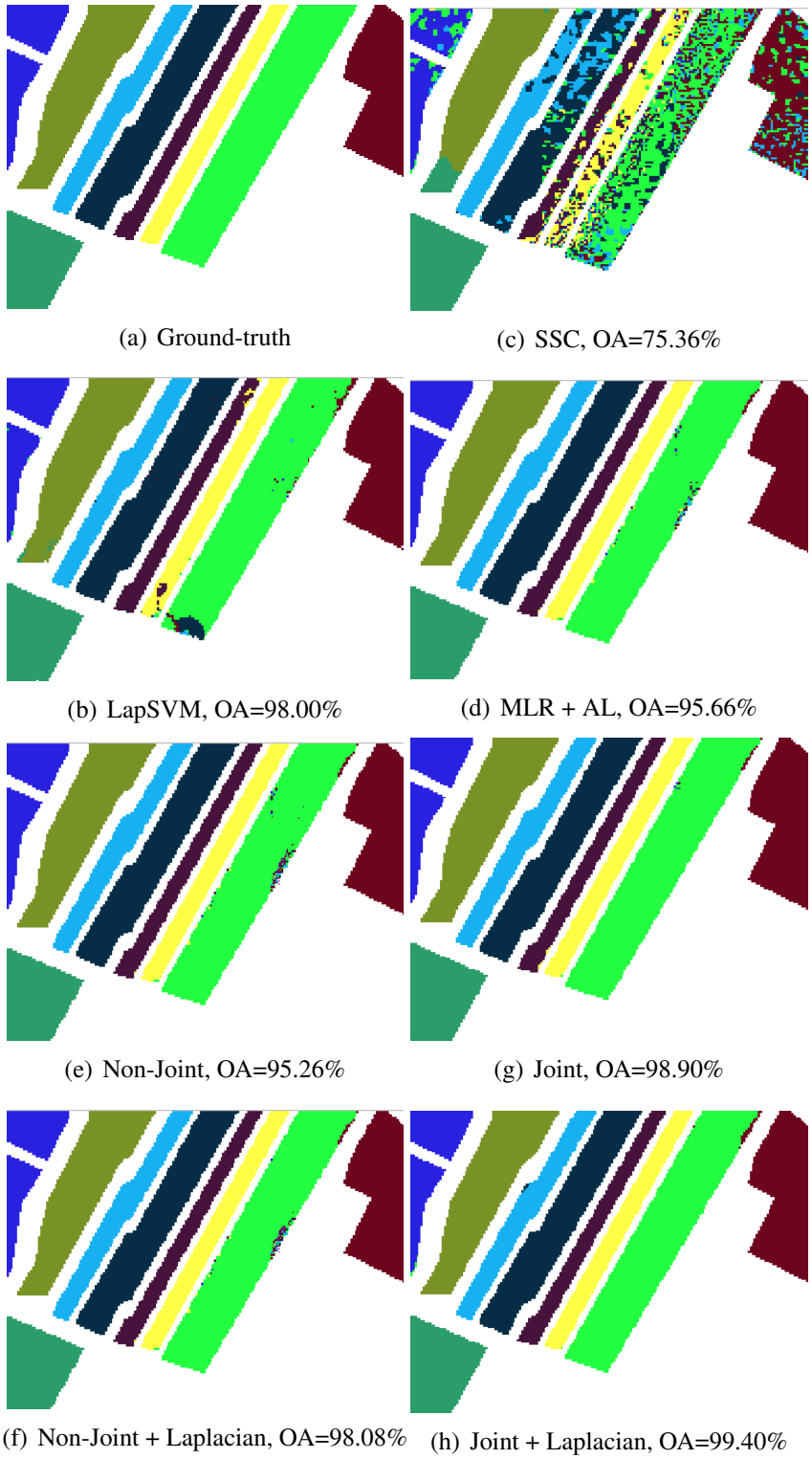


(a) l_c = 2, OA=76.55% (d) l_c = 5, OA=86.33% (g) l_c = 8, OA=91.87%

(b) l_c = 3, OA=80.63% (e) l_c = 6, OA=88.27% (h) l_c = 9, OA=92.53%

(c) l_c = 4, OA=84.28% (f) l_c = 7, OA=90.68% (i) l_c = 10, OA=93.11%

Figure 2.2: Classification maps for the AVIRIS Indian Pines scene using the proposed Joint + Laplacian method with different numbers of labeled samples per class.



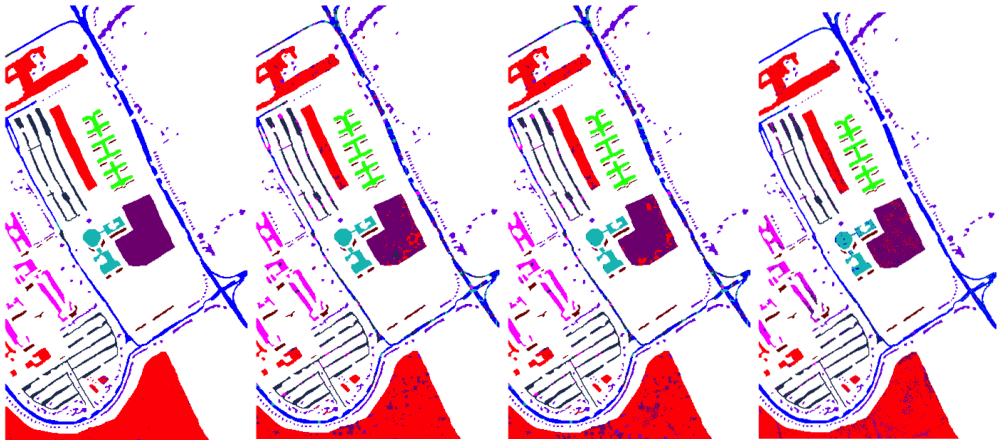
(a) Ground-truth (c) SSC, OA=75.36%

(b) LapSVM, OA=98.00% (d) MLR + AL, OA=95.66%

(e) Non-Joint, OA=95.26% (g) Joint, OA=98.90%

(f) Non-Joint + Laplacian, OA=98.08% (h) Joint + Laplacian, OA=99.40%

Figure 2.3: Classification maps for the AVIRIS Salinas scene using different methods with 10 labeled samples per class.



(a) Ground-truth .

(c) SSC, OA=80.95%

(e) Non-Joint, OA=81.26%

(g) Joint, , OA=88.08%

(b) LapSVM, OA=79.88%

(d) MLR + AL, OA=85.53%

(f) Non-Joint + Laplacian, OA=88.78%

(h) Joint + Laplacian, OA=90.41%

Figure 2.4: Classification maps for the University of Pavia scene using different methods with 10 labeled samples per class.

Table 2.2: Overall classification results (%) for the AVIRIS Salinas data with different numbers of labeled samples per class (u = ALL, $\lambda = 10^{-2}$, $\lambda_1 = 0.25$, $\lambda_2 = 0$, $\rho = 1$, $\sigma_d = 3$, $\sigma_s = 3{,}000$)

l_c	2	3	4	5	6	7	8	9	10
LapSVM [8]	**90.77**	91.53	**92.95**	93.50	94.77	95.08	96.05	97.17	98.00
SSC [161]	59.47	61.84	64.90	67.19	71.04	73.04	72.81	73.51	75.36
MLR+AL [98]	78.98	82.32	84.31	86.27	85.86	89.41	92.27	93.78	95.66
Non-Joint ($p_c = 50$)	85.88	87.21	89.29	90.76	91.42	92.87	93.95	94.78	95.26
Non-Joint + L ($p_c = 50$)	87.67	89.28	91.54	92.67	93.93	95.28	96.79	97.83	98.08
Joint ($p_c = 5$)	89.71	90.03	91.42	92.12	93.25	94.54	96.05	97.45	98.90
Joint + L ($p_c = 5$)	90.65	**91.59**	92.28	**93.63**	**95.22**	**96.58**	**97.81**	**98.53**	**99.40**

Table 2.3: Overall classification results (%) for the University of Pavia Data data with different numbers of labeled samples per class (u=ALL, $\lambda = 10^{-2}$, λ_1=0.15, λ_2=0, ρ=1, σ_d=3, σ_s=3,000)

l_c	2	3	4	5	6	7	8	9	10
LapSVM [8]	64.77	67.83	69.25	71.05	72.97	74.38	76.75	78.17	79.88
SSC [161]	69.54	72.84	74.69	76.21	77.24	78.43	79.81	80.25	80.95
MLR+AL [98]	76.27	78.66	79.30	80.22	81.36	82.41	83.27	84.78	85.53
Non-Joint ($p_c = 50$)	74.21	75.27	76.22	76.83	78.24	79.51	79.67	80.83	81.26
Non-Joint + L ($p_c = 50$)	**79.23**	80.26	82.58	**84.07**	**86.21**	86.88	87.56	88.23	88.78
Joint ($p_c = 5$)	74.21	76.73	79.24	80.82	82.35	84.54	86.97	87.27	88.08
Joint + L ($p_c = 5$)	78.56	**80.29**	**82.84**	83.76	85.12	**87.58**	**88.33**	**89.52**	**90.41**

2.3 and Figure 2.4, that once again verify the merits of both the joint optimization framework and spatial regularization.

Influences of Dictionary Size

To study the influence of the dictionary size, we report the performance on the AVIRIS Indian Pines dataset for different dictionary sizes, with both Joint and Joint + Laplacian methods. Moreover, we also include Non-Joint and Non-Joint + Laplacian methods into the same comparison experiments, in order to validate their optimal dictionary sizes. It is recognized that a larger dictionary usually means a better performance, but at a higher computational cost. Setting the size of the dictionary is therefore often a trade-off between the desired quality of the classification results and computational efficiency of the algorithm.

Table 2.4, as well as Figure 2.5, proves that our proposed method is quite robust to the dictionary size. The performance is only a little bit low even when there are only three dictionary atoms per class (dictionary with p = 48 atoms). This is because the overall dictionary is too small to capture all the features in the training

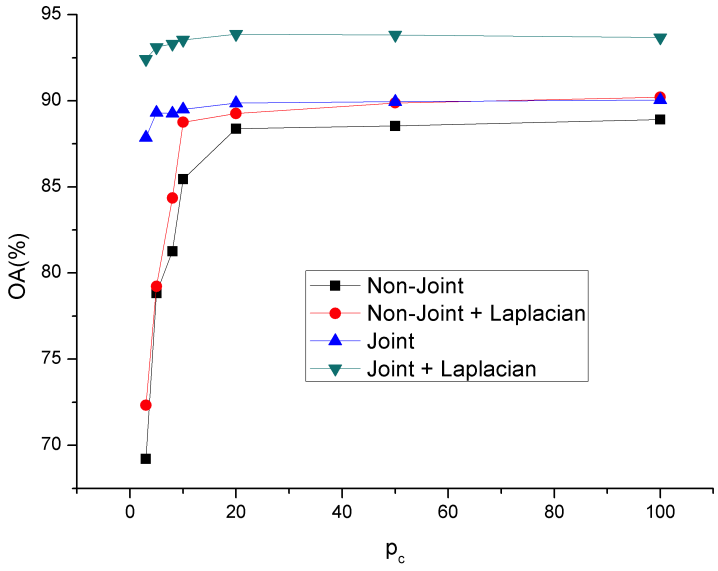


Figure 2.5: Classification results for the AVIRIS Indian Pines data with different p_c (fix l = 160, u = ALL).

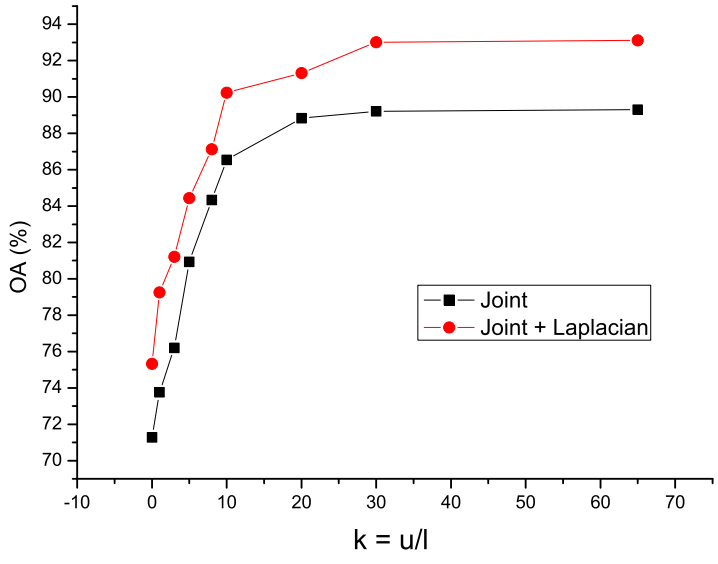


Figure 2.6: Classification results for the AVIRIS Indian Pines data with different u (fix l = 160, p_c = 5).

Table 2.4: Overall classification results (%) for the AVIRIS Indian Pines data with different numbers of dictionary atoms per class (Fix l_c = 10, u = ALL)

p_c	3	5	8	10	20	50	100
Non-Joint	69.21	78.83	81.27	85.45	88.38	88.54	88.91
Non-Joint + L	72.33	79.22	84.35	88.76	89.27	89.88	90.21
Joint	87.87	89.31	89.27	89.51	89.87	89.95	90.05
Joint + L	92.42	93.11	93.30	93.53	93.87	93.82	93.67

Table 2.5: Overall classification results (%) for the AVIRIS Indian Pines data with different numbers of unlabeled samples (Fix l = 160, p_c = 5)

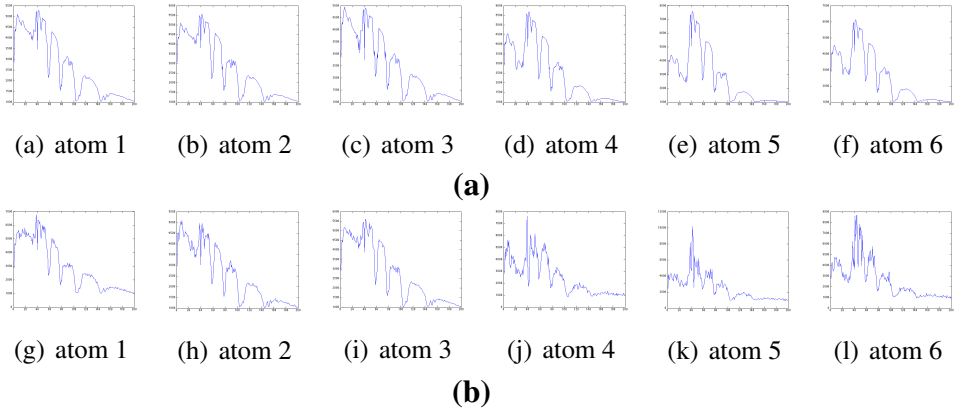
$u = kl$	0l	1l	3l	5l	8l	10l	20l	30l	All
Joint	71.28	73.76	76.20	80.94	84.34	86.54	88.83	89.21	89.31
Joint + Laplacian	75.33	79.25	81.21	84.44	87.12	90.24	91.32	93.02	93.11

data. However, a good classification accuracy can always be achieved with a relatively small dictionary, even with only five atoms per class (dictionary with p = 80 atoms), which indicates that both joint methods can obtain good performance with a computationally reasonable dictionary size. In contrast, the performances of two non-joint methods turn dramatically poorer when the dictionary is highly compact. As the dictionary size is increased and finally turns overcomplete, the performance differences with joint methods become relatively smaller but still quite notable even under p_c = 100, where the Joint + Laplacian method maintains a more than 3% advantage in overall accuracy over its counterpart Non-Joint + Laplacian method. While the non-joint methods have to sacrifice computational efficiency (due to a large overcomplete dictionary) for better accuracy, we can use a compact dictionary and avoid the heavy computational cost in the proposed joint methods.

Influences of Unlabeled Samples

In this part, we evaluate how the number of unlabeled samples for training will influence the resulting accuracy, via experiments on AVIRIS Indian Pines dataset as well. In Table 2.5, we demonstrate the influence of changing u, when fixing l_c at 10 in order to have a total of l = 160 labeled samples. To be more intuitive, we express u as k times the value of l, i.e., $u = kl$, $k \in Z$, and vary k from 0

(which corresponds to the supervised task-specific case, with Laplacian regularization), until when u reaches the total number of unlabeled samples (denoted as "ALL"). As a consequence, with the same number of labeled samples, increasing the number of unlabeled samples u leads to a monotonic increase in the classification accuracy, which validates the advantage of semi-supervised classification. Such an improvement becomes especially remarkable even when the amount of labeled samples is very small, as can also be seen from the plot in Figure 2.6.



(a) atom 1 (b) atom 2 (c) atom 3 (d) atom 4 (e) atom 5 (f) atom 6

(a)

(g) atom 1 (h) atom 2 (i) atom 3 (j) atom 4 (k) atom 5 (l) atom 6

(b)

Figure 2.7: The spectral signatures of (a) the atoms of the K-SVD unsupervised dictionary in the top row (OA=84.45%), and (b) the atoms of the task-specific semi-supervised dictionary in the bottom row (OA=92.05%). For each of the figures, the X-axis and Y-axis stand for the spectral band and radiance value, respectively.

Discriminability of Dictionary

In the proposed method, we jointly optimize the dictionary and the classifier together. The learned dictionary is thus expected to consist of highly discriminative bases for the classification. This fact has already been implied by the performances in the first three experiments, and here we are going to verify the discriminative property directly by visualizing the dictionary atoms.

We choose two classes, the 3rd class "Corn-min" and the 6th class "Grass/-Trees", from the AVIRIS Indian Pines dataset. Each class has only 10 labeled samples while being abundant in unlabeled samples. We first apply K-SVD to each of the classes separately, and obtain a class-wise sub-dictionary of three atoms. Then we concatenate the two sub-dictionaries into a K-SVD dictionary of six atoms, and follow the Non-Joint +Laplacian method (2.14) to do a binary clas-

sification. Next, a task-specific dictionary is generated by our proposed method (2.9), with the same dictionary size and initialized by the K-SVD dictionary as described previously.

We visualize our results in Figure 2.7 by plotting the spectral signatures of all the six atoms in the dictionaries. For each figure, the X-axis and Y-axis stand for the spectral band and radiance value, respectively. The atoms generated by K-SVD are plotted in the top row, while the atoms by the proposed method are plotted in the bottom row. Comparing to the K-SVD results, the dictionary atoms learned by the proposed method looks more "dissimilar" from each other, demonstrating visually a higher discriminability. As a consequence, the classification accuracy of our proposed method achieves 92.05%, which remarkably outperforms the 84.45% accuracy by the Non-Joint +Laplacian method.

2.1.4 Conclusion

In this section, we develop a semi-supervised hyperspectral image classification method based on task-specific dictionary learning and spatial Laplacian regularization on the output of the logistic regression classifier. We jointly optimize both the dictionary for feature extraction and the associated classifier parameter, while both the spectral and the spatial information are explored to improve the classification accuracy. Experimental results verify the superior performance of our proposed method on three popular datasets, both quantitatively and qualitatively. A good and stable accuracy is produced in even quite ill-posed problem settings (high dimensional spaces with small number of labeled samples). In the future, we would like to explore the applications of the proposed method to general image classification and segmentation problems.

2.1.5 Proof of (2.11)

Denote $\mathbf{X} \in \mathcal{X}$, $\mathbf{y} \in \mathcal{Y}$ and $\mathbf{D} \in \mathcal{D}$. Let the objective function $B(\mathbf{A}, \mathbf{w})$ in (2.9) be denoted as B for short. The differentiability of B with respect to $\mathbf{w}$ is easy to show, using only the compactness of $\mathcal{X}$ and $\mathcal{Y}$, as well as the fact that B is twice differentiable.

We will therefore focus on showing that B is differentiable with respect to $\mathbf{D}$, which is more difficult since $\mathbf{A}$, and thus $\mathbf{a}_i$, is not differentiable everywhere.

Without loss of geniality, we use a vector $\mathbf{a}$ instead of $\mathbf{A}$ for simplifying the derivations hereinafter. In some cases, we may equivalently express $\mathbf{a}$ as $\mathbf{a}(\mathbf{D}, \mathbf{w})$ in order to emphasize the functional dependence.

We recall the following lemma [2.1.1] that is proved in [113]:

Theorem 2.1.1 (Regularity of the elastic net solution)**.** *Consider the formulation in (3.13). Assume $\lambda_2 > 0$, and both $\mathcal{X}$ and $\mathcal{Y}$ are compact. Then,*

- $\mathbf{a}$ *is uniformly Lipschitz on $\mathcal{X} \times \mathcal{D}$*
- *Let $\mathbf{D} \in \mathcal{D}$, σ be a positive scalar and $\mathbf{s}$ be a vector in $\{-1, 0, 1\}^p$. Define $K_s(\mathbf{D}, \sigma)$ as the set of vectors $\mathbf{x}$ satisfying for all j in $\{1, ..., p\}$,*

$$\begin{array}{l} |\mathbf{d}_j^T(\mathbf{x} - \mathbf{D}\mathbf{a}) - \lambda_2 \mathbf{a}[j]| \leq \lambda_1 - \sigma \quad if \quad \mathbf{s}[j] = 0 \\ \mathbf{s}[j]\mathbf{a}[j] \geq \sigma \quad if \quad \mathbf{s}[j] \neq 0. \end{array} \tag{2.16}$$

  Then there exists $\kappa > 0$ independent of $\mathbf{s}, \mathbf{D}$ and σ so that for all $\mathbf{x} \in K_s(\mathbf{D}, \sigma)$, the function $\mathbf{a}$ is twice continuously differentiable on $B_{\kappa\sigma}(\mathbf{x}) \times B_{\kappa\sigma}(\mathbf{D})$, where $B_{\kappa\sigma}(\mathbf{x})$ and $B_{\kappa\sigma}(\mathbf{D})$ denote the open balls of radius $\kappa\sigma$ respectively centered on $\mathbf{x}$ and $\mathbf{D}$.

Built on [2.1.1] and given a small perturbation $\mathbf{E} \in R^{m \times p}$, it follows that

$$B(\mathbf{a}(\mathbf{D} + \mathbf{E}), \mathbf{w}) - B(\mathbf{a}(\mathbf{D}), \mathbf{w}) = \nabla_{\mathbf{z}} B_{\mathbf{w}}^T(\mathbf{a}(\mathbf{D} + \mathbf{E}) - \mathbf{a}(\mathbf{D})) + O(||\mathbf{E}||_F^2), \tag{2.17}$$

where the term $O(||\mathbf{E}||_F^2)$ is based on the fact that $\mathbf{a}(\mathbf{D}, \mathbf{x})$ is uniformly Lipschitz and $\mathcal{X} \times \mathcal{D}$ is compact. It is then possible to show that

$$B(\mathbf{a}(\mathbf{D} + \mathbf{E}), \mathbf{w}) - B(\mathbf{a}(\mathbf{D}), \mathbf{w}) = Tr(\mathbf{E}^T g(\mathbf{a}(\mathbf{D} + \mathbf{E}), \mathbf{w})) + O(||\mathbf{E}||_F^2), \tag{2.18}$$

where g has the form given in (2.11). This shows that the objective in (2.11) is differentiable on $\mathcal{D}$, and its gradient with respect to $\mathbf{D}$ is g.

2.2 Bi-Level Sparse Coding for Clustering

Many clustering methods highly depend on extracted features. In this section, we propose a joint optimization framework in terms of both feature extraction and discriminative clustering. We utilize graph regularized sparse codes as the features,

and formulate sparse coding as the constraint for clustering. Two cost functions are developed based on entropy-minimization and maximum-margin clustering principles, respectively. They are considered as the objectives to be minimized. Solving such a bi-level optimization mutually reinforces both sparse coding and clustering steps. Experiments on several benchmark datasets verify remarkable performance improvements led by the proposed joint optimization.

2.2.1 Introduction

Clustering aims to divide data into groups of similar objects (clusters), and plays an important role in many real world data mining applications. To learn the hidden patterns of the dataset in an unsupervised way, existing clustering algorithms can be described as either generative or discriminative in nature. Generative clustering algorithms model categories in terms of their geometric properties in feature spaces, or as statistical processes of data. Examples include K-means [59] and Gaussian mixture model (GMM) clustering [14], which assume a parametric form of the underlying category distributions. Rather than modeling categories explicitly, discriminative clustering techniques search for the boundaries or distinctions between categories. With fewer assumptions being made, these methods are powerful and flexible in practice. For example, maximum-margin clustering [174], [184], [183] aims to find the hyperplane that can separate the data from different classes with a maximum margin. Informatics theoretic clustering [101], [6] minimizes the conditional entropy of all samples. Many recent discriminative clustering methods have achieved very satisfactory performance [183].

Moreover, many clustering methods extract discriminative features from input data, prior to clustering. The Principal Component Analysis (PCA) feature is a common choice but not necessarily discriminative [185]. Kernel-based clustering methods [43] were explored to find implicit feature representations of input data. In [133], the features are selected for optimizing the discriminativity of the used partitioning algorithm, by solving a linear discriminant analysis (LDA) problem. More recently, sparse codes have been shown to be robust to noise and capable to handle high dimensional data [170]. Furthermore, ℓ_1-graph [36] builds the graph by reconstructing each data point sparsely and locally with other data. A spectral clustering [124] is followed based on the constructed graph matrix. In [147], [34], dictionary learning is combined with the clustering process, which makes

use of Lloyds-type algorithms that iteratively re-assign data to clusters and then optimize the dictionary associated with each cluster. In [185], the authors learned the sparse codes that explicitly consider the local data manifold structures. Their results indicate that encoding geometrical information will significantly enhance the learning performance. However, the clustering step in [185] is not correlated with the above mentioned discriminative clustering methods.

In this chapter, we propose to jointly optimize feature extraction and discriminative clustering, such that they mutually reinforce each other. We focus on sparse codes as the extracted features, and develop our loss functions based on two representative discriminative clustering methods, the entropy-minimization [101] and maximum-margin [174] clustering, respectively. A task-driven bi-level optimization model [113], [164] is then built upon the proposed framework. The sparse coding step is formulated as the lower-level constraint, where a graph regularization is enforced to preserve the local manifold structure [185]. The clustering-oriented cost functions are considered as the upper-level objectives to be minimized. Stochastic gradient descent algorithms are developed to solve both bi-level models. Experiments on several popular real datasets verify the noticeable performance improvement achieved by such a joint optimization framework.

2.2.2 Model Formulation

Sparse Coding with Graph Regularization

Sparse codes prove to be effective features for clustering. In [36], the authors suggested that the contribution of one sample to the reconstruction of another sample was a good indicator of similarity between these two samples. Therefore, the reconstruction coefficients (sparse codes) can be used to constitute the similarity graph for spectral clustering. ℓ_1-graph performs sparse representation for each data point separately without considering the geometric information and manifold structure of the entire data. Further research shows that the graph regularized sparse representations produce superior results in various clustering and classification tasks [185], [178]. In this chapter, we adopt the graph regularized sparse codes as the features for clustering.

We assume that all the data samples $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n], \mathbf{x}_i \in R^{m\times 1}, i = 1, 2, \cdots, n$, are encoded into their corresponding sparse codes $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n]$,

$\mathbf{a}_i \in R^{p\times 1}, i = 1, 2, \cdots, n$, using a learned dictionary $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_p]$, where $\mathbf{d}_i \in R^{m\times 1}, i = 1, 2, \cdots, p$ are the learned atoms. Moreover, given a pairwise similarity matrix $\mathbf{W}$, the sparse representations that capture the geometric structure of the data according to the manifold assumption should minimize the following objective: $\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{W}_{ij}||\mathbf{a}_i - \mathbf{a}_j||_2^2 = Tr(\mathbf{ALA^T})$, where $\mathbf{L}$ is the graph Laplacian matrix constructed from $\mathbf{W}$. In this chapter, $\mathbf{W}$ is chosen as the Gaussian Kernel: $\mathbf{W}_{ij} = \exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{\delta^2})$, where δ is the controlling parameter selected by cross-validation.

The graph regularized sparse codes are obtained by solving the following convex optimization:

$$\mathbf{A} = \arg\min_{\mathbf{A}} \tfrac{1}{2}||\mathbf{X} - \mathbf{DA}||_F^2 + \lambda\sum_i ||\mathbf{a}_i||_1 + \alpha Tr(\mathbf{ALA^T}) + \lambda_2||\mathbf{A}||_F^2. \tag{2.19}$$

Note that $\lambda_2 > 0$ is necessary for proving the differentiability of the objective function (please refer to Theorem 2.1.1 for the proof). However, setting $\lambda_2 = 0$ proves to work well in practice, and thus the term $\lambda_2||\mathbf{A}||_F^2$ will be omitted hereinafter.

Bi-level Optimization Formulation

The objective cost function for the joint framework can be expressed by the following bi-level optimization:

$$\begin{aligned} &\min_{\mathbf{D},\mathbf{w}} \quad C(\mathbf{A}, \mathbf{w}) \\ &s.t. \quad \mathbf{A} = \arg\min_{\mathbf{A}} \tfrac{1}{2}||\mathbf{X} - \mathbf{DA}||_F^2 + \lambda\sum_i ||\mathbf{a}_i||_1 + \alpha Tr(\mathbf{ALA^T}), \end{aligned} \tag{2.20}$$

where $C(\mathbf{A}, \mathbf{w})$ is a cost function evaluating the loss of clustering. It can be formulated differently based on various clustering principles, two of which will be discussed and solved in Section 2.2.3.

2.2.3 Clustering-Oriented Cost Functions

Assuming K clusters, and $\mathbf{w} = [\mathbf{w}_1, ..., \mathbf{w}_K]$ as the set of parameters of the loss function, where $\mathbf{w}_i$ corresponds to the i-th cluster, $i = 1, 2; ..., K$. We introduce two forms of loss functions, each of which is derived from a representative discriminative clustering method.

Entropy-Minimization Loss

Maximization of the mutual information with respect to parameters of the encoder model effectively defines a discriminative unsupervised optimization framework. The model is parameterized similarly to a conditionally trained classifier, but the cluster allocations are unknown [6]. In [45], [101], the authors adopted an information-theoretic framework as an implementation of the low-density separation assumption by minimizing the conditional entropy. By substituting the logistic posterior probability into the minimum conditional entropy principle, the authors got the logistics clustering algorithm, which is equivalent to find a labelling strategy so that the total entropy of data clustering is minimized.

Since the true cluster label of each $\mathbf{x_i}$ is unknown, we introduce the predicted confidence probability p_{ij} that sample $\mathbf{x_i}$ belongs to cluster $j, i = 1, 2, \cdots, N, j = 1, 2, \cdots, K$, which is set as the likelihood of the multinomial logistic (softmax) regression:

$$p_{ij} = p(j|\mathbf{w}, \mathbf{a}_i) = \frac{1}{1+e^{-j\mathbf{w}^T\mathbf{a}_i}}, \tag{2.21}$$

The loss function for all data could be defined accordingly in an entropy-like form:

$$C(\mathbf{A}, \mathbf{w}) = -\sum_{i=1}^{n}\sum_{j=1}^{K} p_{ij}\log p_{ij}. \tag{2.22}$$

The predicted cluster label of $\mathbf{a}_i$ is the cluster j where it achieves the largest likelihood probability p_{ij}. The logistics regression can deal with multi-class problems more easily compared with the support vector machine (SVM). The next important thing we need to study is the differentiability of (2.20).

Theorem 2.2.1. *The objective $C(\mathbf{A}, \mathbf{w})$ defined in (2.22) is differentiable on $\mathbf{D} \times \mathbf{w}$.*

Proof: Denote $\mathbf{X} \in \mathcal{X}$, and $\mathbf{D} \in \mathcal{D}$. Also let the objective function $C(\mathbf{A}, \mathbf{w})$ in (2.22) be denoted as C for short. The differentiability of C with respect to $\mathbf{w}$ is easy to show, using only the compactness of $\mathcal{X}$, as well as the fact that C is twice differentiable.

We will therefore focus on showing that C is differentiable with respect to $\mathbf{D}$, which is more difficult since $\mathbf{A}$, and thus $\mathbf{a}_i$, is not differentiable everywhere. Without loss of generality, we use a vector $\mathbf{a}$ instead of $\mathbf{A}$ for simplifying the derivations hereinafter. In some cases, we may equivalently express $\mathbf{a}$ as $\mathbf{a}(\mathbf{D}, \mathbf{w})$ in order to emphasize the functional dependence. Based on Theorem 2.1.1, and

given a small perturbation $\mathbf{E} \in R^{m \times p}$, it follows that

$$C(\mathbf{a}(\mathbf{D}+\mathbf{E}),\mathbf{w}) - C(\mathbf{a}(\mathbf{D}),\mathbf{w}) = \nabla_{\mathbf{z}} C_{\mathbf{w}}^T(\mathbf{a}(\mathbf{D}+\mathbf{E}) - \mathbf{a}(\mathbf{D})) + O(||\mathbf{E}||_F^2) \tag{2.23}$$

where the term $O(||\mathbf{E}||_F^2)$ is based on the fact that $\mathbf{a}(\mathbf{D},\mathbf{x})$ is uniformly Lipschitz and $\mathcal{X} \times \mathcal{D}$ is compact. It is then possible to show that

$$C(\mathbf{a}(\mathbf{D}+\mathbf{E}),\mathbf{w}) - C(\mathbf{a}(\mathbf{D}),\mathbf{w}) = Tr(\mathbf{E}^T g(\mathbf{a}(\mathbf{D}+\mathbf{E}),\mathbf{w})) + O(||\mathbf{E}||_F^2) \tag{2.24}$$

where g has the form given in Algorithm I. C is thus differentiable on $\mathcal{D}$.

Built on the differentiability proof, we are able to solve (3.13) using a projected first order stochastic gradient descent (SGD) algorithm, whose detailed steps are outlined in Algorithm 2. At a high level overview, it consists of an outer stochastic gradient descent loop that incrementally samples the training data. It uses each sample to approximate gradients with respect to the classifier parameter $\mathbf{w}$ and the dictionary $\mathbf{D}$, which are then used to update them.

Algorithm 2 Stochastic gradient descent algorithm for solving (2.20), with $C(\mathbf{A},\mathbf{w})$ as defined in (2.22)

Require: $\mathbf{X}, \sigma$; λ; $\mathbf{D}_0$ and $\mathbf{w}_0$ (initial dictionary and classifier parameter); ITER (number of iterations); t_0, ρ (learning rate)

1: Construct the matrix $\mathbf{L}$ from $\mathbf{X}$ and σ.
2: FOR t=1 to ITER DO
3: Draw a subset $(\mathbf{X}_t, \mathbf{Y}_t)$ from $(\mathbf{X}, \mathbf{Y})$
4: Graph-regularized sparse coding: computer $\mathbf{A}^*$:
$\mathbf{A}^* = \arg\min_{\mathbf{A}} \frac{1}{2}||\mathbf{X} - \mathbf{D}\mathbf{A}||_F^2 + \lambda \sum_i ||\mathbf{a}_i||_1 + Tr(\mathbf{A}\mathbf{L}\mathbf{A}^{\mathbf{T}})$.
5: Compute the active set S (the nonzero support of $\mathbf{A}^*$)
6: Compute $\boldsymbol{\beta}^*$: Set $\boldsymbol{\beta}^*_{S^C} = 0$ and $\boldsymbol{\beta}^*_S = (\mathbf{D}_S^T\mathbf{D}_S + \lambda_2\mathbf{I})^{-1}\nabla_{\mathbf{A_S}}[C(\mathbf{A},\mathbf{w})]$
7: Choose the learning rate $\rho_t = \min(\rho, \rho\frac{t_0}{t})$
8: Update $\mathbf{D}$ and $\mathbf{W}$ by a projected gradient step:
$\mathbf{w} = \prod_{\mathbf{w}}[\mathbf{w} - \rho_t \nabla_{\mathbf{w}} C(\mathbf{A},\mathbf{w})]$
$\mathbf{D} = \prod_{\mathbf{D}}[\mathbf{D} - \rho_t(\nabla_{\mathbf{D}}(-\mathbf{D}\boldsymbol{\beta}^*\mathbf{A}^T + (\mathbf{X}_t - \mathbf{D}\mathbf{A})\boldsymbol{\beta}^{*T})]$
where $\prod_{\mathbf{w}}$ and $\prod_{\mathbf{D}}$ are respectively orthogonal projections on the embedding spaces of $\mathbf{w}$ and $\mathbf{D}$.
9: END FOR

Ensure: $\mathbf{D}$ and $\mathbf{w}$

Maximum-Margin Loss

Xu et al. [174] proposed maximum margin clustering (MMC), which borrows the idea from the SVM theory. Their experimental results showed that the MMC technique could often obtain more accurate results than conventional clustering methods. Technically, what MMC does is just to find a way to label the samples by running an SVM implicitly, and the SVM margin obtained would be maximized over all possible labelings [183]. However, unlike supervised large margin methods which are usually formulated as convex optimization problems, maximum margin clustering is a non-convex integer optimization problem, which is much more difficult to solve. [102] made several relaxations to the original MMC problem and reformulated it as a semi-definite programming (SDP) problem. The cutting plane maximum margin clustering (CPMMC) algorithm was presented in [183] to solve MMC with much improved efficiency.

To develop the multi-class max-margin loss of clustering, we refer to the classical multi-class SVM formulation in [41]. Given the sparse code $\mathbf{a}_i$ are the features to be clustered, we define the multi-class model as

$$f(\mathbf{a}_i) = \underset{j=1,...,K}{\arg\max} f^j(\mathbf{a}_i) = \underset{j=1,...,K}{\arg\max} (\mathbf{w}_j^T \mathbf{a}_i) \tag{2.25}$$

where f^j is the prototype for the j-th cluster and $\mathbf{w_j}$ is its corresponding weight vector. The predicted cluster label of $\mathbf{a}_i$ is the cluster of the weight vector that achieves the maximum value $\mathbf{w_j^T a_i}$. Let $\mathbf{w} = [\mathbf{w_1}, ..., \mathbf{w_K}]$, the multi-class max-margin loss for $\mathbf{a}_i$ could be defined as:

$$\begin{aligned} &C(\mathbf{a}_i, \mathbf{w}) = \max(0, 1 + f^{r_i}(\mathbf{a}_i) - f^{y_i}(\mathbf{a}_i)) \\ &\text{where} \quad y_i = \underset{j=1,...,K}{\arg\max} f^j(\mathbf{a}_i), r_i = \underset{j=1,...,K, j \neq y_i}{\arg\max} f^j(\mathbf{a}_i) \end{aligned} \tag{2.26}$$

Note that unlike training a multi-class SVM classier, where y_i is given as a training label, the clustering scenario requires us to jointly estimate y_i as a variable. The overall max-margin loss to be minimized is (λ as the coefficient):

$$C(\mathbf{A}, \mathbf{w}) = \tfrac{\lambda}{2}||\mathbf{w}||^2 + \textstyle\sum_{i=1}^{n} C(\mathbf{a}_i, \mathbf{w}) \tag{2.27}$$

But to solve (2.26) or (2.27) with respect to the same framework as logistic loss will involve two additional concerns, which need to be handled specifically.

First, the hinge loss of the form (2.26) is not differentiable, with only a sub-

gradient existing. That makes the objective function $C(\mathbf{A}, \mathbf{w})$ indifferentiable on $\mathbf{D} \times \mathbf{w}$, and further the analysis in the Theorem [3.4.1] proof cannot be applied. We could have used the squared hinge loss or modified Huber loss for a quadratically smoothed loss function [95]. However, as we checked in the experiments, the quadratically smoothed loss is not as good as hinge loss in training time and sparsity. Also, though not theoretically guaranteed, using the subgradient of $\mathrm{C}(\mathbf{A}, \mathbf{w})$ works well in our case.

Second, given that $\mathbf{w}$ is fixed, it should be noted that y_i and r_i are both functions of $\mathbf{a}_i$. Therefore, calculating the derivative of (2.26) over $\mathbf{a}_i$ would involve expanding both r_i and y_i, and become quite complicated. Instead, we borrow ideas from the regularity of the elastic net solution [113], that the set of non-zero coefficients of the elastic net solution should not change for small perturbations. Similarly, due to the continuity of the objective, it is assumed that a sufficiently small perturbation over the current $\mathbf{a}_i$ will not change y_i and r_i. Therefore in each iteration, we could directly pre-calculate y_i and r_i using the current $\mathbf{w}$ and $\mathbf{a}_i$ and fix them for $\mathbf{a}_i$ updates[1].

Given the above two handlings, for a single sample $\mathbf{a}_i$, if the hinge loss is over 0, the derivative of (2.26) over $\mathbf{w}$ is:

$$\Delta_i^j = \begin{cases} \lambda \mathbf{w}_i^j - \mathbf{a}_i & \text{if} \quad j = y_i \\ \lambda \mathbf{w}_i^j + \mathbf{a}_i & \text{if} \quad j = r_i \\ \lambda \mathbf{w}_i^j & \text{otherwise,} \end{cases} \tag{2.28}$$

where Δ_i^j denotes the j-th element of the derivative for the sample $\mathbf{a}_i$. If the hinge loss is less than 0, then $\Delta_i^j = \lambda \mathbf{w}_i^j$. The derivative of (2.26) over $\mathbf{a}_i$ is $\mathbf{w}^{r_i} - \mathbf{w}^{y_i}$ if the hinge loss is over 0, and 0 otherwise. Note the above deduction can be conducted in a batch mode. It is then similarly solved using a projected SGD algorithm, whose steps are outlined in Algorithm 3.

[1]To avoid ambiguity, if y_i and r_i are the same, i.e., the max value is reached by two cluster prototypes simultaneously in current iteration, then we ignore the gradient update of $\mathbf{a}_i$.

Algorithm 3 Stochastic gradient descent algorithm for solving (2.20), with $C(\mathbf{A}, \mathbf{w})$ as defined in (2.27)

Require: $\mathbf{X}, \sigma$; λ; $\mathbf{D}_0$ and $\mathbf{w}_0$ (initial dictionary and classifier parameter); ITER (number of iterations); t_0, ρ (learning rate)

1: Construct the matrix $\mathbf{L}$ from $\mathbf{X}$ and σ.
2: Estimate the initialization of y_i and r_i by pre-clustering, $i = 1, 2, ..., N$
3: FOR t=1 to ITER DO
4: Conduct the same step 4-7 in Algorithm 2.
5: Update $\mathbf{D}$ and $\mathbf{W}$ by a projected gradient step, based on the derivatives of (2.27) over $\mathbf{a}_i$ and $\mathbf{w}$ (2.28).
6: Update y_i and r_i using the current $\mathbf{w}$ and $\mathbf{a_i}$, $i = 1, 2, ..., N$.
7: END FOR

Ensure: $\mathbf{D}$ and $\mathbf{w}$

2.2.4 Experiments

Datasets

We conduct our clustering experiments on four popular real datasets, which are summarized in Table 2.6. The ORL face database contains 400 facial images for 40 subjects, and each subject has 10 images of size 32 $\times$ 32. The images are taken at different times with varying lighting and facial expressions. The subjects are all in an upright, frontal position with a dark homogeneous background. The MNIST handwritten digit database consists of a total number of 70, 000 images, with digits ranging from 0 to 9. The digits are normalized and centered in fixed-size images of 28 $\times$ 28. The COIL20 image library contains 1,440 images of size 32 $\times$ 32, for 20 objects. Each object has 72 images, which were taken 5 degree apart as the object was rotated on a turntable. The CMU-PIE face database contains 68 subjects with 41,368 face images as a whole. For each subject, we have 21 images of size 32 $\times$ 32, under different lighting conditions.

Table 2.6: Comparison of all datasets

Name	Number of Images	Class	Dimension
ORL	400	10	1,024
MNIST	70,000	10	784
COIL20	1,440	20	1,024
CMU-PIE	41,368	68	1,024

Table 2.7: Accuracy and NMI performance comparisons on all datasets

		KM	KM + SC	EMC	EMC + SC	MMC	MMC + SC	joint EMC	joint MMC
ORL	Acc	0.5250	0.5887	0.6011	0.6404	0.6460	0.6968	0.7250	**0.7458**
	NMI	0.7182	0.7396	0.7502	0.7795	0.8050	0.8043	0.8125	**0.8728**
MNIST	Acc	0.6248	0.6407	0.6377	0.6493	0.6468	0.6581	0.6550	**0.6784**
	NMI	0.5142	0.5397	0.5274	0.5671	0.5934	0.6161	0.6150	**0.6451**
COIL20	Acc	0.6280	0.7880	0.7399	0.7633	0.8075	0.8493	0.8225	**0.8658**
	NMI	0.7621	0.9010	0.8621	0.8887	0.8922	0.8977	0.8850	**0.9127**
CMU-PIE	Acc	0.3176	0.8457	0.7627	0.7836	0.8482	0.8491	0.8250	**0.8783**
	NMI	0.6383	0.9557	0.8043	0.8410	0.9237	0.9489	0.9020	**0.9675**

Evaluation Metrics

We apply two widely used measures to evaluate the performance of the clustering methods: the accuracy and the Normalized Mutual Information (NMI) [185], [36]. Suppose the predicted label of the $\mathbf{x}_i$ is $\hat{y}_i$ which is produced by the clustering method, and y_i is the ground truth label. The accuracy is defined as:

$$Acc = \frac{\mathbb{1}_{\Phi(\hat{y}_i)\neq y_i}}{n}, \tag{2.29}$$

where $\mathbb{1}$ is the indicator function, and Φ is the best permutation mapping function [108]. On the other hand, suppose the clusters obtained from the predicted labels $\{\hat{y}_i\}_{i=1}^n$ and $\{y_i\}_{i=1}^n$ as $\hat{C}$ and C, respectively. The mutual information between $\hat{C}$ and C is defined as:

$$MI(\hat{C}, C) = \sum_{\hat{c}\in\hat{C}, c\in C} p(\hat{c}, c) \log \frac{p(\hat{c},c)}{p(\hat{c})p(c)}, \tag{2.30}$$

where $p(\hat{c})$ and $p(c)$ are the probabilities that a data point belongs to the clusters $\hat{C}$ and C, respectively, and $p(\hat{c}, c)$ is the probability that a data point jointly belongs to $\hat{C}$ and C. The NMI is defined as:

$$NMI(\hat{C}, C) = \frac{MI(\hat{C},C)}{\max\{H(\hat{C}),H(C)\}}, \tag{2.31}$$

where $H(\hat{C})$ and $H(C)$ are the entropies of $\hat{C}$ and C, respectively. NMI takes values between [0,1].

Comparison Experiments

Comparison Methods We compare the following eight methods on all four datasets:

- **KM:** K-Means clustering on the input data.
- **KM + SC:** A dictionary $\mathbf{D}$ is first learned from the input data by K-SVD [60]. Then KM is performed on the graph-regularized sparse code features (3.13) over $\mathbf{D}$.
- **EMC:** Entropy-minimization clustering, by minimizing (2.22) on the input data.
- **EMC + SC:** EMC performed on the graph-regularized sparse codes over the pre-learned K-SVD dictionary $\mathbf{D}$.
- **MMC:** Maximum-margin clustering [183].
- **MMC + SC:** MMC performed on the graph-regularized sparse codes over the pre-learned K-SVD dictionary $\mathbf{D}$.
- **Joint EMC:** The proposed joint optimization (2.20), with $C(\mathbf{A}, \mathbf{w})$ as defined in (2.22).
- **Joint MMC:** The proposed joint optimization (2.20), with $C(\mathbf{A}, \mathbf{w})$ as defined in (2.27).

All images are first reshaped into vectors, and PCA is then applied to reduce the data dimensionality by keeping 98% information, which is also used in [185] to improving efficiency. The multi-class MMC algorithm is implemented based on the publicly available CPMMC code for two-class clustering [183], following the multi-class case descriptions in the original paper. For all algorithms that involve graph-regularized sparse coding, the graph regularization parameter α is fixed to be 1, and the dictionary size p is 128 by default. For joint EMC and joint MMC, we set ITER as 30, ρ as 0.9, and t_0 as 5. Other parameters in competing methods are tuned in cross-validation experiments to our best efforts.

Comparison Analysis All the comparison results (accuracy and NMI) are listed in Table. 2.7, from which we conclude the following:

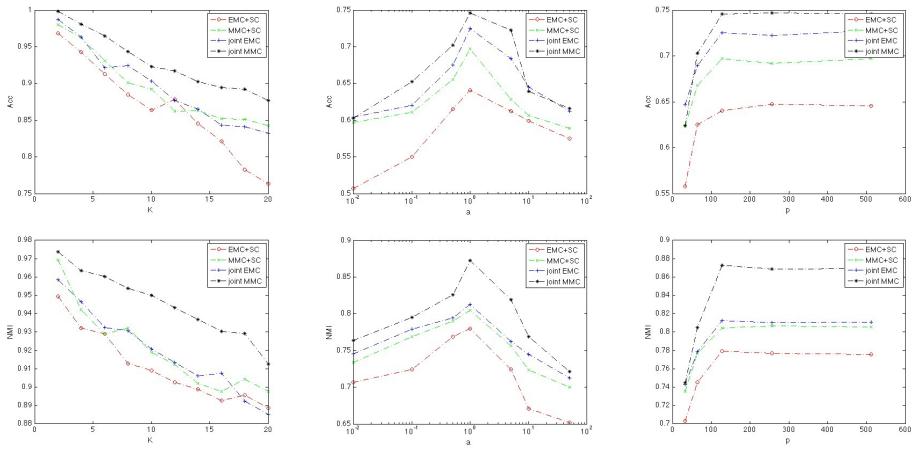


Figure 2.8: (a) The clustering accuracy and NMI measurements versus the number of clusters K; (b) The clustering accuracy and NMI measurements versus the parameter choices of α; (c) The clustering accuracy and NMI measurements versus the parameter choices of p.

- **1:** The joint EMC and joint MMC methods each outperform their "non-joint" counterparts, e.g., EMC + SC and MMC + SC, respectively. For example, on the *ORL* dataset, joint MMC surpasses MMC + SC by around 5% in accuracy and 7% in NMI. Those demonstrate that the key contribution of this chapter, i.e., jointly optimizing the sparse coding and clustering steps, indeed leads to improved performance.

- **2:** KM + SC, EMC + SC, and MMC + SC all outperform their counterparts using raw input data, which verifies that sparse codes are effective features that help improve the clustering discriminability.

- **3:** The joint MMC obtains the best performance in all cases, outperforming the others, including joint EMC, with significant margins. The MMC + SC obtains the second best performance for the last three datasets (for ORL, it is joint EMC that ranks the second). The above facts reveal the power of the max-margin loss (2.27).

Varying the number of clusters On the COIL20 dataset, We re-conduct the clustering experiments with the cluster number K ranging from 2 to 20, using EMC + SC, MMC + SC, joint EMC, and joint MMC. For each K except for 20, 10 test runs are conducted on different randomly chosen clusters, and the final scores are obtained by averaging over the 10 tests. Figure 2.8 (a) shows the clustering

accuracy and NMI measurements versus the number of clusters. It is revealed that the two joint methods consistently outperform their non-joint counterparts. When K goes up, the performances of joint methods seem to degrade less slowly.

Initialization and Parameters As a typical case in machine learning, we use SGD in a setting where it is not guaranteed to converge in theory, but behaves well in practice. As observed in our experiments, a good initialization of $\mathbf{D}$ and $\mathbf{w}$ can affect the final results notably. We initialize Joint EMC by the $\mathbf{D}$ and $\mathbf{w}$ solved from EMC + SC, and Joint MMC by the solutions from MMC + SC, respectively.

There are two parameters that we set empirically ahead: the graph regularization parameter α, and the dictionary size p. The regularization term imposes stronger smoothness constraints on the sparse codes when α grows larger. Also, while a compact dictionary is more desirable computationally, more redundant dictionaries may lead to less cluttered features that can be better discriminated. We investigate how the clustering performances EMC + SC, MMC + SC, joint EMC, and joint MMC change on the ORL dataset, with various α and p values. As depicted in Figure 2.8 (b) and (c), we observe that:

- **1:** While α goes up, the accuracy result will first go up then down (the peak is around α =1). That could be interpreted as when α is too small, the local manifold information is not sufficiently encoded. On the other hand, when α turns overly large, the sparse codes are "over-smoothened" with a reduced discriminability.

- **2:** Increasing dictionary size p will first improve the accuracy sharply, which however soon reaches a plateau. Thus in practice, we keep a medium dictionary size p =128 for all experiments.

2.2.5 Conclusion

We propose a joint framework to optimize sparse coding and discriminative clustering simultaneously. We adopt graph regularized sparse codes as the feature to be learned, and design two clustering-oriented cost functions, by entropy-minimization and maximum-margin principles, respectively. The formulation of a task-driven bi-level optimization mutually reinforces both sparse coding and clustering steps. Experiments on several benchmark datasets verify the remarkable performance improvement led by the proposed joint optimization.

CHAPTER 3

DEEP MODELS MADE INTERPRETABLE: A SPARSE CODING PERSPECTIVE AND BEYOND

3.1 Background and Related Work

Albeit effective, conventional sparse coding models rely on iterative approximation algorithms, whose inherently sequential structure, as well as the data-dependent latency, often constitute a major bottleneck in the computational efficiency. Besides, the joint optimization of the (unsupervised) feature learning and the supervised steps often has to rely on solving complex bi-level optimization [165], and thus constitutes another efficiency bottleneck. Further, to effectively represent datasets of growing sizes, sparse coding has to refer to larger dictionaries. Since the inference complexity of sparse coding increases more than linearly with respect to the dictionary size [165], its scalability turns out to be limited. Other "shallow" models suffer from similar problems.

Deep learning has recently attracted great attention [89]. The advantages of deep learning lie in its composition of multiple non-linear transformations to yield more abstract and descriptive representations. The feed-forward networks could be tuned jointly with task-driven loss functions [163]. With the aid of gradient descent, it also scales linearly in time and space with the number of train samples. There has been a booming interest in bridging "shallow" optimization and deep learning models. Our work shares in the spirit of prior wisdom, with many more novel models explored and insights gained. In this chapter, we derive deep models from the ℓ_0 sparse approximation model, the graph-regularized ℓ_1 sparse approximation model, the ℓ_∞ constrained model, and the dual-sparsity model. The structural priors inferred from all those models act as effective network regularizations, and lead to improved generalization ability. The resulting deep models also enjoy faster inference, larger learning capacity, and better scalability, compared to their "shallow" counterparts.

ℓ_0 and ℓ_1-based Sparse Approximations

Finding the sparsest, or minimum ℓ_0-norm, representation of a signal given a dictionary of basis atoms is an important problem in many application domains. Consider a data sample $\mathbf{x} \in R^{m\times 1}$, that is encoded into its sparse code $\mathbf{a} \in R^{p\times 1}$ using a learned dictionary $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_p]$, where $\mathbf{d}_i \in R^{m\times 1}, i = 1, 2, \cdots, p$ are the learned atoms. The sparse codes are obtained by solving the ℓ_0 regularized problem (λ is a constant):

$$\mathbf{a} = \arg\min_{\mathbf{a}} \tfrac{1}{2}||\mathbf{x} - \mathbf{D}\mathbf{a}||_F^2 + \lambda||\mathbf{a}||_0. \tag{3.1}$$

Alternatively, one could explicitly impose constraints on the number of non-zero coefficients of the solution, by solving the M-sparse problem:

$$\mathbf{a} = \arg\min_{\mathbf{a}} ||\mathbf{x} - \mathbf{D}\mathbf{a}||_F^2 \qquad s.t. \quad ||\mathbf{a}||_0 \leq M. \tag{3.2}$$

Unfortunately, these optimization problems are often intractable because there is a combinatorial increase in the number of local minima as the number of the candidate basis vectors increases. One potential remedy is to employ a convex surrogate measure, such as the ℓ_1-norm, in place of the ℓ_0-norm that leads to a more tractable optimization problem. For example, (3.1) could be relaxed as:

$$\mathbf{a} = \arg\min_{\mathbf{a}} \tfrac{1}{2}||\mathbf{x} - \mathbf{D}\mathbf{a}||_F^2 + \lambda||\mathbf{a}||_1. \tag{3.3}$$

It creates a unimodal optimization problem that can be solved via linear programming techniques. The downside is that we have now introduced a mismatch between the ultimate goal and the objective function [169]. Under certain conditions, the minimum ℓ_1-norm solution equals to the minimum ℓ_0-norm one [56]. But in practice, the ℓ_1 approximation is often used way beyond these conditions, and is thus quite heuristic. As a result, we often get a solution which is not exactly minimizing the original ℓ_0-norm.

That said, ℓ_1 approximation is found to work practically well for many sparse coding problems. Yet in certain applications, we intend to control the exact number of nonzero elements, such as basis selection [169], where ℓ_0 approximation is indispensable. Beyond that, ℓ_0-approximation is desirable for performance concerns in many ways. In compressive sensing literature, empirical evidence [26] suggested that using an iterative reweighted ℓ_1 scheme to approximate the ℓ_0 solu-

tion often improved the quality of signal recovery. In image enhancement, it was shown in [180] that ℓ_0 data fidelity was more suitable for reconstructing images corrupted with impulse noise. For the purpose of image smoothening, the authors of [173] utilized ℓ_0 gradient minimization to globally control how many non-zero gradients to approximate prominent structures in a structure-sparsity-management manner. Recent work [159] revealed that ℓ_0 sparse subspace clustering can completely characterize the set of minimal union-of-subspace structure, without additional separation conditions required by its ℓ_1 counterpart.

Network Implementation of ℓ_1-Approximation

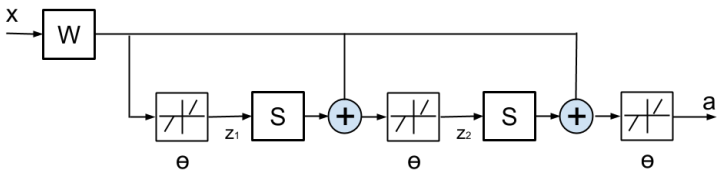


Figure 3.1: A LISTA network [68] with two time-unfolded stages.

In [68], a feed-forward neural network, as illustrated in Figure 3.1, was proposed to efficiently approximate the ℓ_1-based sparse code $\mathbf{a}$ of the input signal $\mathbf{x}$; the sparse code a is obtained by solving (3.3) for a given dictionary $\mathbf{D}$ in advance. The network has a finite number of stages, each of which updates the intermediate sparse code $\mathbf{z}^k$ (k = 1, 2) according to

$$\mathbf{z}^{k+1} = s_\theta(\mathbf{W}\mathbf{x} + \mathbf{S}\mathbf{z}^k), \tag{3.4}$$

where s_θ is an element-wise shrinkage function ($\mathbf{u}$ is a vector and $\mathbf{u}_i$ is its i-th element, $i = 1, 2, ..., p$):

$$[s_\theta(\mathbf{u})]_i = \text{sign}(\mathbf{u}_i)(|\mathbf{u}_i| - \theta_i)_+. \tag{3.5}$$

The parameterized encoder, named learned ISTA (LISTA), is a natural network implementation of the iterative shrinkage and thresholding algorithm (ISTA). LISTA learned all its parameters $\mathbf{W}$, $\mathbf{S}$ and θ from training data using a back-propagation algorithm [93]. In this way, a good approximation of the underlying sparse code can be obtained after a fixed small number of stages.

In [146], the authors leveraged a similar idea on fast trainable regressors and constructed feed-forward network approximations of the learned sparse models. Such a process-centric view was later extended in [145] to develop a principled process of learned deterministic fixed-complexity pursuits, in lieu of iterative proximal gradient descent algorithms, for structured sparse and robust low rank models. Recently, [75] summarized the methodology of the problem-level and model-based "deep unfolding", and developed new architectures as inference algorithms for both Markov random fields and non-negative matrix factorization.

3.2 Deep ℓ_0 Encoders for Classification and Clustering

Despite its nonconvex nature, ℓ_0 sparse approximation is desirable in many theoretical and application cases. We study the ℓ_0 sparse approximation problem with the tool of deep learning, by proposing Deep ℓ_0 Encoders. Two typical forms, the ℓ_0 regularized problem and the M-sparse problem, are investigated. Based on solid iterative algorithms, we model them as feed-forward neural networks, through introducing novel neurons and pooling functions. Enforcing such structural priors acts as an effective network regularization. The deep encoders also enjoy faster inference, larger learning capacity, and better scalability compared to conventional sparse coding solutions. Furthermore, under task-driven losses, the models can be conveniently optimized from end to end. Numerical results demonstrate the impressive performances of the proposed encoders.

3.2.1 Introduction

Sparse signal approximation has gained popularity over the last decade. The sparse approximation model suggests that a natural signal could be compactly approximated, by only a few atoms out of a properly given dictionary, where the weights associated with the dictionary atoms are called the sparse codes. Proven to be both robust to noise and scalable to high dimensional data, sparse codes are known as powerful features, and benefit a wide range of signal processing applications, such as source coding [57], denoising [55], source separation [47], pattern classification [170], and clustering [36].

We are particularly interested in the ℓ_0-based sparse approximation problem, which is the fundamental formulation of sparse coding [56]. The nonconvex ℓ_0

problem is intractable and often instead attacked by minimizing surrogate measures, such as the ℓ_1-norm, which leads to more tractable computational methods. However, it has been both theoretically and practically discovered that solving ℓ_0 sparse approximation is still preferable in many cases.

In this section, we investigate two typical forms of ℓ_0-based sparse approximation problems: the ℓ_0 regularized problem, and the M-sparse problem. Based on solid iterative algorithms [17], we formulate them as feed-forward neural networks [68], called **Deep** ℓ_0 **Encoders**, through introducing novel neurons and pooling functions. We study their applications in image classification and clustering; in both cases the models are optimized in a task-driven, end-to-end manner. Impressive performances are observed in numerical experiments.

3.2.2 Deep ℓ_0 Encoders

Deep ℓ_0-Regularized Encoder

To solve the optimization problem in (3.1), an iterative hard-thresholding (IHT) algorithm was derived in [17]:

$$\mathbf{a}^{k+1} = h_{\lambda^{0.5}}(\mathbf{a}^k + \mathbf{D}^{\mathbf{T}}(\mathbf{x} - \mathbf{D}\mathbf{a}^k)), \tag{3.6}$$

where $\mathbf{a}^k$ denotes the intermediate result of the k-th iteration, and h_θ is an element-wise **hard thresholding** operator:

$$[h_{\lambda^{0.5}}(\mathbf{u})]_i = \begin{cases} 0 & \text{if} \quad |\mathbf{u}_i| < \lambda^{0.5} \\ \mathbf{u}_i & \text{if} \quad |\mathbf{u}_i| \geq \lambda^{0.5}. \end{cases} \tag{3.7}$$

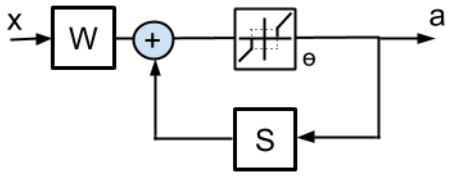


Figure 3.2: The block diagram of solving (3.6).

.

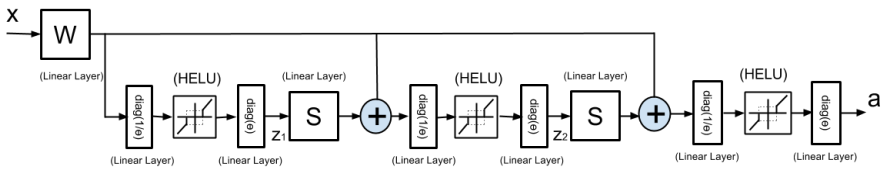


Figure 3.3: Deep ℓ_0-Regularized Encoder, with two time-unfolded stages.

Eqn. (3.6) could be alternatively rewritten as:

$$\begin{aligned} &\mathbf{a}^{k+1} = h_\theta(\mathbf{W}\mathbf{x} + \mathbf{S}\mathbf{a}^k), \\ &\mathbf{W} = \mathbf{D}^T, \mathbf{S} = \mathbf{I} - \mathbf{D}^T\mathbf{D}, \theta = \lambda^{0.5}, \end{aligned} \tag{3.8}$$

and expressed as the block diagram in Figure 3.2, which outlines a recurrent network form of solving (3.6).

By time-unfolding and truncating Figure 3.2 to a fixed number of K iterations (K = 2 in this section by default), we obtain a feed-forward network structure in Figure 3.3, where $\mathbf{W}$, $\mathbf{S}$ and θ are shared among both stages, named **Deep ℓ_0-Regularized Encoder**, Furthermore, $\mathbf{W}$, $\mathbf{S}$ and θ are all to be learnt, instead of being directly constructed from any pre-computed $\mathbf{D}$. Although the equations in (3.8) do not directly apply any more to solving the Deep ℓ_0-Regularized Encoder, they can usually serve as a high-quality initialization of the latter.

Note that the activation thresholds θ are less straightforward to update. We rewrite (3.35) as: $[h_\theta(\mathbf{u})]_i = \theta_i h_1(\mathbf{u}_i/\theta_i)$. It indicates that the original neuron with trainable thresholds can be decomposed into two linear scaling layers, plus a unit-hard-threshold neuron, the latter of which we call Hard thrEsholding Linear Unit (**HELU**). The weights of the two scaling layers are diagonal matrices defined by θ and its element-wise reciprocal, respectively.

Discussion on HELU While being inspired by LISTA, the differentiating point of Deep ℓ_0-Regularized Encoder lies in the HELU neuron. Compared to classical neuron functions such as logistic, sigmoid, and ReLU [118], as well as the soft shrinkage and thresholding operation (3.35) in LISTA, HELU does not penalize large values, yet enforces strong (in theory infinite) penalty over small values. As such, HELU tends to produce highly sparse solutions.

The neuron form of LISTA (3.35) could be viewed as a double-sided and translated variant of ReLU, which is continuous and piecewise linear. In contrast, HELU is a **discontinuous** function that rarely occurs in existing deep network neurons. As pointed out by [76], HELU has countably many discontinuities and

is thus (Borel) measurable, in which case the universal approximation capability of the network is not compromised. However, experiments remind us that the algorithmic learnability with such discontinuous neurons (using popular first-order methods) is in question, and the training is in general hard. For computation concerns, we replace HELU with the following continuous and piecewise linear function HELU_σ, during network training:

$$[\text{HELU}_\sigma(\mathbf{u})]_i = \begin{cases} 0 & \text{if} \quad |\mathbf{u}_i| \leq 1-\sigma \\ \frac{(\mathbf{u}_i - 1 + \sigma)}{\sigma} & \text{if} \quad 1-\sigma < \mathbf{u}_i < 1 \\ \frac{(\mathbf{u}_i + 1 - \sigma)}{\sigma} & \text{if} -1 < \mathbf{u}_i < \sigma - 1 \\ \mathbf{u}_i & \text{if} \quad |\mathbf{u}_i| \geq 1. \end{cases} \tag{3.9}$$

Obviously, HELU_σ becomes HELU when $\sigma \to 0$. To approximate HELU, we tend to choose very small σ, while avoiding putting the training ill-posed. In practice, we start with a moderate σ (0.2 by default), and divide it by 10 after each epoch. After several epoches, HELU_σ becomes very close to the ideal HELU.

In [136], the authors introduced an ideal hard thresholding function for solving sparse coding, whose formulation was close to HELU. Note that [136] approximates the ideal function with a sigmoid function, which has connections with our HELU_σ approximation. In [87], a similar truncated linear ReLU was adopted.

Deep M-Sparse ℓ_0 Encoder

Both the ℓ_0 regularized problem in (3.1) and Deep ℓ_0-Regularized Encoder have no explicit control on the sparsity level of the solution. We therefore look at the M-sparse problem in (3.2), and derive the following iterative algorithm [17]:

$$\mathbf{a}^{k+1} = h_M(\mathbf{a}^k + \mathbf{D}^\mathbf{T}(\mathbf{x} - \mathbf{D}\mathbf{a}^k)). \tag{3.10}$$

Eqn. (3.10) resembles (3.6), except that h_M is now a non-linear operator retaining the M coefficients with the **top M-largest absolute values**. Following the same methodology as in the previous section, the iterative form could be unfolded and truncated to the **Deep M-sparse Encoder**. To deal with the h_M operation, we refer to the popular concepts of pooling and unpooling [181] in deep networks, and introduce the pairs of $\max_M$ pooling and unpooling, in Figure 3.4.

Discussion on $\max_M$ pooling/unpooling Pooling is popular in convolutional networks to obtain translation-invariant features [89]. It is yet less common in other

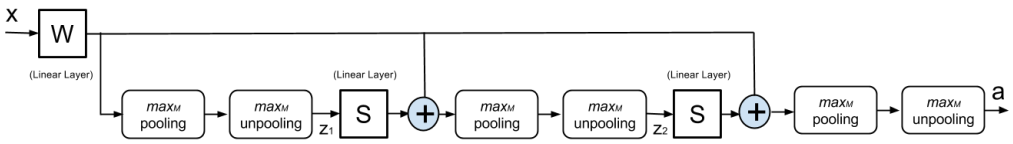


Figure 3.4: Deep M-sparse Encoder, with two time-unfolded stages.

forms of deep networks [72]. The unpooling operation was introduced in [181] to insert the pooled values back to the appropriate locations of feature maps for reconstruction purposes.

In our proposed Deep M-sparse Encoder, the pooling and unpooling operation pair is used to construct a projection from R^m to its subset $S : \{s \in R^m | ||s||_0 \leq M\}$. The $\max_M$ pooling and unpooling functions are intuitively defined as:

$$\begin{aligned} [\mathbf{p}_M, \mathbf{idx}_M] &= \max_M.\text{pooling}(\mathbf{u}) \\ \mathbf{u}_M &= \max_M.\text{unpooling}(\mathbf{p}_M, \mathbf{idx}_M). \end{aligned} \tag{3.11}$$

For each input $\mathbf{u}$, the *pooled map* $\mathbf{p}_M$ records the top M-largest values (irrespective of sign), and the *switch* $\mathbf{idx}_M$ records their locations. The corresponding unpooling operation takes the elements in $\mathbf{p}_M$ and places them in $\mathbf{u}_M$ at the locations specified by $\mathbf{idx}_M$, the remaining elements being set to zero. The resulting $\mathbf{u}_M$ is of the same dimension as $\mathbf{u}$ but has exactly no more than M non-zero elements. In back propagation, each position in $\mathbf{idx}_M$ is propagated with the entire error signal.

Theoretical Properties

It is shown in [17] that the iterative algorithms in both (3.6) and (3.10) are guaranteed not to increase the cost functions. Under mild conditions, their targeted fixed points are local minima of the original problems. As the next step after the time truncation, the deep encoder models are to be solved by the stochastic gradient descent (SGD) algorithm, which converges to stationary points under a few stricter assumptions than ones satisfied in this section [18].[1] However, the entanglement of the iterative algorithms and the SGD algorithm makes the overall convergence analysis a serious hardship.

[1] As a typical case, we use SGD in a setting where it is not guaranteed to converge in theory, but behaves well in practice.

One must emphasize that in each step, the back propagation procedure requires only operations of order O(p) [68]. The training algorithm takes O(Cnp) time (C is the constant absorbing epochs, stage numbers, etc.). The testing process is purely feed-forward and is therefore dramatically faster than traditional inference methods by solving (3.1) or (3.2). SGD is also easy to be parallelized.

Table 3.1: Prediction error (%) comparison on solving the ℓ_0-regularized problem (3.1)

p	128	256	512
Iterative (2 iterations)	17.52	18.73	22.40
Iterative (5 iterations)	8.14	6.75	9.37
Iterative (10 iterations)	3.55	4.33	4.08
Baseline Encoder	8.94	8.76	10.17
Deep ℓ_0-Regularized Encoder	0.92	0.91	0.81

3.2.3 Task-Driven Optimization

It is often desirable to jointly optimize the learned sparse code features and the targeted task so that they mutually reinforce each other. The authors of [80] associated label information with each dictionary item by adding discriminable regularization terms to the objective. Recent work [113], [165] developed task-driven sparse coding via bi-level optimization models, where (ℓ_1-based) sparse coding is formulated as the lower-level constraint while a task-oriented cost function is minimized as its upper-level objective. The above approaches in sparse coding are complicated and computationally expensive. It is much more convenient to implement end-to-end task-driven training in deep architectures, by concatenating the proposed deep encoders with certain task-driven loss functions.

In this section, we mainly discuss two tasks: classification and clustering, while being aware of other immediate extensions, such as semi-supervised learning. Assuming K classes (or clusters), and $\boldsymbol{\omega} = [\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}^k]$ as the set of parameters of the loss function, where $\boldsymbol{\omega}_i$ corresponds to the j-th class (cluster), $j = 1, 2, ..., K$. For the **classification** case, one natural choice is the well-known softmax loss function. For the **clustering** case, we adopt the entropy-minimization loss function in Section 2.2.3.

3.2.4 Experiment

Implementation

Two proposed deep ℓ_0 encoders are implemented with the CUDA ConvNet package [89]. We use a constant learning rate of 0.01 with no momentum, and a batch size of 128. In practice, given that the model is well initialized, the training takes approximately 1 hour on the MNIST dataset, on a workstation with 12 Intel Xeon 2.67GHz CPUs and 1 GTX680 GPU. It is also observed that the training efficiency of our model scales approximately linearly with the size of data.

While many neural networks train well with random initializations without pre-training, given that the training data is sufficient, it has been discovered that poorly initialized networks can hamper the effectiveness of first-order methods (e.g., SGD) [153]. For the proposed models, it is however much easier to initialize the model in the right regime, benefiting from the analytical relationships between sparse coding and network hyperparameters in (3.8).

Simulation on ℓ_0 Sparse Approximation

We first compare the performance of different methods on ℓ_0 sparse code approximation. The first 60,000 samples of the MNIST dataset are used for training and the last 10,000 for testing. Each patch is resized to 16×16 and then preprocessed to remove its mean and normalize its variance. The patches with small standard deviations are discarded. A sparsity coefficient $\lambda = 0.5$ is used in (3.1), and the sparsity level $M = 32$ is fixed in (3.2). The sparse code dimension (dictionary size) p is to be varied.

Our prediction task resembles the setup in [68]: first learning a dictionary from training data, followed by solving sparse approximation (3.3) with respect to the dictionary, and finally training the network as a regressor from input samples to the solved sparse codes. The only major difference here lies in that unlike the ℓ_1-based problems, the non-convex ℓ_0-based minimization could only reach a (non-unique) local minimum. To improve stability, we first solve the ℓ_1 problems to obtain a good initialization for ℓ_0 problems, and then run the iterative algorithms (3.6) or (3.10) until convergence. The obtained sparse codes are called "optimal codes" hereinafter and used in both training and testing evaluation (as "groundtruth"). One must keep in mind that we are not seeking to produce approximate sparse

code for all possible input vectors, but only for *input vectors drawn from the same distribution as our training samples*.

Table 3.2: Prediction error (%) comparison on solving the M-sparse problem (3.2)

p	128	256	512
Iterative (2 iterations)	17.23	19.27	19.31
Iterative (5 iterations)	10.84	12.52	12.40
Iterative (10 iterations)	5.67	5.44	5.20
Baseline Encoder	14.04	16.76	12.86
Deep M-Sparse Encoder	2.94	2.87	3.29

Table 3.3: Averaged non-zero support error comparison on solving the M-sparse problem (3.2)

p	128	256	512
Iterative (2 iterations)	10.8	13.4	13.2
Iterative (5 iterations)	6.1	8.0	8.8
Iterative (10 iterations)	4.6	5.6	5.3
Deep M-Sparse Encoder	2.2	2.7	2.7

We compare the proposed deep ℓ_0 encoders with the iterative algorithms under different number of iterations. In addition, we include a *baseline encoder* in the comparison, which is a fully-connected feed-forward network, consisting of three hidden layers of dimension p with ReLu neurons. The baseline encoder thus has the same parameter capacity as deep ℓ_0 encoders.[2] We apply dropout to the baseline encoders, with the probabilities of retaining the units being 0.9, 0.9, and 0.5. The proposed encoders do not apply dropout.

The deep ℓ_0 encoders and the baseline encoder are first trained, and all are then evaluated on the testing set. We calculate the total prediction errors, i.e., the normalized squared errors between the optimal codes and the predicted codes, as in Tables 3.1 and 3.2. For the M-sparse case, we also compare their recovery of nonzero supports in Table 3.3, by counting the mismatched nonzero element locations between optimal and predicted codes (averaged on all samples). Immediate conclusions from the numerical results are as follows:

[2]Except for the "diag(θ)" layers, each of which contains only p free parameters.

Table 3.4: Classification error rate (%) comparison on the MNIST dataset

p	128	256	512
TDSC	0.71	0.55	0.53
Tuned LISTA	0.74	0.62	0.57
Deep ℓ_0-Regularized	0.72	0.58	0.52
Deep M-Sparse (M = 10)	0.72	0.57	0.53
Deep M-Sparse (M = 20)	0.69	0.54	0.51
Deep M-Sparse (M = 30)	0.73	0.57	0.52

- The proposed deep encoders have outstanding generalization performances, thanks to the effective regularization brought by their task-specific architectures, derived from specific formulations (i.e., (3.1) and (3.2)) as priors. The "general-architecture" baseline encoders, which have the same parameter complexity, appear to overfit the training set and generalize much worse.
- While the deep encoders only unfold two stages, they outperform their iterative counterparts even when the later ones have passed 10 iterations. Meanwhile, the former enjoy much faster inference as being feed-forward.
- The Deep ℓ_0-Regularized Encoder obtains a particularly low prediction error. It is interpretable that while the iterative algorithm has to work with a fixed λ, the Deep ℓ_0-Regularized Encoder is capable of "fine-tuning" this hyper-parameter automatically (after diag(θ) is initialized from λ), by exploring the training data structure.
- The Deep M-Sparse Encoder is able to find the nonzero support with high accuracy.

Applications on Classification

Since the task-driven models are trained from end to end, **no pre-computation of a is needed**. For classification, we evaluate our methods on the MNIST dataset, and the AVIRIS Indiana Pines hyperspectral image dataset (see [164] for details). We compare our two proposed deep encoders with two competitive sparse coding-based methods: 1) task-driven sparse coding (TDSC) in [113], with the original setting followed and all parameters carefully tuned; 2) a pre-trained LISTA followed by supervised tuning with softmax loss. Note that for Deep M-Sparse

Encoder, M is not known in advance and has to be tuned. To our surprise, the fine-tuning of M is likely to improve the performance significantly, which is analyzed next. The overall error rates are compared in Tables 3.4 and 3.5.

Table 3.5: Classification error rate (%) comparison on the AVIRIS Indiana Pines dataset

p	128	256	512
TDSC	15.55	15.27	15.21
Tuned LISTA	16.12	16.05	15.97
Deep ℓ_0-Regularized	15.20	15.07	15.01
Deep M-Sparse (M = 10)	13.77	13.56	13.52
Deep M-Sparse (M = 20)	14.67	14.23	14.07
Deep M-Sparse (M = 30)	15.14	15.02	15.00

In general, the proposed deep ℓ_0 encoders provide superior results to the deep ℓ_1-based method (tuned LISTA). TDSC also generates competitive results, but at the cost of the high complexity for inference, i.e., solving conventional sparse coding. It is of particular interest to us that when supplied with specific M values, the Deep M-Sparse encoder can generate remarkably improved results.[3] Especially in Table 3.5, when M = 10, the error rate is around 1.5% lower than that of M = 30. Note that in the AVIRIS Indiana Pines dataset, the training data volume is much smaller than that of MNIST. In that way, we conjecture that it might not be sufficiently effective to depend the training process fully on data; instead, to craft a stronger sparsity prior by smaller M could help learn more discriminative features.[4] Such a behavior provides us with an important hint to **impose suitable structural priors to deep networks.**

Applications on Clustering

For clustering, we evaluate our methods on the COIL 20 and the CMU PIE dataset [143]. Two state-of-the-art methods to compare are the jointly optimized sparse coding and clustering method proposed in [165], as well as the graph-regularized

[3]To get a good estimate of M, one might first try to perform (unsupervised) sparse coding on a subset of samples.

[4]Interestingly, there are a total of 16 classes in the AVIRIS Indiana Pines dataset. When p = 128, each class has on average 8 "atoms" for class-specific representation. Therefore M = 10 approximately coincides with the sparse representation classification (SRC) principle [164] of forcing sparse codes to be compactly focused on one class of atoms.

deep clustering method in [162].[5] The overall error rates are compared in Tables 3.6 and 3.7.

Note that the method in [162] incorporated Laplacian regularization as an additional prior while the others did not. It is thus no wonder that this method often performs better than others. Even without any graph information utilized, the proposed deep encoders are able to obtain very close performances, and outperform [162] in certain cases. On the COIL 20 dataset, the lowest error rate is reached by the Deep M-Sparse (M = 10) Encoder, when p = 512, followed by the Deep ℓ_0-Regularized Encoder.

Table 3.6: Clustering error rate (%) comparison on the COIL 20 dataset

p	128	256	512
[165]	17.75	17.14	17.15
[162]	14.47	14.17	14.08
Deep ℓ_0-Regularized	14.52	14.27	14.06
Deep M-Sparse (M = 10)	14.59	14.25	14.03
Deep M-Sparse (M = 20)	14.84	14.33	14.15
Deep M-Sparse (M = 30)	14.77	14.37	14.12

Table 3.7: Clustering error rate (%) comparison on the CMU PIE dataset

p	128	256	512
[165]	17.50	17.26	17.20
[162]	16.14	15.58	15.09
Deep ℓ_0-Regularized	16.08	15.72	15.41
Deep M-Sparse (M = 10)	16.77	16.46	16.02
Deep M-Sparse (M = 20)	16.44	16.23	16.05
Deep M-Sparse (M = 30)	16.46	16.17	16.01

On the CMU PIE dataset, the Deep ℓ_0-Regularized Encoder leads to competitive accuracies with [162], and outperforms all Deep M-Sparse Encoders with noticeable margins, which is different from other cases. Pervious work discovered that sparse approximations over CMU PIE had significant errors [177], which is also verified by us. Therefore, hardcoding exact sparsity could even hamper the model performance.

[5]Both papers train their model under both soft-max and max-margin type losses. To ensure fair comparison, we adopt the former, with the same form of loss function as ours.

Remark: From those experiments, we gain additional insights in designing deep architectures:

- If one expects the model to explore the data structure by itself, and provided that there is sufficient training data, then the Deep ℓ_0-Regularized Encoder (and its peers) might be preferred as all its parameters, including the desired sparsity, are fully learnable from the data.
- If one has certain correct prior knowledge of the data structure, including but not limited to the exact sparsity level, one should choose Deep M-Sparse Encoder, or other models of its type that are designed to maximally enforce that prior. The methodology could be especially useful when the training data is less than sufficient.

We hope the above insights could be of reference to many other deep models.

3.2.5 Conclusion

We propose Deep ℓ_0 Encoders to solve the ℓ_0 sparse approximation problem. Rooted in solid iterative algorithms, the deep ℓ_0 regularized encoder and deep M-sparse encoder are developed, each designed to solve one typical formulation, accompanied with the introduction of the novel HELU neuron and $\max_M$ pooling/unpooling. When applied to specific tasks of classification and clustering, the models are optimized in an end-to-end manner. The latest deep learning tools enable us to solve them in a highly effective and efficient fashion. They not only provide us with impressive performances in numerical experiments, but also inspire us with important insights into designing deep models.

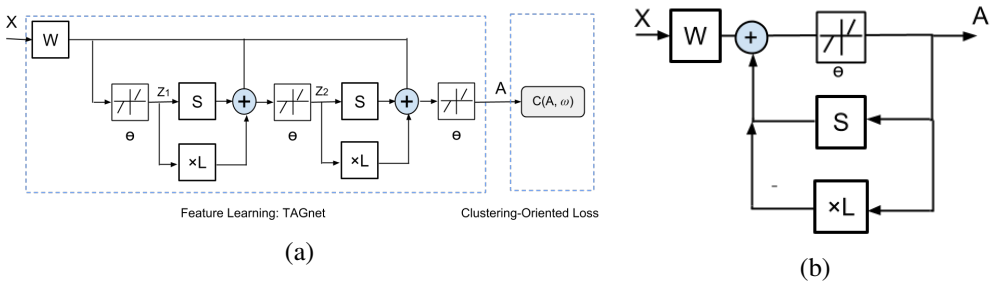
3.3 Task-Specific Deep Sparse Coding for Clustering

3.3.1 Introduction

Effectiveness and scalability are two major concerns in designing a clustering algorithm under Big Data scenarios [29]. Conventional sparse coding models rely on iterative approximation algorithms, whose inherently sequential structure as

well as the data-dependent complexity and latency often constitute a major bottleneck in the computational efficiency [68]. That also results in the difficulty when one tries to jointly optimize the unsupervised feature learning and the supervised task-driven steps [113]. What is more, to effectively model and represent datasets of growing sizes, sparse coding needs to refer to larger dictionaries [96]. Since the inference complexity of sparse coding increases more than linearly with respect to the dictionary size [165], the scalability of sparse coding-based clustering work turns out to be quite limited.

To conquer those limitations, we are motivated to introduce the tool of deep learning in clustering, to which there has been a lack of attention paid. The advantages of deep learning are achieved by its large learning capacity, the linear scalability with the aid of stochastic gradient descent (SGD), and the low inference complexity [9]. The feed-forward networks could be naturally tuned jointly with task-driven loss functions. On the other hand, generic deep architectures [89] largely ignore the problem-specific formulations and prior knowledge. As a result, one may encounter difficulties in choosing optimal architectures, interpreting their working mechanisms, and initializing the parameters.



(a) (b)

Figure 3.5: (a) The proposed pipeline, consisting of the TAGnet network for feature learning, followed by the clustering-oriented loss functions. The parameters $\mathbf{W}, \mathbf{S}, \boldsymbol{\theta}$ and $\boldsymbol{\omega}$ are all learnt end-to-end from training data. (b) The block diagram of solving (3.14).

In this section, we demonstrate how to **combine the sparse coding-based pipeline into deep learning models for clustering**. The proposed framework takes advantage of both sparse coding and deep learning. Specifically, the feature learning layers are inspired by the graph-regularized sparse coding inference process, via reformulating iterative algorithms [68] into a feed-forward network, named **TAGnet**. Those layers are then jointly optimized with the task-specific loss functions from end to end. Our technical merits are summarized in three aspects:

- As a deep feed-forward model, the proposed framework provides extremely efficient inference process and high scalability to large scale data. It allows to learn more descriptive features than conventional sparse codes.
- We discover that incorporating the expertise of sparse code-based clustering pipelines [36, 185] improves our performance significantly. Moreover, it greatly facilitates the model initialization and interpretation.
- We further enforce auxiliary clustering tasks on the hierarchy of features, we develop **DTAGnet** and observe further performance boosts on the CMU MultiPIE dataset [69].

3.3.2 Related Work

Sparse coding for clustering

Assuming data samples $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]$, where $\mathbf{x}_i \in \mathbf{R}^{m\times 1}$ and $i = 1, 2, \cdots, n$. They are encoded into sparse codes $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_n]$, where $\mathbf{a}_i \in \mathbf{R}^{p\times 1}$ and $i = 1, 2, \cdots, n$, using a learned dictionary $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_p]$, where $\mathbf{d}_i \in \mathbf{R}^{m\times 1}, i = 1, 2, \cdots, p$ are the learned atoms. The sparse codes are obtained by solving the following convex optimization (λ is a constant):

$$\mathbf{A} = \arg\min_{\mathbf{A}} \tfrac{1}{2}||\mathbf{X} - \mathbf{D}\mathbf{A}||_F^2 + \lambda \textstyle\sum_i ||\mathbf{a}_i||_1, \ . \tag{3.12}$$

In [36], the authors suggested that the sparse codes can be used to construct the similarity graph for spectral clustering [124]. Furthermore, to capture the geometric structure of local data manifolds, the graph regularized sparse codes are further suggested in [185, 178] by solving:

$$\mathbf{A} = \arg\min_{\mathbf{A}} \tfrac{1}{2}||\mathbf{X} - \mathbf{D}\mathbf{A}||_F^2 + \lambda \textstyle\sum_i ||\mathbf{a}_i||_1 + \tfrac{\alpha}{2} Tr(\mathbf{A}\mathbf{L}\mathbf{A}^{\mathbf{T}}), \tag{3.13}$$

where $\mathbf{L}$ is the graph Laplacian matrix and can be constructed from a pre-chosen pairwise similarity (affinity) matrix $\mathbf{P}$. More recently in [165], the authors suggested simultaneously learning feature extraction and discriminative clustering, by formulating a task-driven sparse coding model [113]. They proved that such joint methods consistently outperformed non-joint counterparts.

Deep learning for clustering

In [154], the authors explored the possibility of employing deep learning in graph clustering. They first learned a nonlinear embedding of the original graph by an auto encoder (AE), followed by a K-means algorithm on the embedding to obtain the final clustering result. However, it neither exploits more adapted deep architectures nor performs any task-specific joint optimization. In [32], a deep belief network with nonparametric clustering was presented. As a generative graphical model, DBN provides a faster feature learning, but is less effective than AEs in terms of learning discriminative features for clustering. In [157], the authors extended the semi non-negative matrix factorization (SNMF) model to a Deep SNMF model, whose architecture resembles stacked AEs. Our proposed model is substantially different from all these previous approaches, due to its unique task-specific architecture derived from sparse coding domain expertise, as well as the joint optimization with clustering-oriented loss functions.

3.3.3 Model Formulation

The proposed pipeline consists of two blocks. As depicted in Figure 3.5 (a), it is trained end-to-end in an unsupervised way. It includes a feed-forward architecture, termed *Task-specific And Graph-regularized Network* (**TAGnet**), to learn discriminative features, and the clustering-oriented loss function (the same as Section 2.2.3).

TAGnet: Task-specific And Graph-regularized Network

Different from generic deep architectures, TAGnet is designed in a way to take advantage of the successful sparse code-based clustering pipelines [185, 165]. It aims to learn features that are optimized under clustering criteria, while encoding graph constraints (3.13) to regularize the target solution. TAGnet is derived from the following theorem:

Theorem 3.3.1. *The optimal sparse code* $\mathbf{A}$ *from (3.13) is the fixed point of*

$$\mathbf{A} = h_{\frac{\lambda}{N}}[(\mathbf{I} - \tfrac{1}{N}\mathbf{D}^T\mathbf{D})\mathbf{A} - \mathbf{A}(\tfrac{\alpha}{N}\mathbf{L}) + \tfrac{1}{N}\mathbf{D}^T\mathbf{X}], \tag{3.14}$$

where $h_{\boldsymbol{\theta}}$ is an element-wise shrinkage function parameterized by $\boldsymbol{\theta}$:

$$[h_{\boldsymbol{\theta}}(\mathbf{u})]_i = sign(\mathbf{u}_i)(|\mathbf{u}_i| - \boldsymbol{\theta}_i)_+. \tag{3.15}$$

N is an upper bound on the largest eigenvalue of $\mathbf{D}^T\mathbf{D}$.

The proof of Theorem 3.4.1 can be completed with similar techniques in [68]. Theorem 3.4.1 outlines an iterative algorithm to solve (3.13). Under quite mild conditions [7], after $\mathbf{A}$ is initialized, one may repeat the shrinkage and thresholding process in (3.14) until convergence. Moreover, the iterative algorithm could be alternatively expressed as the block diagram in Figure 3.5 (b), where

$$\mathbf{W} = \tfrac{1}{N}\mathbf{D}^T, \mathbf{S} = \mathbf{I} - \tfrac{1}{N}\mathbf{D}^T\mathbf{D}, \boldsymbol{\theta} = \tfrac{\lambda}{N}. \tag{3.16}$$

In particular, we define the new operator "$\times\mathbf{L}$": $\mathbf{A} \rightarrow -\frac{\alpha}{N}\mathbf{A}\mathbf{L}$, where the input $\mathbf{A}$ is multiplied by the pre-fixed $\mathbf{L}$ from the right side and scaled by $-\frac{\alpha}{N}$.

By time-unfolding and truncating Figure 3.5 (b) to a fixed number of K iterations (K = 2 by default)[6], we obtain the TAGnet form in Figure 3.5 (a). $\mathbf{W}$, $\mathbf{S}$ and $\boldsymbol{\theta}$ are all to be learnt jointly from data. $\mathbf{S}$ and $\boldsymbol{\theta}$ are tied weights for both stages[7]. It is important to note that the output $\mathbf{A}$ of TAGnet is not necessarily identical to the predicted sparse codes by solving (3.13). Instead, the goal of TAGnet is to learn discriminative embedding that is optimal for clustering.

To facilitate training, we further rewrite (3.35) as:

$$[h_{\boldsymbol{\theta}}(\mathbf{u})]_i = \boldsymbol{\theta}_i \cdot \text{sign}(\mathbf{u}_i)(|\mathbf{u}_i|/\boldsymbol{\theta}_i - 1)_+ = \boldsymbol{\theta}_i h_1(\mathbf{u}_i/\boldsymbol{\theta}_i). \tag{3.17}$$

Eqn. (3.37) indicates that the original neuron with trainable thresholds can be decomposed into two linear scaling layers plus a unit-threshold neuron. The weights of the two scaling layers are diagonal matrices defined by $\boldsymbol{\theta}$ and its element-wise reciprocal, respectively.

A notable component in TAGnet is the $\times\mathbf{L}$ **branch** of each stage. The graph Laplacian $\mathbf{L}$ could be computed in advance. In the feed-forward process, a $\times\mathbf{L}$ branch takes the intermediate $\mathbf{Z}_k$ (k = 1, 2) as the input, and applies the "$\times\mathbf{L}$" operator defined above. The output is aggregated with the output from the learnable $\mathbf{S}$ layer. In the back propagation, $\mathbf{L}$ will not be altered. In such a way, the graph

[6]We test larger K values (3 or 4), but they do not bring noticeable performance improvements.

[7]Out of curiosity, we have also tried the architecture that treat $\mathbf{W}$, $\mathbf{S}$ and $\boldsymbol{\theta}$ in both stages as independent variables. We find that sharing parameters improves the performance.

regularization is effectively encoded in the TAGnet structure as a prior.

An appealing highlight of (D)TAGnet lies in its very effective and straightforward initialization strategy. With sufficient data, many latest deep networks train well with random initializations without pre-training. However, it has been discovered that poor initializations hamper the effectiveness of first-order methods (e.g., SGD) in certain cases [153]. For (D)TAGnet, it is however much easier to initialize the model in the right regime. That benefits from the analytical relationships between sparse coding and network hyperparameters defined in (3.16): we could initialize deep models from corresponding sparse coding components, the latter of which is easier to obtain. Such an advantage becomes much more important when the training data is limited.

Model Complexity The proposed framework can handle large-scale and high-dimensional data effectively via the stochastic gradient descent (SGD) algorithm. In each step, the back propagation procedure requires only operations of order O(p) [68]. The training algorithm takes O(Cnp) time (C is a constant in terms of the total numbers of epochs, stage numbers, etc.). In addition, SGD is easy to parallelize and thus could be efficiently trained using GPUs.

Connections to Existing Models

There is a close connection between sparse coding and neural networks. In [68], a feed-forward neural network, named LISTA, is proposed to efficiently approximate the sparse code $\mathbf{a}$ of input signal $\mathbf{x}$, which is obtained by solving (3.12) in advance. The LISTA network learns the hyperparameters as a general regression model from training data to their pre-solved sparse codes using back-propagation.

LISTA overlooks the useful geometric information among data points [185], and therefore could be viewed as a **special case** of TAGnet in Figure 3.5 when $\alpha = 0$ (i.e., removing the $\times\mathbf{L}$ branches). Moreover, LISTA aims to approximate the "optimal" sparse codes pre-obtained from (3.12), and therefore requires the estimation of $\mathbf{D}$ and the tedious pre-computation of $\mathbf{A}$. The authors did not exploit its potential in supervised and task-specific feature learning.

3.3.4 A Deeper Look: Hierarchical Clustering by DTAGnet

Deep networks are well known for their capabilities to learn semantically rich representations by hidden layers [51]. In this section, we investigate how the intermediate features $\mathbf{Z}_k$ ($k = 1, 2$) in TAGnet (Figure 3.5 (a)) can be interpreted, and further utilized to improve the model, for specific clustering tasks. Compared to related non-deep models [165], such a hierarchical clustering property is another unique advantage of being deep.

Our strategy is mainly inspired by the algorithmic framework of deeply supervised nets [94]. As in Figure 3.6, our proposed Deeply-Task-specific And Graph-regularized Network (**DTAGnet**) brings in additional deep feedbacks, by associating a clustering-oriented local auxiliary loss $C_k(\mathbf{Z}_k, \boldsymbol{\omega}_k)$ (k = 1, 2) with each stage. Such an auxiliary loss takes the same form as the overall $C(\mathbf{A}, \boldsymbol{\omega})$, except that the expected cluster number may be different, depending on the auxiliary clustering task to be performed. The DTAGnet backpropagates errors not only from the overall loss layer, but also simultaneously from the auxiliary losses.

While seeking the optimal performance of the target clustering, DTAGnet is also driven by two auxiliary tasks that are explicitly targeted at clustering specific attributes. It enforces constraint at each hidden representation for directly making a good cluster prediction. In addition to the overall loss, the introduction of auxiliary losses gives another strong push to obtain discriminative and sensible features at each individual stage. As discovered in the classification experiments in [94], the auxiliary loss both acts as feature regularization to reduce generalization errors and results in faster convergence. We also find later that every $\mathbf{Z}_k$ ($k = 1, 2$) is indeed most suited for its targeted task.

In [157], a Deep SNMF model was proposed to learn hidden representations, that grant themselves an interpretation of clustering according to different attributes. The authors considered the problem of mapping facial images to their identities. A face image also contains attributes like pose and expression that help identify the person depicted. In their experiments, the authors found that by further factorizing this mapping in a way that each factor adds an extra layer of abstraction, the deep model could automatically learn latent intermediate representations that are implied for clustering identity-related attributes. Although there is a clustering interpretation, those hidden representations are not specifically optimized in clustering sense. Instead, the entire model is trained with only the overall reconstruction loss, after which clustering is performed using K-means

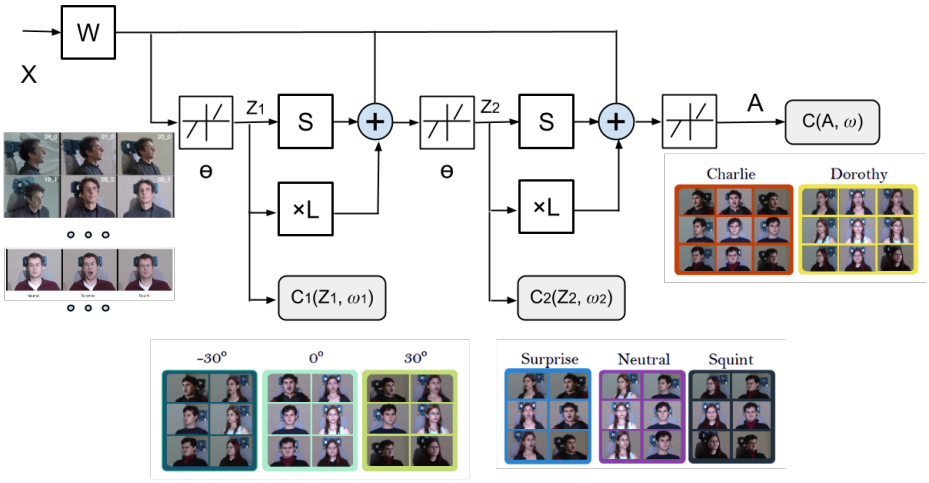


Figure 3.6: The DTAGnet architecture, taking the CMU MultiPIE dataset as an example. The model is able to simultaneously learn features for pose clustering ($\mathbf{Z}_1$), for expression clustering ($\mathbf{Z}_2$), and for identity clustering ($\mathbf{A}$). The first two attributes are related to and helpful for the last (overall) task. Part of image sources are referred from [69] and [157].

on learnt features. Consequently, their clustering performance is not satisfactory. Our study shares the similar observation and motivation with [157], but in a more task-specific manner by performing the optimizations of auxiliary clustering tasks jointly with the overall task.

3.3.5 Experiment Results

Datasets and measurements

We evaluate the proposed model on three publicly available datasets:

- **MNIST** [185] consists of a total number of 70, 000 quasi-binary, handwritten digit images, with digits 0 to 9. The digits are normalized and centered in fixed-size images of 28×28.

- **CMU MultiPIE** [69] contains around 750, 000 images of 337 subjects, that are captured under varied laboratory conditions. A unique property of CMU MultiPIE lies in that each image comes with labels for the identity, illumination, pose and expression attributes. That is why CMU MultiPIE is chosen in [157] to learn multi-attribute features (Figure 3.6) for hierarchical clus-

tering. In our experiments, we follow [157] and adopt a subset of 13, 230 images of 147 subjects in 5 different poses and 6 different emotions. Notably, we do not pre-process the images by using piece-wise affine warping as utilized by [157] to align these images.

- **COIL20** [123] contains 1, 440 32×32 gray scale images of 20 objects (72 images per object). The images of each object were taken 5 degrees apart.

Although the section only evaluates the proposed method using image datasets, the methodology itself is not limited to only image subjects. We apply two widely-used measures to evaluate the clustering performances: the accuracy and the Normalized Mutual Information (NMI) [185], [36]. We follow the convention of clustering work [185, 178, 165], and do not distinguish training from testing. We train our models on all available samples of each dataset, reporting the clustering performances as our testing results. Results are averaged from 5 independent runs.

Experiment settings

The proposed networks are implemented using the cuda-convnet package [89]. The network takes $K = 2$ stages by default. We apply a constant learning rate of 0.01 with no momentum to all trainable layers. The batch size is 128. In particular, to encode graph regularization as a prior, we fix $\mathbf{L}$ during model training by setting its learning rate to be 0. Experiments are run on a workstation with 12 Intel Xeon 2.67GHz CPUs and 1 GTX680 GPU. The training takes approximately 1 hour on the MNIST dataset. It is also observed that the training efficiency of our model scales approximately linearly with data.

In our experiments, we set the default value of α to be 5, p to be 128, and λ to be chosen from [0.1, 1] by cross-validation.[8] A dictionary $\mathbf{D}$ is first learned from $\mathbf{X}$ by K-SVD [60]. $\mathbf{W}$, $\mathbf{S}$ and $\boldsymbol{\theta}$ are then initialized based on (3.16). $\mathbf{L}$ is also pre-calculated from $\mathbf{P}$, which is formulated by the Gaussian Kernel: $\mathbf{P}_{ij} = \exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{\delta^2})$ (δ is also selected by cross-validation). After obtaining the output $\mathbf{A}$ from the initial (D)TAGnet models, $\boldsymbol{\omega}$ (or $\boldsymbol{\omega}_k$) could be initialized based on minimizing (2.22) or (2.27) over $\mathbf{A}$ (or $\mathbf{Z}_k$).

[8] The default values of α and p are inferred from the related sparse coding literature [185], and validated in experiments.

Comparison experiments and analysis

Benefits of the task-specific deep architecture We denote the proposed model of TAGnet plus entropy-minimization loss (EML) (2.22) as TAGnet-EML, and the one plus maximum-margin loss (MML) (2.27) as TAGnet-MML, respectively. We include the following comparison methods:

- We refer to the initializations of the proposed joint models as their **"Non-Joint"** counterparts, denoted as NJ-TAG-EML and NJ-TAG-MML (NJ short for non-joint), respectively.

- We design a **Baseline Encoder** (BE), which is a fully-connected feedforward network, consisting of three hidden layers of dimension p with ReLU neuron. It is obvious that the BE has the *same parameter complexity* as TAGnet.[9] The BEs are also tuned by EML or MML in the same way, denoted as BE-EML or BE-MML, respectively. We intend to verify our important argument, that *the proposed model benefits from the task-specific TAGnet architecture, rather than just the large learning capacity of generic deep models*.

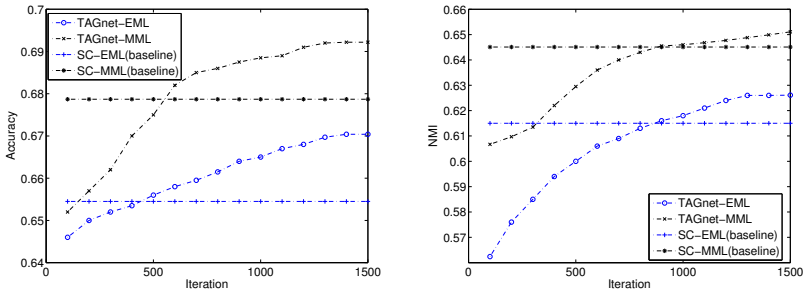
- We compare the proposed models with their closest **"shallow"** competitors, i.e., the joint optimization methods of graph-regularized sparse coding and discriminative clustering in [165]. We re-implement their work using both (2.22) or (2.27) losses, denoted as SC-EML and SC-MML (SC short for sparse coding). Since in [165] the authors already revealed SC-MML outperforms the classical methods such as MMC and ℓ_1 graph methods, we do not compare with them again.

- We also include **Deep SNMF** [157] as a state-of-the-art deep learning-based clustering work. We mainly compare our results with their reported performances on CMU MultiPIE.[10]

As revealed by the full comparison results in Table 3.8, the proposed task-specific deep architectures outperform others by a noticeable margin. The underlying domain expertise guides the data-driven training in a more principled way. In contrast, the "general-architecture" baseline encoders (BE-EML and BE-MML) appear to produce much worse (even worst) results. Furthermore, it is evident that the proposed end-to-end optimized models outperform their "non-joint"

[9] Except for the "θ" layers, each of which contains only p free parameters and is thus ignored

[10] With various component numbers tested in [157], we choose their best cases (60 components).



(a) (b)

Figure 3.7: The accuracy and NMI plots of TAGnet-EML/TAGnet-MML on MNIST, starting from the initialization, and tested every 100 iterations. The accuracy and NMI of SC-EML/SC-MML are also plotted as baselines.

Table 3.8: Accuracy and NMI performance comparisons on all three datasets

		TAGnet -EML	TAGnet -MML	NJ-TAG -EML	NJ-TAG -MML	BE -EML	BE -MML	SC -EML	SC -MML	Deep SNMF
MNIST	Acc	0.6704	0.6922	0.6472	0.5052	0.5401	0.6521	0.6550	0.6784	/
	NMI	0.6261	0.6511	0.5624	0.6067	0.5002	0.5011	0.6150	0.6451	/
CMU MultiPIE	Acc	0.2176	0.2347	0.1727	0.1861	0.1204	0.1451	0.2002	0.2090	0.17
	NMI	0.4338	0.4555	0.3167	0.3284	0.2672	0.2821	0.3337	0.3521	0.36
COIL20	Acc	0.8553	0.8991	0.7432	0.7882	0.7441	0.7645	0.8225	0.8658	/
	NMI	0.9090	0.9277	0.8707	0.8814	0.8028	0.8321	0.8850	0.9127	/

counterparts. For example, on the MNIST dataset, TAGnet-MML surpasses NJ-TAG-MML by around 4% in accuracy and 5% in NMI.

By comparing the TAGnet-EML/TAGnet-MML with SC-EML/SC-MML, we draw a promising conclusion: adopting a more parameterized deep architecture allows a larger feature learning capacity compared to conventional sparse coding. Although similar points are well made in many other fields [89], we are interested in a closer look between the two. Figure 3.7 plots the clustering accuracy and NMI curves of TAGnet-EML/TAGnet-MML on the MNIST dataset, along with iteration numbers. Each model is well initialized at the very beginning, and the clustering accuracy and NMI are computed every 100 iterations. At first, the clustering performances of deep models are even slightly worse than sparse-coding methods, mainly since the initialization of TAGnet hinges on a truncated approximation of graph-regularized sparse coding. After a small number of iterations, the performance of the deep models surpasses sparse coding ones, and continues rising monotonically until reaching a higher plateau.

Effects of graph regularization In (3.13), the graph regularization term imposes stronger smoothness constraints on the sparse codes with a larger α. It also happens to the TAGnet. We investigate how the clustering performances of TAGnet-EML/TAGnet-MML are influenced by various α values. From Figure 3.8, we observe the identical general tendency on all three datasets. While α increases, the accuracy/NMI result will first rise then decrease, with the peak appearing between $\alpha \in [5, 10]$. As an interpretation, the local manifold information is not sufficiently encoded when α is too small (α = 0 will completely disable the $\times\mathbf{L}$ branch of TAGnet, and reduces it to the LISTA network [68] fine-tuned by the losses). On the other hand, when α is large, the sparse codes are "over-smoothened" with a reduced discriminative ability. Note that similar phenomena are also reported in other relevant literature, e. g., [185, 165].

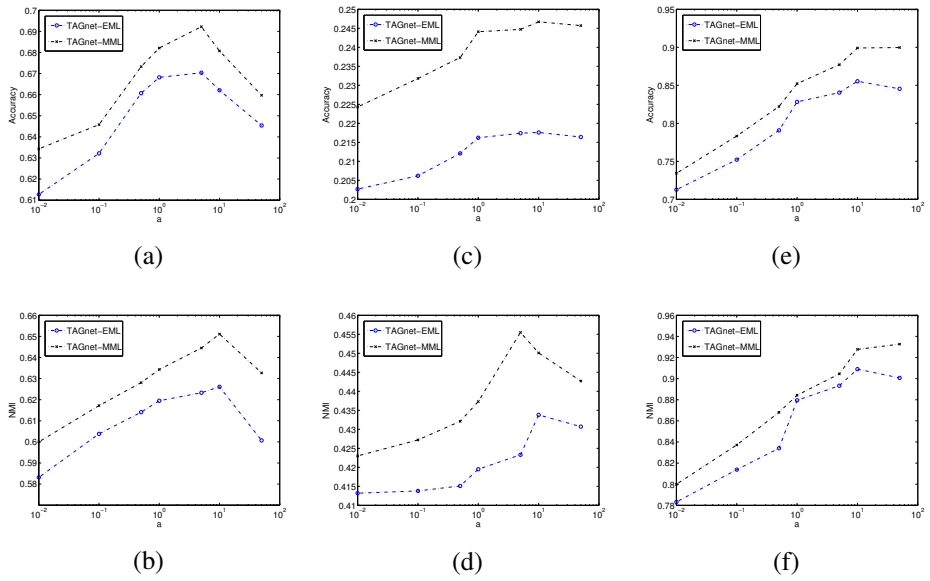
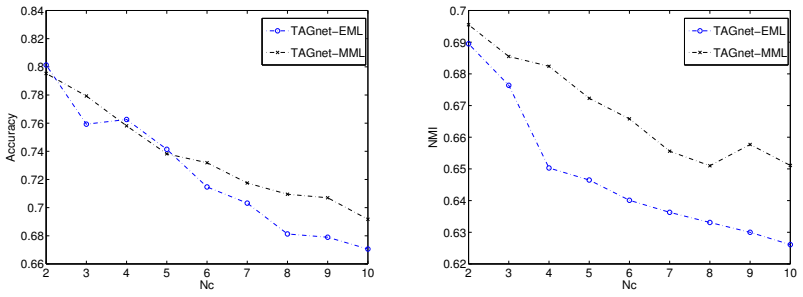


Figure 3.8: The clustering accuracy and NMI plots (x-axis logarithm scale) of TAGnet-EML/TAGnet-MML versus the parameter choices of α, on: (a) (b) MNIST; (c) (d) CMU MultiPIE; (e) (f) COIL20.

Furthermore, comparing among Figure 3.8 (a) - (f), it is noteworthy to observe how graph regularization behaves differently on three of them. We notice that the COIL20 dataset is the one that is the most sensitive to the choice of α. Increasing α from 0.01 to 50 leads to an improvement of more than 10%, in terms of both accuracy and NMI. It verifies the significance of graph regularization when training samples are limited [178]. On the MNIST dataset, both models achieve a gain of up to 6% in accuracy and 5% in NMI, by tuning α from 0.01 to 10. However,

unlike COIL20 that almost always favors larger α, the model performance on the MNIST dataset tends to be not only saturated, but even significantly hampered when α continues rising to 50. The CMU MultiPIE dataset witnesses moderate improvements of around 2% in both measurements. It is not as sensitive to α as the other two. Potentially, it might be due to the complex variability in original images that makes the graph $\mathbf{W}$ unreliable for estimating the underlying manifold geometry. We suspect that more sophisticated graphs may help alleviate the problem, and will explore it in the future.

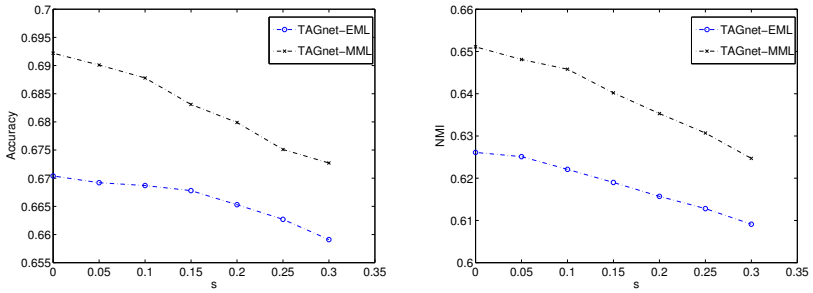
Scalability and robustness On the MNIST dataset, we re-conduct the clustering experiments with the cluster number N_c ranging from 2 to 10, using TAGnet-EML/TAGnet-MML. Figure 3.9 shows that the clustering accuracy and NMI change by varying the number of clusters. The clustering performance transits smoothly and robustly when the task scale changes.



(a) (b)

Figure 3.9: The clustering accuracy and NMI plots of TAGnet-EML / TAGnet-EML versus the cluster number N_c ranging from 2 to 10, on MNIST.

To examine the proposed models' robustness to noise, we add various Gaussian noise, whose standard deviation s ranges from 0 (noiseless) to 0.3, to re-train our MNIST model. Figure 3.10 indicates that both TAGnet-EML and TAGnet-MML have certain robustness to noise. When s is less than 0.1, there is even little visible performance degradation. While TAGnet-MML constantly outperforms TAGnet-EML in all experiments (as MMC is known to be highly discriminative [174]), it is interesting to observe in Figure 3.10 that the latter one is slightly more robust to noise than the former. It is perhaps owing to the probability-driven loss form (2.22) of EML that allows for more flexibility.



(a) (b)

Figure 3.10: The clustering accuracy and NMI plots of TAGnet-EML / TAGnet-MML versus the noise level s, on MNIST.

Hierarchical clustering on CMU MultiPIE

As observed, CMU MultiPIE is very challenging for the basic identity clustering task. However, it comes with several other attributes: pose, expression, and illumination, which could be of assistance in our proposed DTAGnet framework. In this part, we apply the similar setting of [157] on the same CMU MultiPIE subset, by setting pose clustering as the Stage I auxiliary task, and expression clustering as the Stage II auxiliary task.[11] In that way, we target $C_1(\mathbf{Z}_1, \boldsymbol{\omega}_1)$ at 5 clusters, $C_2(\mathbf{Z}_2, \boldsymbol{\omega}_2)$ at 6 clusters, and finally $C(\mathbf{A}, \boldsymbol{\omega})$ as 147 clusters.

The training of DTAGnet-EML/DTAGnet-MML follows the same aforementioned process except for considering extra back-propagated gradients from task $C_k(\mathbf{Z}_k, \boldsymbol{\omega}_k)$ in Stage k (k = 1, 2). Then, we test each $C_k(\mathbf{Z}_k, \boldsymbol{\omega}_k)$ separately on their targeted task. In DTAGnet, each auxiliary task is also jointly optimized with its intermediate feature $\mathbf{Z}_k$, which differentiates our methodology substantially from [157]. It is thus no surprise to see in Table 3.9 that each auxiliary task achieves much better performance than [157].[12] Most notably, the performances of the overall identity clustering task witness **a very impressive boost of around 7% in accuracy**. We also test DTAGnet-EML/DTAGnet-MML with only $C_1(\mathbf{Z}_1, \boldsymbol{\omega}_1)$ or $C_2(\mathbf{Z}_2, \boldsymbol{\omega}_2)$ kept. Experiments verify that by adding auxiliary tasks gradually, the overall task keeps being benefited. Those auxiliary tasks, when enforced together, can also reinforce each other mutually.

[11] In fact, although claimed to be applicable to multiple attributes, [157] only examined the first level features for pose clustering without considering expressions, since it relied on a warping technique to pre-process images, that gets rid of most expression variability.

[12] In [157] Table. 2, it reports that the best accuracy of pose clustering task falls around 28%, using the most suited layer features.

Table 3.9: Effects of incorporating auxiliary clustering tasks in DTAGnet-EML / DTAGnet-MML (P: Pose; E: Expression; I: Identity)

Method	Stage I		Stage II		Overall	
	Task	Acc	Task	Acc	Task	Acc
DTAGnet -EML	/	/	/	/	I	0.2176
	P	0.5067	/	/	I	0.2303
	/	/	E	0.3676	I	0.2507
	P	0.5407	E	0.4027	I	0.2833
DTAGnet -MML	/	/	/	/	I	0.2347
	P	0.5251	/	/	I	0.2635
	/	/	E	0.3988	I	0.2858
	P	0.5538	E	0.4231	I	0.3021

One might ask: **Which one matters more in the performance boost**, the deeply task-specific architecture that brings extra discriminative feature learning, or the proper design of auxiliary tasks that capture the intrinsic data structure characterized by attributes?

Table 3.10: Effects of varying target cluster numbers of auxiliary tasks in DTAGnet-EML / DTAGnet-MML

Method	#clusters in Stage I	#clusters in Stage II	Overall Accuracy
DTAGnet -EML	4	4	0.2827
	8	8	0.2813
	12	12	0.2802
	20	20	0.2757
DTAGnet -MML	4	4	0.3030
	8	8	0.3006
	12	12	0.2927
	20	20	0.2805

To answer this important question, we vary the target cluster number in either $C_1(\mathbf{Z}_1, \boldsymbol{\omega}_1)$ or $C_2(\mathbf{Z}_2, \boldsymbol{\omega}_2)$, and re-conduct the experiments. Table 3.10 reveals that more auxiliary tasks, even those without any straightforward task-specific interpretation (e.g., partitioning the Multi-PIE subset into 4, 8, 12 or 20 clusters hardly makes semantic sense), may still help gain better performances. It is comprehensible that they simply promote more discriminative feature learning in a low-to-high, coarse-to-fine scheme. In fact, it is a complementary observation

to the conclusion found in classification [94]. On the other hand, at least in this specific case, while the target cluster numbers of auxiliary tasks get closer to the ground-truth (5 and 6 here), the models seem to achieve the best performances. We conjecture that when properly "matched", every hidden representation in each layer is in fact most suited for clustering the attributes corresponding to the layer of interest. The whole model resembles the problem of sharing low-level feature filters among several relevant high-level tasks in convolutional networks [65], but in a distinct context.

We hence conclude that the deeply-supervised fashion proves to be helpful for the deep clustering models, even when there are no explicit attributes for constructing a practically meaningful hierarchical clustering problem. However, it is preferable to exploit those attributes when available, as they lead to not only superior performance but more clearly interpretable models. The learned intermediate features can be potentially utilized for multi-task learning [160].

3.3.6 Conclusion

In this section, we present a deep learning-based clustering framework. Trained from end to end, it features a task-specific deep architecture inspired by the sparse coding domain expertise, which is then optimized under clustering-oriented losses. Such a well-designed architecture leads to more effective initialization and training, and significantly outperforms generic architectures of the same parameter complexity. The model could be further interpreted and enhanced by introducing auxiliary clustering losses to the intermediate features. Extensive experiments verify the effectiveness and robustness of the proposed models.

3.4 Deep ℓ_∞ Encoder for Hashing

We investigate the ℓ_∞-constrained representation which demonstrates robustness to quantization errors, utilizing the tool of deep learning. Based on the Alternating Direction Method of Multipliers (ADMM), we formulate the original convex minimization problem as a feed-forward neural network, named *Deep ℓ_∞ Encoder*, by introducing the novel Bounded Linear Unit (BLU) neuron and modeling the Lagrange multipliers as network biases. Such a structural prior acts as an effective network regularization, and facilitates the model initialization. We then inves-

tigate the effective use of the proposed model in the application of hashing, by coupling the proposed encoders under a supervised pairwise loss, to develop a *Deep Siamese ℓ_∞ Network*, which can be optimized from end to end. Extensive experiments demonstrate the impressive performances of the proposed model. We also provide an in-depth analysis of its behaviors against the competitors.

3.4.1 Introduction

Problem Definition and Background

While ℓ_0 and ℓ_1 regularizations have been well-known and successfully applied in sparse signal approximations, it has been less explored to utilize the ℓ_∞ norm to regularize signal representations. In this section, we are particularly interested in the following ℓ_∞-constrained least squares problem:

$$\min_x ||\mathbf{D}x - y||_2^2 \quad s.t. \quad ||x||_\infty \leq \lambda, \tag{3.18}$$

where $y \in R^{n\times 1}$ denotes the input signal, $\mathbf{D} \in R^{n\times N}$ the (overcomplete) basis (often called frame or dictionary) with $N < n$, and $x \in R^{N\times 1}$ the learned representation. Further, the maximum absolute magnitude of x is bounded by a positive constant λ, so that each entry of x has the smallest dynamic range [112]. As a result, the model (3.18) tends to spread the information of y approximately evenly among the coefficients of x. Thus, x is called "democratic" [151] or "anti-sparse" [63], as all of its entries are of approximately the same importance.

In practice, x usually has most entries reaching the same absolute maximum magnitude [151], therefore resembling to an antipodal signal in an N-dimensional Hamming space. Furthermore, the solution x to (3.18) withstands errors in a very powerful way: the representation error gets bounded by the average, rather than the sum, of the errors in the coefficients. These errors may be of arbitrary nature, including distortion (e.g., quantization) and losses (e.g., transmission failure). This property was quantitatively established in Section II.C of [112]:

Theorem 3.4.1. *Assume $||x||_2 < 1$ without loss of generality, and each coefficient of x is quantized separately by performing a uniform scalar quantization of the dynamic range $[-\lambda, \lambda]$ with L levels. The overall quantization error of x from (3.18) is bounded by $\frac{\lambda\sqrt{N}}{L}$. In comparison, a least squares solution x_{LS}, by*

minimizing $||\mathbf{D}x_{LS} - y||_2^2$ *without any constraint, would only give the bound* $\frac{\sqrt{n}}{L}$.

In the case of $N << n$, the above will yield great robustness for the solution to (3.18) with respect to noise, in particular quantization errors. Also note that its error bound will not grow with the input dimensionality n, a highly desirable stability property for high-dimensional data. Therefore, (3.18) appears to be favorable for the applications such as vector quantization, hashing and approximate nearest neighbor search.

In this section, we investigate (3.18) in the context of deep learning. Based on the Alternating Direction Methods of Multipliers (**ADMM**) algorithm, we formulate (3.18) as a feed-forward neural network [68], called **Deep** ℓ_∞ **Encoder**, by introducing the novel Bounded Linear Unit (**BLU**) neuron and modeling the Lagrange multipliers as network biases. The major technical merit is how a specific optimization model (3.18) could be translated to designing a task-specific deep model, which displays the desired quantization-robust property. We then study its application in hashing, by developing a **Deep Siamese** ℓ_∞ **Network** that couples the proposed encoders under a supervised pairwise loss, which could be optimized from end to end. Impressive performances are observed in our experiments.

Related Work

Similar to the case of ℓ_0/ℓ_1 sparse approximation problems, solving (3.18) and its variants (e.g., [151]) relies on iterative solutions. [149] proposed an active set strategy similar to that of [92]. In [2], the authors investigated a primal-dual path-following interior-point method. Albeit effective, the iterative approximation algorithms suffer from their inherently sequential structures, as well as the data-dependent complexity and latency, which often constitute a major bottleneck in the computational efficiency. In addition, the joint optimization of the (unsupervised) feature learning and the supervised steps has to rely on solving complex bi-level optimization problems [165]. Further, to effectively represent datasets of growing sizes, a larger dictionary $\mathbf{D}$ is usually needed. Since the inference complexity of those iterative algorithms increases more than linearly with respect to the dictionary size [11], their scalability turns out to be limited. Last but not least, while the hyperparameter λ sometimes has physical interpretations, e.g., for signal quantization and compression, it remains unclear how to be set or adjusted for many application cases.

3.4.2 An ADMM Algorithm

ADMM has been popular for its remarkable effectiveness in minimizing objectives with linearly separable structures [11]. We first introduce an auxiliary variable $z \in R^{N\times 1}$, and rewrite (3.18) as:

$$\min_{\mathbf{x},\mathbf{z}} \tfrac{1}{2}||\mathbf{D}x - y||_2^2 \quad s.t. \quad ||z||_\infty \leq \lambda, \quad z - x = 0. \tag{3.19}$$

The augmented Lagrangian function of (3.19) is:

$$\tfrac{1}{2}||\mathbf{D}x - y||_2^2 + p^T(z - x) + \tfrac{\beta}{2}||z - x||_2^2 + \Phi_\lambda(z). \tag{3.20}$$

Here $p \in R^{N\times 1}$ is the Lagrange multiplier attached to the equality constraint, β is a positive constant (with a default value 0.6), and $\Phi_\lambda(z)$ is the indicator function which goes to infinity when $||z||_\infty > \lambda$ and 0 otherwise. ADMM minimizes (3.20) with respect to x and z in an alternating direction manner, and updates p accordingly. It guarantees global convergence to the optimal solution to (3.18). Starting from any initialization points of x, z, and p, ADMM iteratively solves (t = 0, 1, 2... denotes the iteration number):

$$\textit{x update:} \quad \min_{x_{t+1}} \tfrac{1}{2}||\mathbf{D}x - y||_2^2 - p_t^T x + \tfrac{\beta}{2}||z_t - x||_2^2, \tag{3.21}$$

$$\textit{z update:} \quad \min_{z_{t+1}} \tfrac{\beta}{2}||z - (x_{t+1} - \tfrac{p_t}{\beta})||_2^2 + \Phi_\lambda(z), \tag{3.22}$$

$$\textit{p update:} \quad p_{t+1} = p_t + \beta(z_{t+1} - x_{t+1}). \tag{3.23}$$

Furthermore, both (3.21) and (3.22) enjoy closed-form solutions:

$$x_{t+1} = (\mathbf{D}^T\mathbf{D} + \beta\mathbf{I})^{-1}(\mathbf{D}^T y + \beta z_t + p_t), \tag{3.24}$$

$$z_{t+1} = \min(\max(x_{t+1} - \tfrac{p_t}{\beta}, -\lambda), \lambda). \tag{3.25}$$

The above algorithm could be categorized to the primal-dual scheme. However, discussing the ADMM algorithm in more detail is beyond the focus of this section. Instead, the purpose of deriving (3.19)-(3.25) is to prepare us for the design of the task-specific deep architecture, as presented below.

3.4.3 Deep ℓ_∞ Encoder

We first substitute (3.24) into (3.25), in order to derive an update form explicitly dependent on only z and p:

$$z_{t+1} = B_\lambda((\mathbf{D}^T\mathbf{D} + \beta\mathbf{I})^{-1}(\mathbf{D}^T y + \beta z_t + p_t) - \tfrac{p_t}{\beta}), \tag{3.26}$$

where B_λ is defined as a box-constrained element-wise operator (u denotes a vector and u_i is its i-th element):

$$[B_\lambda(u)]_i = \min(\max(u_i, -\lambda), \lambda). \tag{3.27}$$

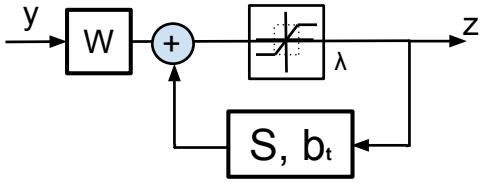


Figure 3.11: The block diagram of solving (3.18).

.

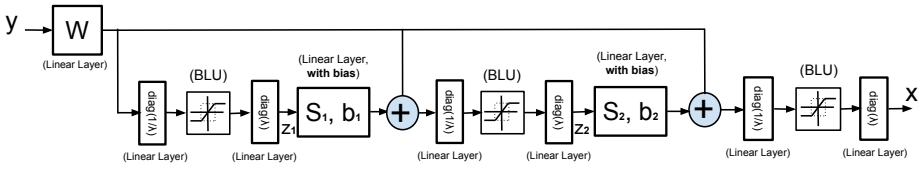


Figure 3.12: Deep ℓ_∞ Encoder, with two time-unfolded stages.

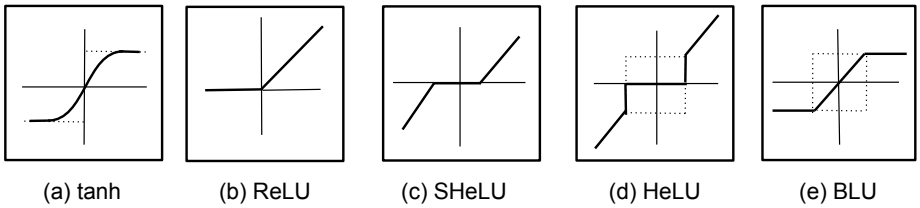


Figure 3.13: A comparison among existing neurons and BLU.

Eqn. (3.26) could be alternatively rewritten as:

$$\begin{aligned} &\mathbf{z}_{t+1} = B_\lambda(\mathbf{W}y + \mathbf{S}z_t + b_t), \text{ where:} \\ &\mathbf{W} = (\mathbf{D}^T\mathbf{D} + \beta\mathbf{I})^{-1}\mathbf{D}^T, \mathbf{S} = \beta(\mathbf{D}^T\mathbf{D} + \beta\mathbf{I})^{-1}, \\ &b_t = [(\mathbf{D}^T\mathbf{D} + \beta\mathbf{I})^{-1} - \tfrac{1}{\beta}\mathbf{I}]p_t, \end{aligned} \tag{3.28}$$

and expressed as the block diagram in Figure 3.11, which outlines a recurrent structure of solving (3.18). In (3.28), while $\mathbf{W}$ and $\mathbf{S}$ are pre-computed hyperparameters shared across iterations, b_t remains a variable dependent on p_t, and has to be updated throughout iterations too (b_t's update block is omitted in Figure 3.11).

By time-unfolding and truncating Figure 3.11 to a fixed number of K iterations (K = 2 by default), we obtain a feed-forward network structure in Figure 3.12, named **Deep ℓ_∞ Encoder**. Since the threshold λ is less straightforward to update, we repeat the same trick in [163] to rewrite (3.32) as: $[B_\lambda(u)]_i = \lambda_i B_1(u_i/\lambda_i)$. The original operator is thus decomposed into two linear diagonal scaling layers, plus a unit-threshold neuron, the latter of which is called a Bounded Linear Unit (**BLU**) by us. All the hyperparameters $\mathbf{W}$, $\mathbf{S}_k$ and b_k (k = 1, 2), as well as λ, are all to be learnt from data by back-propogation. Although the equations in (3.28) do not directly apply to solving the deep ℓ_∞ encoder, they can serve as high-quality initializations.

It is crucial to notice the modeling of the Lagrange multipliers p_t as the biases, and to incorporate its updates into network learning. That provides important clues on how to relate deep networks to a larger class of optimization models, whose solutions rely on dual domain methods.

Comparing BLU with existing neurons As shown in Figure 3.13 (e), BLU suppresses large entries while not penalizing small ones, resulting in dense, nearly antipodal representations. A first look at BLU easily reminds the tanh neuron (Figure 3.13 (a)). In fact, with its output range $[-1, 1]$ and a slope of 1 at the origin, tanh could be viewed as a smoothened differentiable approximation of BLU.

We further compare BLU with other popular and recently proposed neurons: Rectifier Linear Unit (ReLU) [89], Soft-tHresholding Linear Unit (SHeLU) [162], and Hard thrEsholding Linear Unit (HELU) [163], as depicted in Figure 3.13 (b)-(d), respectively. Contrary to BLU and tanh, they all introduce sparsity in the outputs, and thus prove successful and outperform tanh in classification and recognition tasks. Interestingly, HELU seems exactly the rival against BLU, as it does not penalize large entries but suppresses small ones down to zero.

3.4.4 Deep ℓ_∞ Siamese Network for Hashing

Rather than solving (3.18) first and then training the encoder as general regression, as [68] did, we concatenate encoder(s) with a task-driven loss, and optimize the pipeline **from end to end**. In this section, we focus on discussing its application in hashing, although the proposed model is not limited to one specific application.

Background With the ever-growing large-scale image data on the Web, much attention has been devoted to nearest neighbor search via hashing methods [64]. For big data applications, compact bitwise representations improve the efficiency in both storage and search speed. The state-of-the-art approach, learning-based hashing, learns similarity-preserving hash functions to encode input data into binary codes. Furthermore, while earlier methods, such as linear search hashing (LSH) [64], iterative quantization (ITQ) [67] and spectral hashing (SH) [168], do not refer to any supervised information, it has been lately discovered that involving the data similarities/dissimilarities in training benefits the performance [90, 104].

Prior Work Traditional hashing pipelines first represent each input image as a (hand-crafted) visual descriptor, followed by separate projection and quantization steps to encode it into a binary code. [116] first applied the *siamese network* [73] architecture to hashing, which fed two input patterns into two parameter-sharing "encoder" columns and minimized a pairwise-similarity/dissimilarity loss function between their outputs, using pairwise labels. The authors further enforced the sparsity prior on the hash codes in [115] by substituting a pair of LISTA-type encoders [68] for the pair of generic feed-forward encoders in [116, 172, 100], and utilized tailored convolution networks with the aid of pairwise labels. [91] further introduced a triplet loss with a divide-and-encode strategy applied to reduce the hash code redundancy. Note that for the final training step of quantization, [115] relied on an extra hidden layer of tanh neurons to approximate binary codes, while [91] exploited a piece-wise linear and discontinuous threshold function.

Our Approach In view of its robustness to quantization noise, as well as BLU's property as a natural binarization approximation, we construct a siamese network as in [116], and adopt a pair of parameter-sharing deep ℓ_∞ encoders as the two columns. The resulting architecture, named the **Deep ℓ_∞ Siamese Network**, is illustrated in Figure 3.14. Assume y and y^+ make a similar pair while y and y^- make a dissimilar pair, and further denote $x(y)$ the output representation by inputting y. The two coupled encoders are then optimized under the following

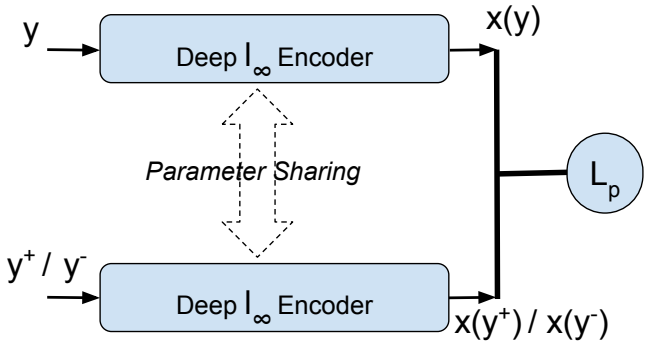


Figure 3.14: Deep ℓ_∞ Siamese Network, by coupling two parameter-sharing encoders, followed by a pairwise loss (3.29).

pairwise loss (the constant m represents the margin between dissimilar pairs):

$$L_p := \tfrac{1}{2}||x(y) - x(y^+)||^2 - \tfrac{1}{2}(\max(0, m - ||x(y) - x(y^-)||))^2. \tag{3.29}$$

The representation is learned to make similar pairs as close as possible and dissimilar pairs at least at distance m. In this section, we follow [116] to use a default m = 5 for all experiments.

Once a deep ℓ_∞ siamese network is learned, we apply its encoder part (i.e., a deep ℓ_∞ encoder) to a new input. The computation is extremely efficient, involving only a few matrix multiplications and element-wise thresholding operations, with a total complexity of $O(nN + 2N^2)$. One can obtain a N-bit binary code by simply quantizing the output.

3.4.5 Experiments in Image Hashing

Implementation The proposed networks are implemented with the CUDA ConvNet package [89]. We use a constant learning rate of 0.01 with no momentum, and a batch size of 128. Different from prior findings such as in [163, 162], we discover that untying the values of $\mathbf{S}_1$, b_1 and $\mathbf{S}_2$, b_2 boosts the performance more than sharing them. It is not only because that more free parameters enable a larger learning capacity, but also due to the important fact that p_t (and thus b_k) is in essence not shared across iterations, as in (3.28) and Figure 3.15.

While many neural networks are trained well with random initializations, sometimes poor initializations can still hamper the effectiveness of first-order methods

[153]. On the other hand, it is much easier to initialize our proposed models in the right regime. We first estimate the dictionary $\mathbf{D}$ using the standard K-SVD algorithm [3], and then inexactly solve (3.18) for up to K (K = 2) iterations, via the ADMM algorithm, with the values of Lagrange multiplier p_t recorded for each iteration. Benefiting from the analytical correspondence relationships in (3.28), it is then straightforward to obtain high-quality initializations for $\mathbf{W}$, $\mathbf{S}_k$ and b_k (k = 1, 2). As a result, we could achieve a steadily decreasing curve of training errors, without performing common tricks such as annealing the learning rate, which are found to be indispensable if random initialization is applied.

Datasets The **CIFAR10** dataset [88] contains 60K labeled images of 10 different classes. The images are represented using 384-dimensional GIST descriptors [125]. Following the classical setting in [115], we used a training set of 200 images for each class, and a disjoint query set of 100 images per class. The remaining 59K images are treated as database.

NUS-WIDE [38] is a dataset containing 270K annotated images from Flickr. Every images is associated with one or more of the different 81 concepts, and is described using a 500-dimensional bag-of-features. In training and evaluation, we followed the protocol of [105]: two images were considered as neighbors if they share at least one common concept (only 21 most frequent concepts are considered). We use 100K pairs of images for training, and a query set of 100 images per concept in testing.

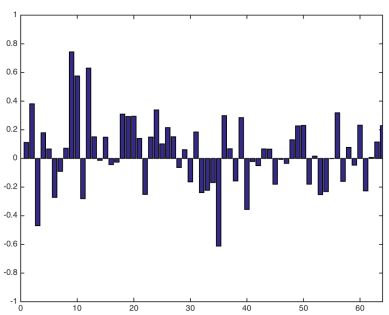
Comparison Methods We compare the proposed deep ℓ_∞ siamese network to six state-of-the-art hashing methods:

- four representative "shallow" hashing methods: kernelized supervised hashing (KSH) [104], anchor graph hashing (AGH) [105] (we compare with its two alternative forms: AGH1 and AGH2; see the original paper), parameter-sensitive hashing (PSH) [139], and LDA Hash (LH) [150] [13].

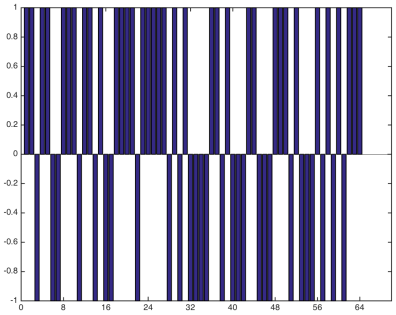
- two latest "deep" hashing methods: neural-network hashing (NNH) [116], and sparse neural-network hashing (SNNH) [115].

Comparing the two "deep" competitors to the deep ℓ_∞ siamese network, the only difference among the three is the type of encoder adopted in each's twin columns, as listed in Table 3.11. We re-implement the encoder parts of NNH and SNNH,

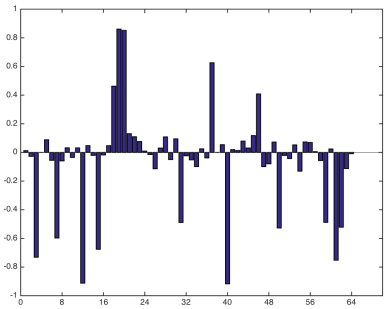
[13] Most of the results are collected from the comparison experiments in [115], under the same settings.



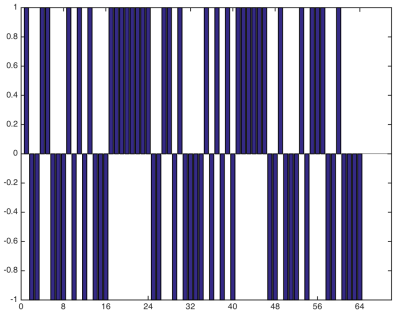
(a) NNH representation



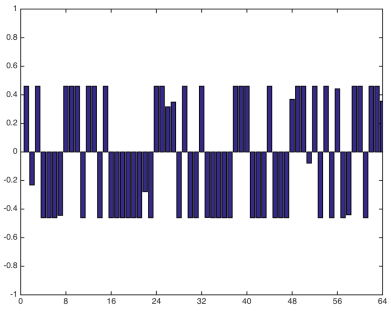
(b) NNH binary hashing code



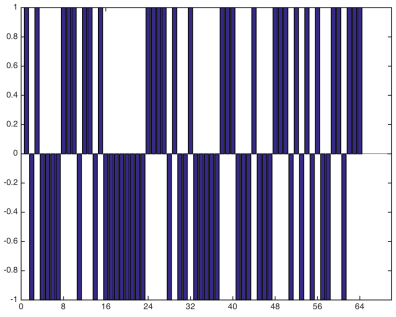
(c) SNNH representation



(d) SNNH binary hashing code



(e) Deep ℓ_∞ representation



(f) Deep ℓ_∞ binary hashing code

Figure 3.15: The learned representations and binary hashing codes of one test image from CIFAR10, through: (a) (b) NNH; (c) (d) SNNH; (e) (f) proposed.

with three hidden layers (i.e., two unfolded stages for LISTA), so that all three deep hashing models have the same depth.[14] Recall that the input $y \in R^n$ and the hash code $x \in R^N$, we immediately see from (3.28) that $\mathbf{W} \in R^{n \times N}$, $\mathbf{S}_k \in R^{N \times N}$, and $b_k \in R^N$. We carefully ensure that both NNHash and SparseHash have all their weight layers of the same dimensionality with ours,[15] for a fair comparison.

Table 3.11: Comparison of NNH, SNNH, and the proposed deep ℓ_∞ siamese network.

	encoder type	neuron type	structural prior on hashing codes
NNH	generic	tanh	/
SNNH	LISTA	SHeLU	sparse
Proposed	deep ℓ_∞	BLU	nearly antipodal & quantization-robust

Table 3.12: Performance (%) of different hashing methods on the CIFAR10 dataset, with different code lengths N.

Method		mAP	Hamming radius ≤ 2			Hamming radius $= 0$		
			Prec.	Recall	F1	Prec.	Recall	F1
	N							
KSH	48	31.10	18.22	0.86	1.64	5.39	5.6×10^{-2}	0.11
	64	32.49	10.86	0.13	0.26	2.49	9.6×10^{-3}	1.9×10^{-2}
AGH1	48	14.55	15.95	2.8×10^{-2}	5.6×10^{-2}	4.88	2.2×10^{-3}	4.4×10^{-3}
	64	14.22	6.50	4.1×10^{-3}	8.1×10^{-3}	3.06	1.2×10^{-3}	2.4×10^{-3}
AGH2	48	15.34	17.43	7.1×10^{-2}	3.6×10^{-2}	5.44	3.5×10^{-3}	6.9×10^{-3}
	64	14.99	7.63	7.2×10^{-3}	1.4×10^{-2}	3.61	1.4×10^{-3}	2.7×10^{-3}
PSH	48	15.78	9.92	6.6×10^{-3}	1.3×10^{-2}	0.30	5.1×10^{-5}	1.0×10^{-4}
	64	17.18	1.52	3.0×10^{-4}	6.1×10^{-4}	1.0×10^{-3}	1.69×10^{-5}	3.3×10^{-5}
LH	48	13.13	3.0×10^{-3}	1.0×10^{-4}	5.1×10^{-5}	1.0×10^{-3}	1.7×10^{-5}	3.4×10^{-5}
	64	13.07	1.0×10^{-3}	1.7×10^{-5}	3.3×10^{-5}	0.00	0.00	0.00
NNH	48	31.21	34.87	1.81	3.44	10.02	9.4×10^{-2}	0.19
	64	35.24	23.23	0.29	0.57	5.89	1.4×10^{-2}	2.8×10^{-2}
SNNH	48	26.67	32.03	12.10	17.56	19.95	**0.96**	**1.83**
	64	27.25	30.01	**36.68**	33.01	30.25	**9.8**	**14.90**
Proposed	48	**31.48**	**36.89**	**12.47**	**18.41**	**24.82**	0.94	1.82
	64	**36.76**	**38.67**	30.28	**33.96**	**33.30**	8.9	14.05

[14]The performance is thus better than reported in their original papers using two hidden layers, although with extra complexity.

[15]Both the deep ℓ_∞ encoder and the LISTA network will introduce the diagonal layers, while the generic feed-forward networks will not. Besides, neither LISTA nor generic feed-forward networks contain layer-wise biases. Yet since either a diagonal layer or a bias contains only N free parameters, the total amount is negligible.

Table 3.13: Performance (%) of different hashing methods on the NUS-WIDE dataset, with different code lengths N.

Method				Hamming radius ≤ 2			Hamming radius $= 0$		
		mAP@10	MP@5K	Prec.	Recall	F1	Prec.	Recall	F1
	N								
KSH	64	72.85	42.74	83.80	6.1×10^{-3}	1.2×10^{-2}	84.21	1.7×10^{-3}	3.3×10^{-3}
	256	73.73	45.35	84.24	1.4×10^{-3}	2.9×10^{-3}	84.24	1.4×10^{-3}	2.9×10^{-3}
AGH1	64	69.48	47.28	69.43	0.11	0.22	73.35	3.9×10^{-2}	7.9×10^{-2}
	256	73.86	46.68	75.90	1.5×10^{-2}	2.9×10^{-2}	81.64	3.6×10^{-3}	7.1×10^{-3}
AGH2	64	68.90	47.27	68.73	0.14	0.28	72.82	5.2×10^{-2}	0.10
	256	73.00	47.65	74.90	5.3×10^{-2}	0.11	80.45	1.1×10^{-2}	2.2×10^{-2}
PSH	64	72.17	44.79	60.06	0.12	0.24	81.73	1.1×10^{-2}	2.2×10^{-2}
	256	73.52	47.13	84.18	1.8×10^{-3}	3.5×10^{-3}	84.24	1.5×10^{-3}	2.9×10^{-3}
LH	64	71.33	41.69	**84.26**	1.4×10^{-3}	2.9×10^{-3}	84.24	1.4×10^{-3}	2.9×10^{-3}
	256	70.73	39.02	84.24	1.4×10^{-3}	2.9×10^{-3}	84.24	1.4×10^{-3}	2.9×10^{-3}
NNH	64	76.39	59.76	75.51	1.59	3.11	81.24	0.10	0.20
	256	78.31	61.21	83.46	5.8×10^{-2}	0.11	83.94	4.9×10^{-3}	9.8×10^{-3}
SNNH	64	74.87	56.82	72.32	**1.99**	**3.87**	81.98	**0.37**	**0.73**
	256	74.73	59.37	80.98	**0.10**	**0.19**	82.85	**0.98**	**1.94**
Proposed	64	**79.89**	**63.04**	79.95	1.72	3.38	**86.23**	0.30	0.60
	256	**80.02**	**65.62**	**84.63**	7.2×10^{-2}	0.15	**89.49**	0.57	1.13

We adopt the following classical criteria for evaluation: 1) *precision and recall* (PR) for different Hamming radii, and the *F1 score* as their harmonic average; 2) *mean average precision* (MAP) [119]. Besides, for NUS-WIDE, as computing mAP is slow over this large dataset, we follow the convention of [115] to compute the *mean precision* (MP) of top-5K returned neighbors (MP@5K), as well as report mAP of top-10 results (mAP@10).

We have not compared with convolutional network-based hashing methods [172, 100, 91], since it is difficult to ensure their models to have the same parameter capacity as our fully-connected model in controlled experiments. We also do not include triplet loss-based methods, e.g., [91], into comparison because they will require three parallel encoder columns.

Results and Analysis The performances of different methods on two datasets are compared in Tables 3.12 and 3.13. Our proposed method ranks top in almost all cases, in terms of mAP/MP and precision. Even under the Hamming radius of 0, our precision is as high as 33.30% (N = 64) for CIFAR10, and 89.49% (N = 256) for NUS-WIDE. The proposed method also maintains the second best in most cases, in terms of recall, inferior only to SNNH. In particular, when the hashing code dimensionality is low, e.g., when N = 48 for CIFAR10, the proposed method outperforms all else with a significant margin. It demonstrates the competitiveness of the proposed method in generating both compact and accurate hashing codes, that achieves more precise retrieval results at lower computation and storage costs.

The next observation is that, compared to the strongest competitor SNNH, the recall rates of our method seem less compelling. We plot the precision and recall curves of the three best performers (NNH, SNNH, deep l_∞), with regard to the bit length of hashing codes N, within the Hamming radius of 2. Figure 3.16 demonstrates that our method consistently outperforms both SNNH and NNH in precision. On the other hand, SNNH gains advantages in recall over the proposed method, although the margin appears vanishing as N grows.

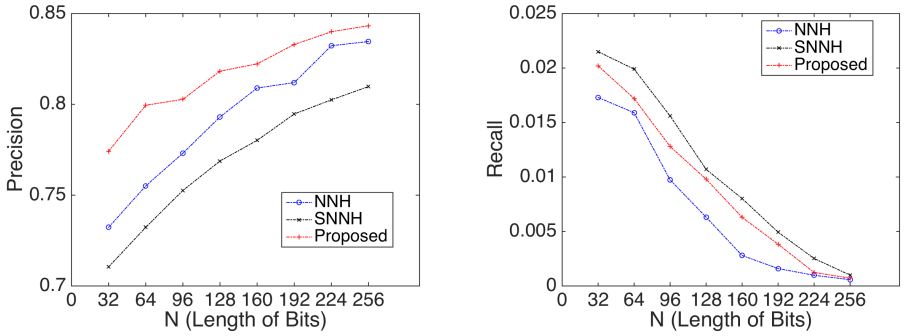


Figure 3.16: The comparison of three deep hashing methods on NUS-WIDE: (a) precision curve; (b) recall curve, both w.r.t the hashing code length N, within the Hamming radius of 2.

Although it seems to be a reasonable performance tradeoff, we are curious about the behavior difference between SNNH and the proposed method. We are again reminded that they only differ in the encoder architecture, i.e., one with LISTA while the other uses the deep l_∞ encoder. We thus plot the learned representations and binary hashing codes of one CIFAR image, using NNH, SNNH, and the proposed method, in Figure 3.15. By comparing the three pairs, one could see that the quantization from (a) to (b) (also (c) to (d)) suffers visible distortion and information loss. Contrary to them, the output of the deep l_∞ encoder has a much smaller quantization error, as it naturally resembles an antipodal signal. Therefore, it suffers minimal information loss during the quantization step.

In view of those, we conclude the following points towards the different behaviors, between SNNH and deep l_∞ encoder:

- Both deep l_∞ encoder and SNNH outperform NNH, by introducing structure into the binary hashing codes.
- The deep l_∞ encoder generates nearly antipodal outputs that are robust to

quantization errors. Therefore, it excels in preserving information against hierarchical information extraction as well as quantization. That explains why our method reaches the highest precision, and performs especially well when N is small.

- SNNH exploits sparsity as a prior on hashing codes. It confines and shrinks the solution space, as many small entries in the SNNH outputs will be suppressed down to zero. That is also evidenced by Table 2 in [115], i.e., the number of unique hashing codes in SNNH results is one order smaller than that of NNH.

- The sparsity prior improves the recall rate, since it obtains hashing codes that clutter more compactly in the high-dimensional space, with lower intra-cluster variations. But it also runs the risk of losing too much information, during the hierarchical sparsifying process. In that case, the inter-cluster variations might also be compromised, which potentially causes the decrease in precision.

Further, it seems that the sparsity and l_∞ structure priors could be complementary. We will explore it as future work.

3.4.6 Conclusion

This section investigates how to import the quantization-robust property of an ℓ_∞-constrained minimization model, to a specially-designed deep models. It is done by first deriving an ADMM algorithm, which is then re-formulated as a feed-forward neural network. We introduce the siamese architecture concatenated with a pairwise loss, for the application purpose of hashing. We analyze in depth the performance and behaviors of the proposed model against its competitors, and hope it will evoke more interests from the community.

3.5 Deep Dual-Domain Sparse Coding for Image Compression Artifact Removal

In this section, we design a Deep Dual-Domain ($\mathbf{D^3}$) based fast restoration model to remove artifacts of JPEG compressed images. It leverages the large learning ca-

pacity of deep networks, as well as the problem-specific expertise that was hardly incorporated in the past design of deep architectures. For the latter, we take into consideration both the prior knowledge of the JPEG compression scheme, and the successful practice of the sparsity-based dual-domain approach. We further design the One-Step Sparse Inference (1-SI) module, as an efficient and lightweight feed-forward approximation of sparse coding. Extensive experiments verify the superiority of the proposed D^3 model over several state-of-the-art methods. Specifically, our best model is capable of outperforming the latest deep model for around 1 dB in PSNR, and is 30 times faster.

3.5.1 Introduction

In visual communication and computing systems, the most common cause of image degradation is arguably compression. Lossy compression, such as JPEG [127] and HEVC-MSP [5], is widely adopted in image and video codecs for saving both bandwidth and in-device storage. It exploits inexact approximations for representing the encoded content compactly. Inevitably, it will introduce undesired complex artifacts, such as blockiness, ringing effects, and blurs. They are usually caused by the discontinuities arising from batch-wise processing, the loss of high-frequency components by coarse quantization, and so on. These artifacts not only degrade perceptual visual quality, but also adversely affect various low-level image processing routines that take compressed images as input [52].

As practical image compression methods are not information theoretically optimal [107], the resulting compression code streams still possess residual redundancies, which makes the restoration of the original signals possible. Different from general image restoration problems, compression artifact restoration has problem-specific properties that can be utilized as powerful priors. For example, JPEG compression first divides an image into 8×8 pixel blocks, followed by discrete cosine transformation (DCT) on every block. Quantization is applied on the DCT coefficients of every block, with pre-known quantization levels [127]. Moreover, the compression noises are more difficult to model than other common noise types. In contrast to the tradition of assuming noise to be white and signal independent [3], the non-linearity of quantization operations makes quantization noises non-stationary and signal-dependent.

Various approaches have been proposed to suppress compression artifacts. Early

works [20, 97] utilized filtering-based methods to remove simple artifacts. Data-driven methods were then considered to avoid inaccurate empirical modeling of compression degradations. Sparsity-based image restoration approaches have been discussed in [28, 37, 81, 106, 134] to produce sharpened images, but they are often accompanied with artifacts along edges, and unnatural smooth regions. In [107], Liu et al. proposed a sparse coding process carried out jointly in the DCT and pixel domains, to simultaneously exploit residual redundancies of JPEG codes and sparsity properties of latent images. Dong et al. [52] first introduced deep learning techniques [89] into this problem, by specifically adapting their SR-CNN model [53]. However, it does not incorporate problem-specific prior knowledge.

The time constraint is often stringent in image or video codec post-processing scenarios. Low-complexity or even real-time attenuation of compression artifacts is highly desirable [142]. The inference process of traditional approaches, for example, sparse coding, usually involves iterative optimization algorithms, whose inherently sequential structure as well as the data-dependent complexity and latency often constitute a major bottleneck in the computational efficiency [68]. Deep networks benefit from the feed-forward structure and enjoy much faster inference. However, to maintain their competitive performances, deep networks show demands for increased width (numbers of filters) and depth (number of layers), as well as smaller strides, all leading to growing computational costs [74].

In this section, we focus on removing artifacts in JPEG compressed images. Our major innovation is to explicitly combine both **the prior knowledge in the JPEG compression scheme** and **the successful practice of dual-domain sparse coding** [107], for designing a task-specific deep architecture. Furthermore, we introduce a One-Step Sparse Inference **(1-SI)** module, that acts as a highly efficient and lightweight approximation of the sparse coding inference [49]. 1-SI also reveals important inner connections between sparse coding and deep learning. The proposed model, named Deep Dual-Domain ($\mathbf{D^3}$) based fast restoration, proves to be more effective and interpretable than general deep models. It gains remarkable margins over several state-of-the-art methods, in terms of both **restoration performance** and **time efficiency**.

3.5.2 Related Work

Our work is inspired by the prior wisdom in [107]. Most previous works restored compressed images in either the pixel domain [3] or the DCT domain [127] solely. However, an isolated quantization error of one single DCT coefficient is propagated to all pixels of the same block. An aggressively quantized DCT coefficient can further produce structured errors in the pixel-domain that correlate to the latent signal. On the other hand, the compression process sets most high frequency coefficients to zero, making it impossible to recover details from only the DCT domain. In view of their complementary characteristics, the dual-domain model was proposed in [107]. While the spatial redundancies in the pixel domain were exploited by a learned dictionary [3], the residual redundancies in the DCT domain were also utilized to directly restore DCT coefficients. In this way, quantization noises were suppressed without propagating errors. The final objective (see Section 3.1) is a combination of DCT- and pixel-domain sparse representations, which could cross validate each other.

To date, deep learning [89] has shown impressive results on both high-level and low-level vision problems [166]. The SR-CNN proposed by Dong et al. [53] showed the great potential of end-to-end trained networks in image super resolution (SR). Their recent work [52] proposed a four-layer convolutional network that was tuned based on SR-CNN, named Artifacts Reduction Convolutional Neural Networks (AR-CNN), which was effective in removing compression artifacts.

3.5.3 Deep Dual-Domain ($\mathbf{D^3}$) based Restoration

Sparsity-based Dual-Domain Formulation

We first review the sparsity-based dual-domain restoration model established in [107]. Considering a training set of **uncompressed** images, pixel-domain blocks $\{\hat{x}_i\} \in R^m$ (vectorized from a $\sqrt{m} \times \sqrt{m}$ patch; m = 64 for JPEG) are drawn for training, along with their **quantized** DCT coefficient blocks $\{y_i\} \in R^m$. For each (JPEG-coded) input $x_t \in R^m$, two dictionaries $\mathbf{\Phi} \in R^{m \times p_\Phi}$ and $\mathbf{\Psi} \in R^{m \times p_\Psi}$ (p_Φ and p_Ψ denote the dictionary sizes) are constructed from training data $\{y_i\}$ and $\{\hat{x}_i\}$, in the DCT and pixel domains, respectively, via locally adaptive feature selection and projection. The following optimization model is then solved during

the testing stage:

$$\begin{aligned} \min_{\{\alpha,\beta\}} &||y_t - \mathbf{\Phi}\alpha||_2^2 + \lambda_1||\alpha||_1 + \lambda_2||T^{-1}\mathbf{\Phi}\alpha - \mathbf{\Psi}\beta||_2^2 + \lambda_3||\beta||_1, \\ s.t. \quad & q^L \preceq \mathbf{\Phi}\alpha \preceq q^U. \end{aligned} \tag{3.30}$$

where $y_t \in R^m$ is the DCT coefficient block for x_t. $\alpha \in R^{p_\Phi}$ and $\beta \in R^{p_\Psi}$ are sparse codes in the DCT and pixel domains, respectively. T^{-1} denotes the inverse discrete cosine transform (IDCT) operator. λ_1, λ_2 and λ_3 are positive scalars. One noteworthy point is the inequality constraint, where q^L and q^U represent the (pre-known) quantization intervals according to the JPEG quantization table [127]. The constraint incorporates the important side information and further confines the solution space. Finally, $\mathbf{\Psi}\beta$ provides an estimate of the original uncompressed pixel block $\hat{x}_t$.

Such a sparsity-based dual-domain model (3.30) exploits residual redundancies (e.g. inter-DCT-block correlations) in the DCT domain without spreading errors into the pixel domain, and at the same time recovers high-frequency information driven by a large training set. However, note that the inference process of (3.30) relies on iterative algorithms, and is computationally expensive. Also in (3.30), the three parameters λ_1, λ_2 and λ_3 have to be manually tuned. The authors of [107] simply set them all equal, which may hamper the performance. In addition, the dictionaries $\mathbf{\Phi}$ and $\mathbf{\Psi}$ have to be individually learned for each patch, which allows for extra flexibility but also brings in heavy computation load.

$\mathbf{D^3}$: A Feed-Forward Network Formulation

In training, we have the compressed pixel-domain blocks $\{x_i\}$, accompanied with the original uncompressed blocks $\{\hat{x}_i\}$. During testing, for an compressed input x_t, our goal is to estimate the original $\hat{x}_t$, using the redundancies in both DCT and pixel domains, as well as JPEG prior knowledge.

As illustrated in Figure 3.17, the input x_t is first transformed into its DCT coefficient block y_t by feeding through the constant 2-D DCT matrix layer T. The subsequent two layers aim to enforce DCT domain sparsity, where we refer to the concepts of analysis and synthesis dictionaries in sparse coding [70]. The Sparse Coding (SC) Analysis Module 1 is implemented to solve the following type of

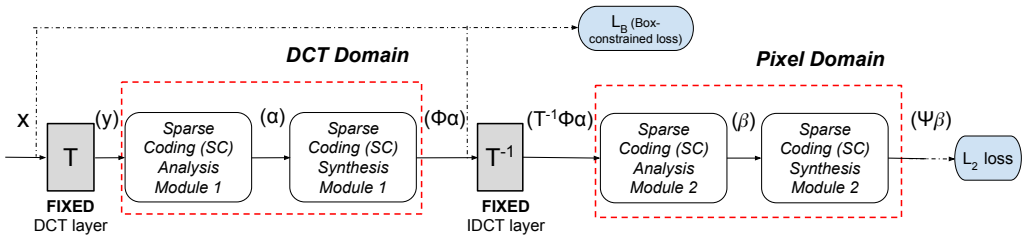


Figure 3.17: The illustration of Deep Dual-Domain ($\mathbf{D^3}$) based model (all subscripts are omitted for simplicity). The black solid lines denote the network inter-layer connections, while the black dash lines connect to the loss functions. The two red dash-line boxes depict the two stages that incorporate DCT and pixel domain sparsity priors, respectively. The two grey blocks denote constant DCT and IDCT layers, respectively. The notations within parentheses along the pipeline are to remind the corresponding variables in (3.30).

sparse inference problem in the DCT domain (λ is a positive coefficient):

$$\min_\alpha \tfrac{1}{2}||y_t - \mathbf{\Phi}\alpha||_2^2 + \lambda||\alpha||_1. \tag{3.31}$$

The Sparse Coding (SC) Synthesis Module 1 outputs the DCT-domain sparsity-based reconstruction in (3.30), i.e., $\mathbf{\Phi}\alpha$.

The intermediate output $\mathbf{\Phi}\alpha$ is further constrained by an auxiliary loss, which encodes the inequality constraint in (3.30): $q^L \preceq \mathbf{\Phi}\alpha \preceq q^U$. We design the following **signal-dependent, box-constrained [86] loss**:

$$L_B(\mathbf{\Phi}\alpha, x) = ||[\mathbf{\Phi}\alpha - q^U(x)]_+||_2^2 + ||[q^L(x) - \mathbf{\Phi}\alpha]_+||_2^2. \tag{3.32}$$

Note it takes not only $\mathbf{\Phi}\alpha$, but also x as inputs, since the actual JPEG quantization interval $[q^L, q^U]$ depends on x. The operator $[\quad]_+$ keeps the nonnegative elements unchanged while setting others to zero. Eqn. (3.32) will thus only penalize the coefficients falling out of the quantization interval.

After the constant IDCT matrix layer T^{-1}, the DCT-domain reconstruction $\mathbf{\Phi}\alpha$ is transformed back to the pixel domain for one more sparse representation. The SC Analysis Module 2 solves (γ is a positive coefficient):

$$\min_\beta \tfrac{1}{2}||T^{-1}\mathbf{\Phi}\alpha - \mathbf{\Psi}\beta||_2^2 + \gamma||\beta||_1, \tag{3.33}$$

while the SC Synthesis Module 2 produces the final pixel-domain reconstruction $\mathbf{\Psi}\beta$. Finally, the L_2 loss between $\mathbf{\Psi}\beta$ and $\hat{x}_i$ is enforced.

Note that in the above, we try to correspond the intermediate outputs of $\mathbf{D^3}$ with the variables in (3.30), in order to help understand the close analytical relationship between the proposed deep architecture with the sparse coding-based model. That does not necessarily imply any exact numerical equivalence, since $\mathbf{D^3}$ allows for end-to-end learning of all parameters (including λ in (3.31) and γ in (3.33)). However, we will see in experiments that such enforcement of the specific problem structure improves the network performance and efficiency remarkably. In addition, the above relationships remind us that the deep model could be well initialized from the sparse coding components.

One-Step Sparse Inference Module

The implementation of SC Analysis and Synthesis Modules appears to be the core of $\mathbf{D^3}$. While the synthesis process is naturally feed-forward by multiplying the dictionary, it is less straightforward to transform the sparse analysis (or inference) process into a feed-forward network.

We take (3.31) as an example, while the same solution applies to (3.33). Such a sparse inference problem could be solved by the iterative shrinkage and thresholding algorithm (ISTA) [17], each iteration of which updates as follows:

$$\boldsymbol{\alpha}^{k+1} = s_\lambda(\boldsymbol{\alpha}^k + \boldsymbol{\Phi}^T(y_t - \boldsymbol{\Phi}\boldsymbol{\alpha}^k)), \tag{3.34}$$

where $\boldsymbol{\alpha}^k$ denotes the intermediate result of the k-th iteration, and where s_λ is an element-wise shrinkage function ($\mathbf{u}$ is a vector and $\mathbf{u}_i$ is its i-th element, $i = 1, 2, ..., p$):

$$[s_\lambda(\mathbf{u})]_i = \text{sign}(\mathbf{u}_i)[|\mathbf{u}_i| - \lambda_i]_+. \tag{3.35}$$

The learned ISTA (LISTA) [68] parameterized encoder further proposed a natural network implementation of ISTA. The authors time-unfolded and truncated (3.34) into a fixed number of stages (more than 2), and then jointly tuned all parameters with training data, for a good feed-forward approximation of sparse inference.

In our work, we launch a more aggressive approximation, by only keeping one iteration of (3.34), leading to a One-Step Sparse Inference (**1-SI**) Module. Our major motivation lies in the same observation as in [52] that overly deep networks could adversely affect the performance in low-level vision tasks. Note that we have two SC Analysis modules where the original LISTA applies, and two more

SC Synthesis modules (each with one learnable layer). Even if only two iterations are kept as in [68], we end up with a six-layer network that suffers from both difficulties in training [52] and fragility in generalization [148] for this task.

A 1-SI module takes the following simplest form:

$$\boldsymbol{\alpha} = s_{\lambda}(\boldsymbol{\Phi} y_t), \tag{3.36}$$

which could be viewed as first passing through a fully-connected layer ($\boldsymbol{\Phi}$), followed by neurons of s_{λ}. We further rewrite (3.35) as [166] did[16]:

$$[s_{\lambda}(\mathbf{u})]_i = \lambda_i \cdot \text{sign}(\mathbf{u}_i)(|\mathbf{u}_i|/\lambda_i - 1)_+ = \lambda_i s_1(\mathbf{u}_i/\lambda_i) \tag{3.37}$$

Eqn. (3.37) indicates that the original neuron with trainable thresholds can be decomposed into two linear scaling layers plus a unit-threshold neuron. The weights of the two scaling layers are diagonal matrices defined by $\boldsymbol{\theta}$ and its element-wise reciprocal, respectively. The unit-threshold neuron s_1 could in essence be viewed as a double-sided and translated variant of ReLU [89].

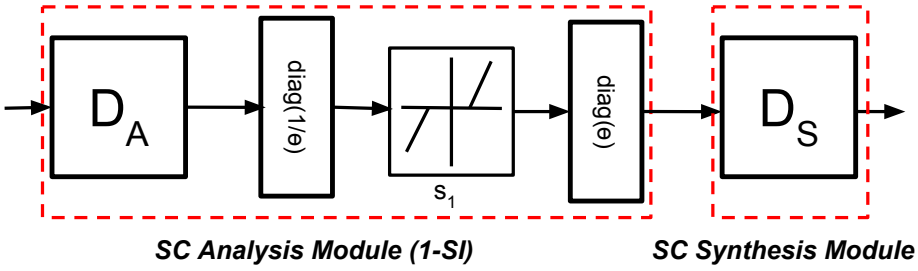


Figure 3.18: The illustration of SC Analysis and Synthesis Modules. The former is implemented by the proposed 1-SI module (3.36). Both D_A and D_S are fully-connected layers, while diag(θ)/diag($1/\theta$) denotes diagonal scaling layers.

A related form to (3.36) was obtained in [49] on a different case of non-negative sparse coding. The authors studied its connections with the soft-threshold feature for classification, but did not correlate it with network architectures.

Model Overview

By plugging in the 1-SI module (3.36), we are ready to obtain the SC Analysis and Synthesis Modules, as in Figure 3.18. By comparing Figure 3.18 with Eqn. (3.31)

[16] In (3.37), we slightly abuse notations, and set λ to be a vector of the same dimension as $\mathbf{u}$, in order for extra element-wise flexibility.

(or (3.33)), it is easy to notice the analytical relationships between D_A and $\mathbf{\Phi}^T$ (or $\mathbf{\Psi}^T$), D_S and $\mathbf{\Phi}$ (or $\mathbf{\Psi}$), as well as θ and λ (or γ). Those network parameters could be well initialized from the sparse coding parameters, which could be obtained easily. The entire D^3 model, consisting of four learnable fully-connected weight layers (except for the diagonal layers), is trained from end to end.[17]

In Figure 3.18, we intentionally do not combine θ into $\mathbf{D}_A$ layer (also $1/\theta$ into $\mathbf{D}_S$ layer), for the reason that we still wish to keep θ and $1/\theta$ layers tied as element-wise reciprocal. That proves to have positive implications in our experiments. If we absorb the two diagonal layers into $\mathbf{D}_A$ and $\mathbf{D}_S$, Figure 3.18 is reduced to two fully connected weight matrices, concatenated by one layer of hidden neurons (3.37). However, keeping the "decomposed" model architecture facilitates the incorporation of problem-specific structures.

Complexity Analysis

From the clear correspondences between the sparsity-based formulation and the D^3 model, we immediately derive the dimensions of weight layers in Table 3.14.

Table 3.14: Dimensions of all layers in the D^3 model

Layer	$\mathbf{D}_A$	$\mathbf{D}_S$	diag(θ)
Stage I (DCT Domain)	$p_\Phi \times m$	$m \times p_\Phi$	p_Φ
Stage II (Pixel Domain)	$p_\Psi \times m$	$m \times p_\Psi$	p_Ψ

Time Complexity During training, deep learning with the aid of gradient descent scales linearly in time and space with the number of training samples. We are primarily concerned with the time complexity during testing (inference), which is more relevant to practical usages. Since all learnable layers in the D^3 model are fully-connected, the inference process of D^3 is nothing more than a series of matrix multiplications. The multiplication times are counted as: $p_\Phi m$ (D_A in Stage I) + $2p_\Phi$ (two diagonal layers) + $p_\Phi m$ (D_S in Stage I) + $p_\Psi m$ (D_A in Stage II) + $2p_\Psi$ (two diagonal layers) + $p_\Psi m$ (D_S in Stage II). The 2D DCT and IDCT each takes $\frac{1}{2}m\log(m)$ multiplications [127] . Therefore, the total inference time complexity

[17] From the analytical perspective, $\mathbf{D}_S$ is the transpose of $\mathbf{D}_A$, but we untie them during training for larger learning capability.

of D^3 is:

$$C_{D^3} = 2(p_\Phi + p_\Psi)(m+1) + m\log(m) \approx 2m(p_\Phi + p_\Psi). \tag{3.38}$$

The complexity could also be expressed as $O(p_\Phi + p_\Psi)$.

It is obvious that the sparse coding inference [107] has dramatically higher time complexity. We are also interested in the inference time complexity of other competitive deep models, especially AR-CNN [52]. For their fully convolutional architecture, the total complexity [74] is:

$$C_{conv} = \sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2, \tag{3.39}$$

where l is the layer index, d is the total depth, n_l is the number of filters in the l-th layer, s_l is the spatial size of the filter, and m_l is the spatial size of the output feature map.

The theoretical time complexities in (3.38) and (3.39) do not represent the actual running time, as they depend on different configurations and can be sensitive to implementations and hardware. Yet, our actual running time scales nicely with those theoretical results.

Parameter Complexity The total number of free parameters in D^3 is:

$$N_{D^3} = 2p_\Phi m + p_\Phi + 2p_\Psi m + p_\Psi = 2(p_\Phi + p_\Psi)(m+1). \tag{3.40}$$

As a comparison, the AR-CNN model [52] contains:

$$N_{conv} = \sum_{l=1}^{d} n_{l-1} \cdot n_l \cdot s_l^2. \tag{3.41}$$

3.5.4 Experiments

Implementation and Setting

We use the disjoint training set (200 images) and test set (200 images) of BSDS500 database [4], as our training set; its validation set (100 images) is used for validation, which follows [52]. For training the D^3 model, we first divide each original image into overlapped 8×8 patches, and subtract the pixel values by 128 as in the

JPEG mean shifting process. We then perform JPEG encoding on them by MATLAB JPEG encoder with a specific quality factor Q, to generate the corresponding compressed samples. Whereas JPEG works on non-overlapping patches, we emphasize that the training patches are overlapped and extracted from arbitrary positions. For a testing image, we sample 8×8 blocks with a stride of 4, and apply the D^3 model in a patch-wise manner. For a patch that misaligns with the original JPEG block boundaries, we find its most similar coding block from its 16×16 local neighborhood, whose quantization intervals are then applied to the misaligned patch. We find this practice effective and important for removing blocking artifacts and ensuring the neighborhood consistency. The final result is obtained via aggregating all patches, with the overlapping regions averaged.

The proposed networks are implemented using the cuda-convnet package [89]. We apply a constant learning rate of 0.01, a batch size of 128, with no momentum. Experiments run on a workstation with 12 Intel Xeon 2.67GHz CPUs and 1 GTX680 GPU. The two losses, L_B and L_2, are equally weighted. For the parameters in Table 3.14, m is fixed as 64. We try different values of p_Φ and p_Ψ in experiments.

Based on the solved Eqn. (3.30), one could initialize D_A, D_S, and θ from Φ, Φ^T and λ in the DCT domain block of Figure 3.17, and from Ψ, Ψ^T and γ in the pixel domain block, respectively. In practice, we find that such an initialization strategy benefits the performances, and usually leads to faster convergence.

We test the quality factor Q = 5, 10, and 20. For each Q, we train a dedicated model. We further find the easy-hard transfer useful. As images of low Q values (heavily compressed) contain more complex artifacts, it is helpful to use the features learned from images of high Q values (lightly compressed) as a starting point. In practice, we first train the D^3 model on JPEG compressed images with $Q = 20$ (the highest quality). We then initialize the $Q = 10$ model with the $Q = 20$ model, and similarly, initialize $Q = 5$ model from the $Q = 10$ one.

Restoration Performance Comparison

We include the following two relevant, state-of-the-art methods for comparison:

- **Sparsity-based Dual-Domain Method (S-D^2)** [107] could be viewed as the "shallow" counterpart of D^3. It has outperformed most traditional methods [107], such as BM3D [44] and DicTV [28], with which we thus do not

compare again. The algorithm has a few parameters to be manually tuned. Especially, their dictionary atoms are adaptively selected by a nearest-neighbor type algorithm; the number of selected atoms varies for every testing patch. Therefore, the parameter complexity of S-D^2 cannot be exactly computed.

- **AR-CNN** has been the latest deep model resolving the JPEG compression artifact removal problem. In [52], the authors show its advantage over SA-DCT [62], RTF [79], and SR-CNN [53]. We adopt the default network configuration in [52]: $s_1 = 9$, $s_2 = 7$, $s_3 = 1$, $s_4 = 5$; $n_1 = 64$, $n_2 = 32$, $n_3 = 16$, $n_4 = 1$. The authors adopted the easy-hard transfer in training.

For D^3, we test $p_\Phi = p_\Psi = 128$ and 256 [18]. The resulting D^3 models are denoted as D^3-128 and D^3-256, respectively. In addition, to verify the superiority of our task-specific design, we construct a fully-connected Deep Baseline Model (D-Base), of the same complexity with D^3-256, named D-Base-256. It consists of four weight matrices of the same dimensions as D^3-256's four trainable layers.[19] D-Base-256 utilizes ReLU [89] neurons and the dropout technique.

We use the 29 images in the LIVE1 dataset [141] (converted to the gray scale) to evaluate both the quantitative and qualitative performances. Three quality assessment criteria are evaluated: PSNR, structural similarity (SSIM) [167], and PSNR-B [179], the last of which is designed specifically to assess blocky images. The averaged results on the LIVE1 dataset are list in Table 3.15.

Compared to S-D^2, both D^3-128 and D^3-256 gain remarkable advantages, thanks to the end-to-end training as deep architectures. As p_Φ and p_Ψ grow from 128 to 256, one observes clear improvements in PSNR/SSIM/PSNR-B. D^3-256 has outperformed the state-of-the-art ARCNN, for around 1 dB in PSNR. Moreover, D^3-256 also demonstrates a notable performance margin over D-Base-256, although they possess the same number of parameters. D^3 is thus verified to benefit from its task-specific architecture inspired by the sparse coding process (3.30), rather than just the large learning capacity of generic deep models. The parameter numbers of different models are compared in the last row of Table 3.15. It is impressive to see that D^3-256 also takes less parameters than AR-CNN.

We display three groups of visual results, on *Bike*, *Monarch* and *Parrots* images, when $Q = 5$, in Figures 3.19, 3.20 and 3.21, respectively. AR-CNN tends to

[18] From the common experiences of choosing dictionary sizes [3]

[19] D-Base-256 is a four-layer neural network, performed on the pixel domain, without DCT/IDCT layers. The diagonal layers contain a very small portion of parameters and are ignored here.

Table 3.15: The average results of PSNR (dB), SSIM, PSNR-B (dB) on the LIVE1 dataset.

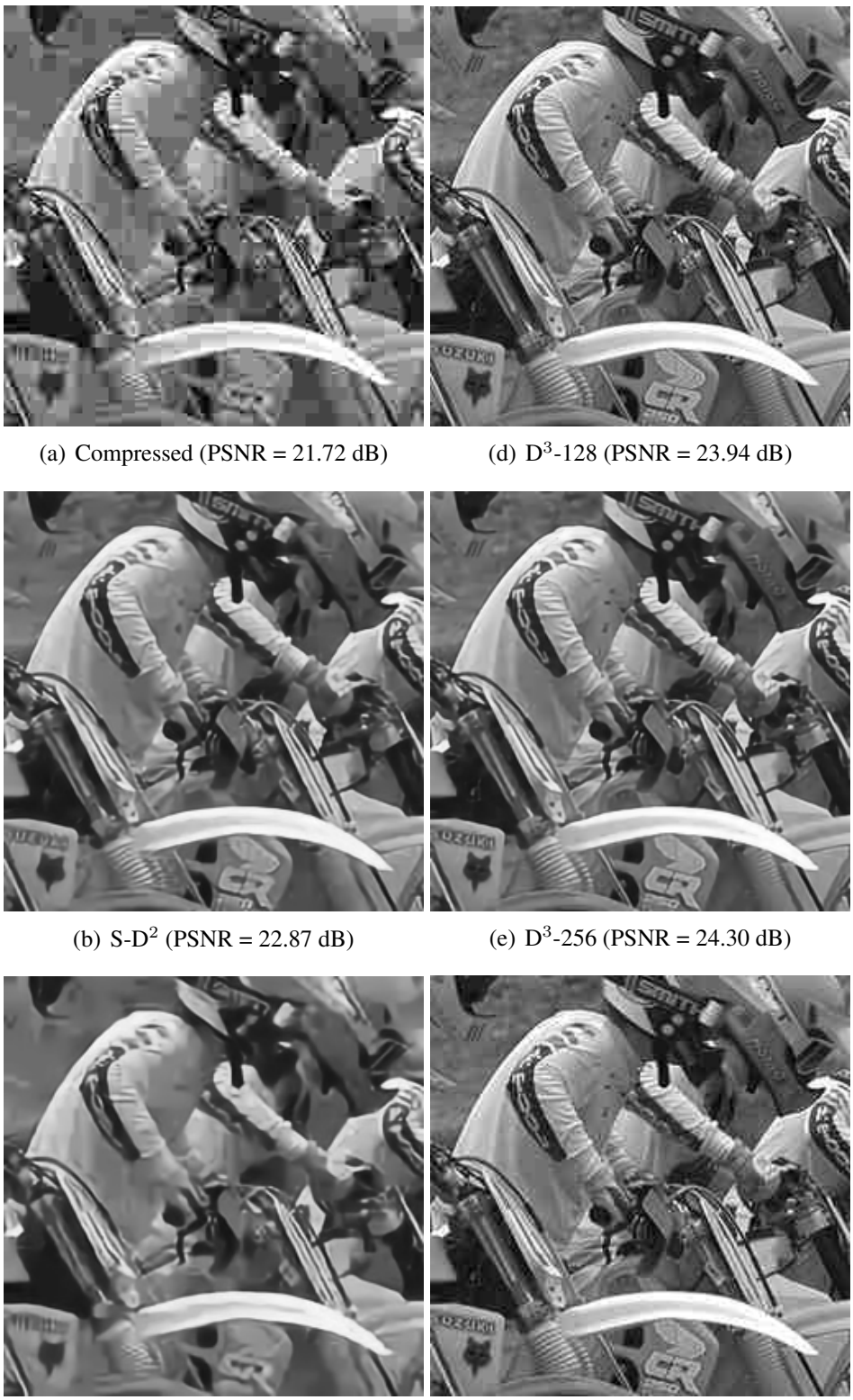
		Compressed	S-D^2	AR-CNN	D^3-128	D^3-256	D-Base-256
Q = 5	PSNR	24.61	25.83	26.64	26.26	**27.37**	25.83
	SSIM	0.7020	0.7170	0.7274	0.7203	**0.7303**	0.7186
	PSNR-B	22.01	25.64	26.46	25.86	**26.95**	25.51
Q = 10	PSNR	27.77	28.88	29.03	28.62	**29.96**	28.24
	SSIM	0.7905	0.8195	0.8218	0.8198	**0.8233**	0.8161
	PSNR-B	25.33	27.96	28.76	28.33	**29.45**	27.57
Q = 20	PSNR	30.07	31.62	31.30	31.20	**32.21**	31.27
	SSIM	0.8683	0.8830	0.8871	0.8829	**0.8903**	0.8868
	PSNR-B	27.57	29.73	30.80	30.56	**31.35**	29.25
#Param		\	NA	106,448	33, 280	66, 560	66, 560

generate over-smoothness, such as in the edge regions of butterfly wings and parrot head. S-D^2 is capable of restoring sharper edges and textures. The D^3 models further reduce the unnatural artifacts occurring in S-D^2 results. Especially, while D^3-128 results still suffer from a small amount of visible ringing artifacts, D^3-256 not only shows superior in preserving details, but also suppresses artifacts well.

Analyzing the Impressive Results of D^3

We attribute our impressive recovery of clear fine details to the combination of our specific pipeline, the initialization, and the box-constrained loss.

Task-specific and interpretable pipeline The benefits of our specifically designed architecture were demonstrated by the comparison experiments to baseline encoders. Further, we provide intermediate outputs of the IDCT layer, i.e., the recovery after the DCT-domain reconstruction. We hope that it helps understand how each component, i.e., the DCT-domain reconstruction or the pixel-domain reconstruction, contributes to the final results. As shown in Figure 3.22 (a)-(c), such intermediate reconstruction results contain both sharpened details (see the characters in (a), which become more recognizable), and unexpected noisy patterns (see (a) (b) (c) for the blockiness, and ringing-type noise along edges and textures). It implies that Stage I DCT-domain reconstruction has enhanced the high-frequency features, yet introducing artifacts simultaneously due to quantization noises. Afterwards, Stage II pixel-domain reconstruction performs extra noise suppression and global reconstruction, which leads to the artifact-free and

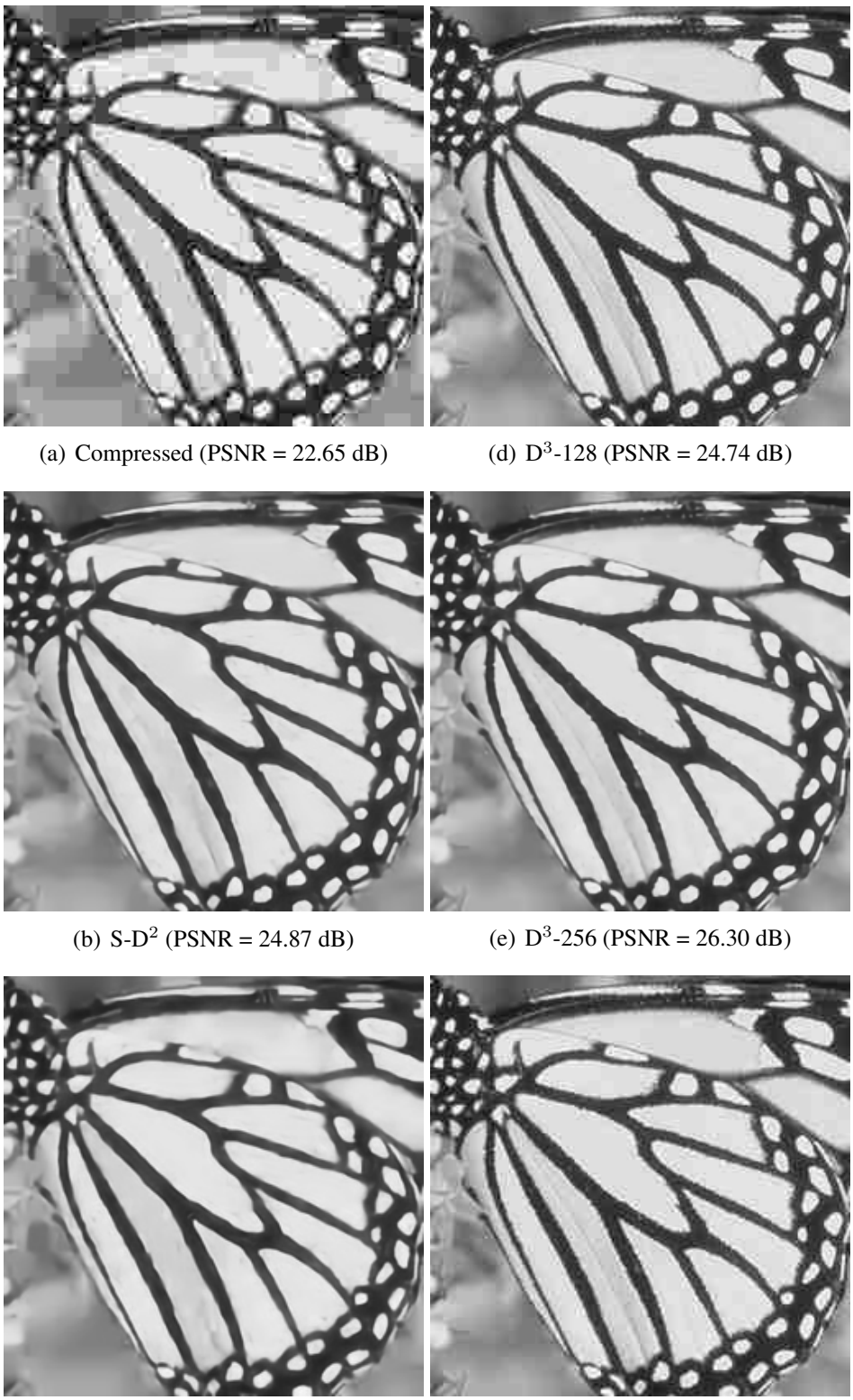


(a) Compressed (PSNR = 21.72 dB) (d) D^3-128 (PSNR = 23.94 dB)

(b) S-D^2 (PSNR = 22.87 dB) (e) D^3-256 (PSNR = 24.30 dB)

(c) AR-CNN (PSNR = 23.27 dB) (f) D-Base-256 (PSNR = 23.48 dB)

Figure 3.19: Visual comparison of various methods on *Bike* at Q = 5. The corresponding PSNR values (in dB) are also shown.

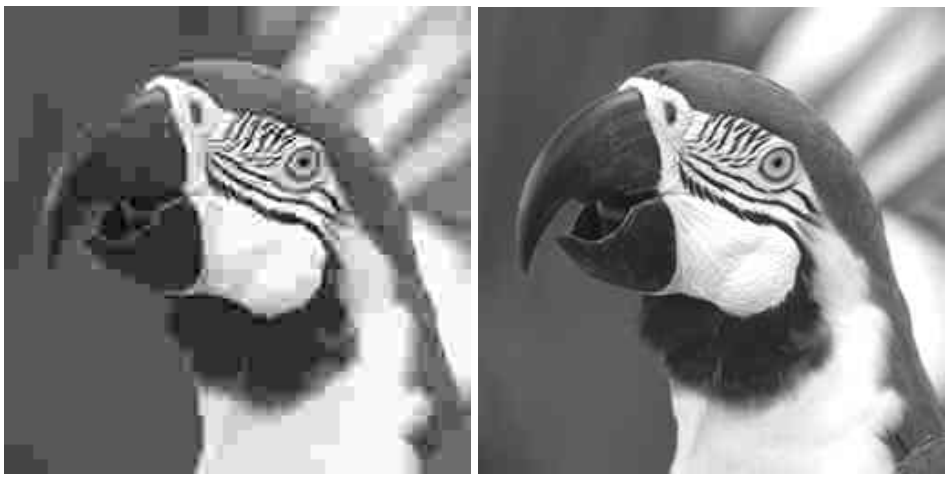


(a) Compressed (PSNR = 22.65 dB) (d) D^3-128 (PSNR = 24.74 dB)

(b) S-D^2 (PSNR = 24.87 dB) (e) D^3-256 (PSNR = 26.30 dB)

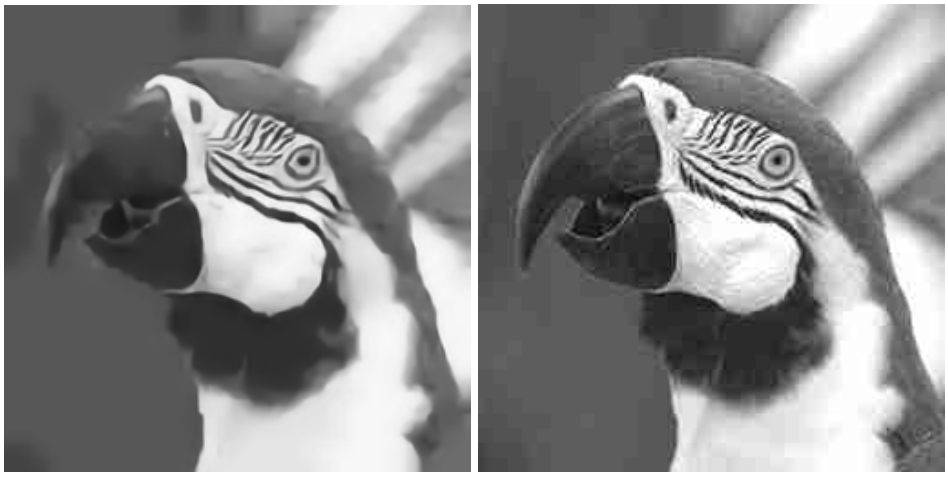
(c) AR-CNN (PSNR = 25.81 dB) (f) D-Base-256 (PSNR = 24.28 dB)

Figure 3.20: Visual comparison of various methods on *Monarch* at Q = 5. The corresponding PSNR values (in dB) are also shown.



(a) Compressed (PSNR = 26.15 dB) (d) D^3-128 (PSNR = 27.52 dB)

(b) S-D^2 (PSNR = 27.92 dB) (e) D^3-256 (PSNR = 28.84 dB)



(c) AR-CNN (PSNR = 28.20 dB) (f) D-Base-256 (PSNR = 27.21 dB)

Figure 3.21: Visual comparison of various methods on *Parrots* at Q = 5. The corresponding PSNR values are also shown.

more visually pleasing final results.

Sparse coding-based initialization We conjecture that the reason why D^3 is more capable in restoring the text on *Bike* and other subtle textures hinges on our sparse coding-based initialization, as an important training detail in D^3. To verify that, we re-train D^3 with random initialization, with the testing results in Figure 3.22 (d)-(f), which turn out to be visually smoother (closer to AR-CNN results). For example, the characters in (d) are now hardly recognizable. We notice that the S-D^2 results, as in original Figures 3.19-3.21 (c), also presented sharper and more recognizable texts and details than AR-CNN. These observations validate our conjecture. So the next question is: **Why does sparse coding help significantly here**? The quantization process can be considered as as a low-pass filter that cuts off high-frequency information. The dictionary atoms are learned from offline high-quality training images, which contain rich high-frequency information. The sparse linear combination of atoms is thus richer in high-frequency details, which might not necessarily be the case in generic regression (as in deep learning).

Box-constrained loss The loss L_B (3.32) acts as another effective regularization. We re-train D^3 without the loss, and obtain the results in Figure 3.22 (g)-(i). It is observed that the box-constrained loss helps generate details (e.g., comparing characters in (g) with those in Figure 3.19 (f)), by bounding the DCT coefficients, and brings PSNR gains.

Running Time Comparison

The image or video codecs desire highly efficient compression artifact removal algorithms as the post-processing tool. Traditional TV and digital cinema business uses frame rate standards such as 24p (i.e., 24 frames per second), 25p, and 30p. Emerging standards require much higher rates. For example, high-end High-Definition (HD) TV systems adopt 50p or 60p; the Ultra-HD (UHD) TV standard advocates 100p/119.88p/120p; the HEVC format could reach the maximum frame rate of 300p [1]. To this end, higher time efficiency is as desirable as improved performances.

We compare the averaged testing times of AR-CNN and the proposed D^3 models in Table 3.16, on the LIVE29 dataset, using the same machine and software environment. All running time was collected from GPU tests. Our best model, D^3-256, takes approximately 12 ms per image; that is more than **30 times faster** than AR-CNN. The speed difference is NOT mainly caused by the different im-

Table 3.16: Averaged running time comparison (ms) on LIVE1.

	AR-CNN	D^3-128	D^3-256	D-Base-256
Q = 5	396.76	7.62	12.20	9.85
Q = 10	400.34	8.84	12.79	10.27
Q = 20	394.61	8.42	12.02	9.97

plementations. Both being completely feed-forward, AR-CNN relies on the time-consuming convolution operations while ours takes only a few matrix multiplications. That is in accordance with the theoretical time complexities computed from (3.38) and (3.39), too. As a result, D^3-256 is able to process 80p image sequences (or even higher). To our best knowledge, D^3 is the **fastest** among all state-of-the-art algorithms, and proves to be a practical choice for HDTV industrial usage.

3.5.5 Conclusion

We introduce the D^3 model for the fast restoration of JPEG compressed images. The successful combination of both JPEG prior knowledge and sparse coding expertise has made D^3 highly effective and efficient. In the future, we aim to extend the methodology to more related applications.

(a) PSNR = 22.14 dB (d) PSNR = 23.42 dB (g) PSNR = 23.80 dB

(b) PSNR = 25.14 dB (e) PSNR = 24.85 dB (h) PSNR = 25.63 dB

(c) PSNR = 26.74 dB (f) PSNR = 27.63 dB (i) PSNR = 28.28 dB

Figure 3.22: Intermediate and comparison results, on *Bike*, *Monarch*, and *Parrot*, at Q = 5: (a) - (c) the intermediate recovery results after the DCT-domain reconstruction; (d) - (f) the results trained with random initialization; (g) - (i) the results trained without the box-constrained loss. PSNR values are reported.

CHAPTER 4

DEEP MODELS MADE INTERPRETABLE: A COGNITIVE SCIENCE AND NEUROSCIENCE PERSPECTIVE

Deep networks attempt to emulate the underlying complex neural mechanisms of human perception, and display the ability to describe semantic content from the primitive level to the abstract level. On the other hand, the study of the cognitive and neural underpinnings of human perception, by means of neuroimaging techniques and behavioral experiments, yields some promise for understanding brain perception. Whereas deep learning is known to be analogous to brain mechanisms, there has been little progress to combine scientific findings and learning-based models into an integrative framework.

We currently start investigating the synergy among feature learning, cognitive science and neuroscience. We expect to help understand the psychological and neuroscience mechanisms of human perception tasks, which will promote a better feature learning model to describe and estimate human preferences. The learning model can in turn help understand brain mechanisms. One recent progress lies in tackling the automated assessment of pictorial aesthetics, a challenging visual perception task due to its subjective nature. Inspired by the neuroaesthetics findings, we design a task-specific deep architecture that proves predictive of human aesthetics ratings. We further apply deep models to the novel application domain of brain encoding, with promising preliminary results achieved.

4.1 Image Aesthetics Assessment using Deep Chatterjee's Machine

Image aesthetics assessment has been challenging due to its subjective nature. Inspired by the Chatterjee's visual neuroscience model, we design Deep Chatterjee's Machine (DCM) tailored for this task. DCM first learns attributes through the parallel supervised pathways, on a variety of selected feature dimensions. A high-level synthesis network is trained to associate and transform those attributes

into the overall aesthetics rating. We also highlight our first-of-its-kind study of label-preserving transformations in the context of aesthetics assessment, which leads to an effective data augmentation approach. Experimental results on the AVA dataset show that DCM gains significant performance improvement, compared to other state-of-the-art models.

4.1.1 Introduction

Automated assessment or rating of pictorial aesthetics has many applications, such as in an image retrieval system or a picture editing software [35]. Compared to many typical machine vision problems, the aesthetics assessment is even more challenging, due to the highly subjective nature of aesthetics, and the seemingly inherent semantic gap between low-level computable features and high-level human-oriented semantics. Though aesthetics influences many human judgments, our understanding of what makes an image aesthetically pleasing is still limited. Contrary to semantics, an aesthetic response is usually very subjective and difficult to gauge even among human beings.

Existing research has predominantly focused on constructing hand-crafted features that are empirically related to aesthetics. Those features are designed under the guidance of photography and psychological rules, such as rule-of-thirds composition, depth of field (DOF), and colorfulness [46], [85]. With the images being represented by these hand-crafted features, aesthetic classification or regression models can be trained on datasets consisting of images associated with human aesthetic ratings. However, the effectiveness of hand-crafted features is only empirical, due to the vagueness of certain photographic or psychologic rules. Recently, Lu et al. [110] proposed the *Rating Pictorial Aesthetics using Deep Learning* (**RAPID**) model, with impressive accuracies on the *Aesthetic Visual Analysis* (**AVA**) dataset [121]. However, they have not yet studied more precise predictions, such as finer-grained ratings or rating distributions [171]. On the other hand, the study of the cognitive and neural underpinnings of aesthetic appreciation by means of neuroimaging techniques yields some promise for understanding human aesthetics [27]. Although the results of these studies have been somewhat divergent, a hierarchical set of core mechanisms involved in aesthetic preference have been identified [31].

In this work, we develop a novel deep-learning based image aesthetics assess-

ment model, called *Brain-Inspired Deep Chatterjee's Machine* (**DCM**). DCM clearly distinguishes itself from prior models, for its unique architecture inspired the Chatterjee's visual neuroscience model [30]. We introduce the specific architecture of *parallel supervised pathways*, to learn multiple attributes on a variety of selected feature dimensions. Those attributes are then associated and transformed into the overall aesthetic rating, by a *high-level synthesis network*. Our technical contribution also includes the study of label-preserving transformations in the context of aesthetics assessment, which is applied to effective data augmentation. We examine DCM on the large-scale AVA dataset [121], for the aesthetics rating prediction task, and confirm its superiority over a few competitive methods, with the same or larger amounts of parameters.

Related Work

Datta et al. [46] first cast the image aesthetics assessment problem as a classification or regression problem. A given image is mapped to an aesthetic rating, which is usually collected from multiple subject raters and is normally quantized with discrete values. [46], [85] extracted various handcrafted features, including low-level image statistics such as distributions of edges and color histograms, and high-level photographic rules such as the rule of thirds. A few subsequent efforts, such as [12], [50], [111], focus on improving the quality of those features. Generic image features [114], such as SIFT and Fisher Vector [109], were applied to predict aesthetics. However, empirical features cannot accurately and exhaustively represent the aesthetic properties.

The human brain transforms and synthesizes a torrent of complex and ambiguous sensory information into coherent thought and decisions. Most aesthetic assessment methods adopt simple linear classifiers to categorize the input features, which is obviously oversimplified. Deep networks [9] attempt to emulate the underlying complex neural mechanisms of human perception, and display the ability to describe image content from the primitive level (low-level features) to the abstract level (high-level features). The RAPID model [110] is among the first to apply deep convolutional neural networks (CNN) [89] to the aesthetics rating prediction, where the features are automatically learned. They further improved the model by exploring style annotations [121] associated with images. In fact, even the hidden activations from a generic CNN were found to work reasonably well for aesthetics features [54].

Datasets

Large and reliable datasets, consisting of images and corresponding human ratings, are the essential foundation for the development of machine assessment models. Several Web photo resources have taken advantage of crowdsourcing contributions, such as Flickr and DPChallenge.com [121]. The AVA dataset is a large-scale collection of images and meta-data derived from DPChallenge.com. It contains over 250,000 images with aesthetic ratings from 1 to 10, and a 14,079 subset with binary style labels (e.g., rule of thirds, motion blur, and complementary colors), making automatic feature learning using deep learning approaches possible. In this section, we focus on AVA as our research subject.

4.1.2 The Neuroaesthetics Models

Multiple parallel processing strategies, involving over a dozen retinal ganglion cell types, can be found in the retina. Each ganglion cell type focuses on one specific kind of feature, and provides a complete representation across the entire visual field [122]. Retinal ganglion cells project in parallel from the retina, through the lateral geniculate nucleus of the thalamus to the primary visual cortex. Primary visual cortex receives parallel inputs from the thalamus and uses modularity, defined spatially and by cell-type specific connectivity, to recombine these inputs into new parallel outputs. Beyond primary visual cortex, separate but interacting dorsal and ventral streams perform distinct computations on similar visual information to support distinct behavioural goals [130]. The integration of visual information is then achieved progressively. Independent groups of cells with different functions are brought into temporary association, by a so-called "binding" mechanism [27], for the final decision-making.

From the retina to the prefrontal cortex, the human visual processing system will first conduct a very rapid holistic image analysis [156], [78], [158]. The divergence comes at a later stage, in how the low-level visual features are further processed through parallel pathways [61] before being utilized. The pathway can be characterized by a hierarchical architecture, in which neurons in higher areas code for progressively more complex representations by pooling information from lower areas. For example, there is evidence [135] that neurons in V1 code for relatively simple features such as local contours and colors, whereas neurons in TE fire in response to more abstractive features, that encode the scene's gist and/or

saliency information and act as a holistic signature of the input.
Key Notations: For the consistency of terms, we use *feature dimension* to denote a prominent visual property, that is relevant to aesthetics judgement. We define an *attribute* as the learned abstracted, holistic feature representation over a specific feature dimension. We define a *pathway* as the processing mechanism from a raw visual input to an attribute.

Chatterjee's Visual Neuroscience Model

The main insights for DCM were gained from the classical and important **Chatterjee's visual neuroscience model** [30]. It models the cognitive and affective processes involved in visual aesthetic preference, providing a means to organize the results obtained in the 2004-2006 neuroimaging studies, within a series of information-processing phases. The Chatterjee's model yields the following simplified but important insights that inspire our model:

- The human brain works as a multi-leveled system.
- For the visual sensory input, a variety of relevant feature dimensions are first targeted.
- A set of parallel pathways abstract the visual input. Each pathway processes the input into an attribute on a specific feature dimension.
- The high-level association and synthesis transforms all attributes into an aesthetic decision.

Steps 2 and 3 are derived from the many recent advances [122] showing that aesthetics judgments evidently involve multiple pathways, which could connect from related perception tasks [27], [31]. Previously, many feature dimensions, such as color, shape, and composition, have already been discovered to be crucial for aesthetics. A bold yet rational assumption is thus made by us, that the attribute learning for aesthetics tasks could be decomposed onto those pre-known feature dimensions and processed in parallel.

4.1.3 Deep Chatterjee's Machine

The architecture of Deep Chatterjee's Machine (DCM) is depicted in Figure 4.1. The whole training process is divided in two stages, based on the above insights.

Table 4.1: The 14 style attribute annotations in the AVA dataset

Style	Number	Style	Number
Complementary Colors	949	Duotones	1, 301
High Dynamic Range	396	Image Grain	840
Light on White	1,199	Long Exposure	845
Macro	1,698	Motion Blur	609
Negative Image	959	Rule of Thirds	1,031
Shallow DOF	710	Silhouettes	1,389
Soft Focus	1,479	Vanishing Point	674

In brief, we first learn attributes through parallel (supervised) pathways, over the selected feature dimensions. We then combine those *"pre-trained"* pathways with the high-level synthesis network, and *jointly tune* the entire network to predict the overall aesthetics ratings. The testing process is completely feed-forward and end-to-end.

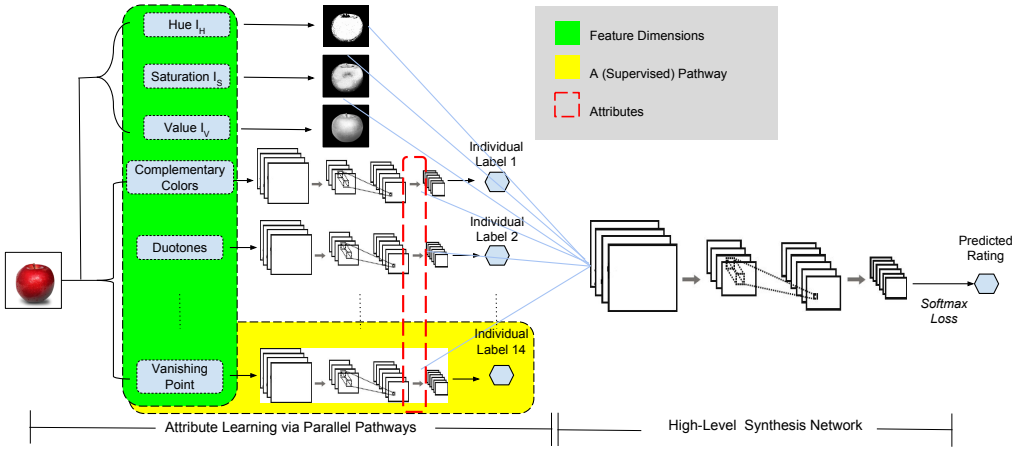


Figure 4.1: The architecture of Deep Chatterjee's Machine (DCM). The input image is first processed by parallel pathways, each of which learns an attribute along a selected feature dimension independently. Except for the first three simplest features (*hue, saturation, value*), all parallel pathways take the form of fully-convolutional networks, supervised by individual labels; their hidden layer activations are utilized as learned attributes. We then associate those *"pre-trained"* pathways with the high-level synthesis network, and *jointly tune* the entire network to predict the overall aesthetics ratings.

Selecting Feature Dimensions We first select feature dimensions that are discovered to be highly related to aesthetic assessment. Despite the lack of firm rules, certain visual features are believed to please humans more than others [46]. We take advantage of those photographically or psychologically inspired features as priors, and force DCM to "focus" on them.

The previous work, e.g., [46], has identified a set of aesthetically discriminative features. It suggested that the light exposure, saturation and hue play indispensable roles. We assume the RGB data of each image is converted to HSV color space, as I_H, I_S, and I_V, where each of them has the same size as the original image.[1] Furthermore, many photographic style features influence humans' aesthetic judgements. [46] proposed six sets of photographic styles, including the rule of thirds composition, textures, shapes, and shallow depth-of-field (DOF). The AVA dataset comes with a more enriched variety of *style annotations*, as listed in Table 4.1, which are leveraged by us. [2]

Parallel Supervised Pathways Among the 17 feature dimensions, the simplest three, I_H, I_S, and I_V, are immediately obtained from the input. However, the remaining 14 style feature dimensions are not qualitatively well-defined; their attributes are not straightforwardly extracted.

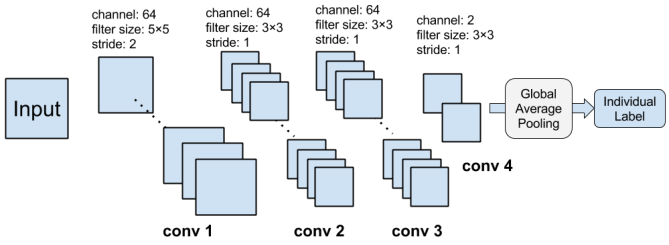


Figure 4.2: The architecture of a supervised pathway as a FCNN. A 2-way softmax classifier is employed after global averaging pooling, to predict the individual label (0 or 1).

For each style category as a feature dimension, we create binary *individual*

[1] We downsample I_H, I_S, and I_V to 1/4 of their original size, to improve the efficiency. It turns out that the model performance is hardly affected, which is understandable since the human perceptions of those features are insensitive to scale changes.

[2] The 14 photographic styles are chosen specifically on the AVA datasets. We do not think they represent all aesthetics-related visual information, and plan to have more photographic styles annotated.

labels, by labelling images with the style annotation as "1" and otherwise "0", which follows many previous works [121, 50]. We design a special architecture, called *parallel supervised pathways*. Each pathway is modeled with a *fully convolutional neural network* (FCNN), as in Figure 4.2. It takes an image as the input, and outputs the image's individual label along this feature dimension. All pathways are learned in parallel without intervening with each other. The choice of FCNN is motivated by the spatial locality-preserving property of the human brain's low-level visual perception [130].

For each feature dimension, the number of labeled samples is limited, as shown in Table 4.1. Therefore, we pre-train the first two layers in Figure 4.2, using all images from the AVA dataset, in a unsupervised way. We construct a 4-layer Stacked Convolutional Auto Encoder (SCAE): its first 2 layers follows the same topology as the conv1 and conv2 layers, and the last 2 layers are mirror-symmetrical deconvolutional layers [181]. After SCAE is trained, the first two layers are applied to initialize the conv1 and conv2 layers for all 14 FCNN pathways. The strategy is based on the common belief that the lower layers of CNNs learn general-purpose features, such as edges and contours, which could be adapted for extensive high-level tasks [51].

After the initialization of the first two layers, for each pathway, we concatenate the conv3 and conv4 layers, and further conduct supervised training using individual labels. The conv4 layer always has the same channel number with the corresponding style classes (here the channel number is 2 for all, since we only have binary labels for each class). It is followed by the global average pooling [103] step, to be correlated with the binary labels. Eventually, the conv4 layer as well as the classifier are discarded, and the conv1-conv3 layers of 14 pathways are passed to the next stage. We treat the conv3 layer activations of each pathway as learned attributes [51].

The pathways in DCM account for progressively extracting more complex features. As observed in experiments, the pre-training of all pathways' conv1 and conv2 layers learns shared low-level features, such as edges and blobs. Each pathway is then independently tuned by its "higher-level" concepts, which guides the adaptation of low-level features. The final outputs of pathways, conv3, are abstracted from the low-level conv1 and conv2 features, and are regarded as mid-level attributes. Each pathway's conv3 attribute displays a different, visible combination of low-level features, but not any semantically meaningful object.

Table 4.2: The subjective evaluation survey on the aesthetic influences of various transformations (s denotes a random number)

Transformation	Description	LP factor
Reflection	Flipping the image horizontally	0.99
Random scaling	Scale the image proportionally by $s \in$ [0.9, 1.1]	0.94
Small noise	Add a Gaussian noise $\in N(0, 5)$	0.87
Large noise	Add a Gaussian noise $\in N(0, 30)$	0.63
Alter RGB	Perturbed the intensities of the RGB channels [89]	0.10
Rotation	Randomly-parameterized affine transformation	0.26
Squeezing	Change the aspect ratio by $s \in$ [0.8, 1.2]	0.55

Training High-Level Synthesis Network

Finally, we simulates the brain's high-level association and synthesis, using a larger FCNN. Its architecture resembles Figure 4.2, except that the first three convolutional layers each have 128 channels instead of 64. The high-level synthesis network takes the attributes from all parallel pathways as inputs, and outputs the overall aesthetics rating. The entire DCM is then tuned from end to end.

4.1.4 Study Label-Preserving Transformations

When training deep networks, the most common approach to reduce overfitting is to artificially enlarge the dataset using label-preserving transformations [16]. In [89], image translations and horizontal reflections are generated, while the intensities of the RGB channels are altered, both of which apparently will not change the object class labels. Other alternatives, such as random noise, rotations, warping and scaling, are also widely adopted by the latest deep-learning based object recognition methods. However, there has been little work on identifying label-preserving transformations for image aesthetics assessment, e.g., those that will not significantly alter the human aesthetics judgements, considering the rating-based labels are very subjective. In [110], motivated by their need to create fixed-size inputs, the authors created randomly-cropped local regions from training images, which was empirically treated as data augmentation.

We make the first exploration to identify whether a certain transformation will preserve the *binary aesthetics rating*, i.e., high quality versus low quality, by conducting a **subjective evaluation survey** among over 50 participants. We select 20

high-quality (δ = 1) images from the AVA dataset (since low-quality images are unlikely to become more aesthetically pleasing after some simple/random transformations). Each image is processed by all different kinds of transformations in Table 4.2. For each time, a participant is shown with a set of image pairs originated from the same image, but processed with different transformations. The groundtruth is also included in the comparison process. For each pair, the participant needs to decide which one is better in terms of aesthetic quality. The image pairs are drawn randomly, and the image winning this pairwise comparison will be compared again in the next round, until the best one is selected.

We fit a Bradley-Terry [19] model to estimate the subjective scores for each method so that they can be ranked. With the groundtruth set as score 1, each transformation will receive a score between [0, 1]. We define the score as the *label-preserving* (**LP**) factor of a transformation; a larger LP factor denotes a smaller impact on image aesthetics. As in Table 4.2, *reflection* and *random scaling* receive the highest LR factors. The small noise seems to affect the aesthetics feelings negatively, but only marginally. All others are shown to significantly degrade human aesthetics perceptions. We therefore adopt reflection, random scaling, and small noise as our default data augmentation approaches.

4.1.5 Experiment

We implement our models based on the cuda-convnet package [89]. The ReLU nonlinearity as well as dropout is applied. Following RAPID [110], we evaluate DCM on the binary aesthetic rating task. We quantize images' mean ratings into binary values. Images with mean ratings smaller than $5 - \delta$ are labeled as low-quality, while those with mean ratings larger than $5 + \delta$ are referred to as high-quality. The adjustment of learning rates in such a hierarchical model calls for special attention. We first train the 14 parallel pathways, with the identical learning rates: η = 0.05 for unsupervised pre-training and 0.01 for supervised tuning, both of which are not annealed throughout training. We then train the high-level synthesis network on top of them and fine-tune the entire DCM. For the pathway part, its learning rate η' starts from 0.001; for the high-level part, the learning rate ρ starts from 0.01. When the training curve reaches a plateau, we first try dividing ρ by 10, and further try dividing ρ by 10 if the training/validation error still does not decrease.

Static Regularization vs. Joint Tuning The RAPID model [110] also extracted attributes along different columns (pathways) and combine them. The pre-trained style classifier was then "frozen" and acted as a static network regularization. Out of curiosity, we also tried to fix our parallel pathways while training the high-level synthesis network, e.g., $\eta' = 0$. The resulting performance was verified to be inferior to that of joint tuning the entire DCM.

We compare DCM with the state-of-the-art RAPID model for binary aesthetics rating prediction. Benefiting from our fully-convolutional architecture, DCM has a much lower parameter capacity than RAPID that relies on fully-connected layers. Besides, we construct three baseline networks, all with exactly the same parameter capacity as DCM:

- **Baseline fully-convolutional network (BFCN)** first binds the conv1 – conv3 layers of 14 pathways horizontally, constituting a three-layer fully convolutional network, each layer with $64 \times 14 = 896$ filter channels. Such an attribute learning part is trained in a unsupervised way, with style annotations utilized. It is then concatenated with the high-level synthesis network, to be jointly supervised-tuned.

- **DCM without parallel pathways (DCM-WP)** utilizes style annotations in an entangled fashion. Its only difference with BFCN lies in that the training of the attribute learning part is supervised by a composite label $\in R^{28\times 1}$, which binds 14 individual labels altogether.

- **DCM without data augmentations (DCM-WA)** denotes DCM without the three data augmentations applied (reflection, scaling, and small noise).

We train the above five models for the binary rating prediction, with both $\delta = 0$ and $\delta = 1$. The overall accuracies are compared in Table 4.3.[3] It appears that BFCN performs significantly worse than others, due to the absence of the style attribute information. While RAPID, DCM-WP and DCM all utilize style annotations as the supervision, DCM outperforms the other two in both cases with remarkable margins. By comparing DCM-WP with DCM, we observe that the biologically-inspired parallel pathway architecture in DCM facilitates the learning. Such a specific architecture avoids overly large all-in-one models (such as DCM-WP), but instead have more effective, dedicated sub-models. In DCM, style annotations serve as powerful priors, to enforce DCM to focus on extracting features that are

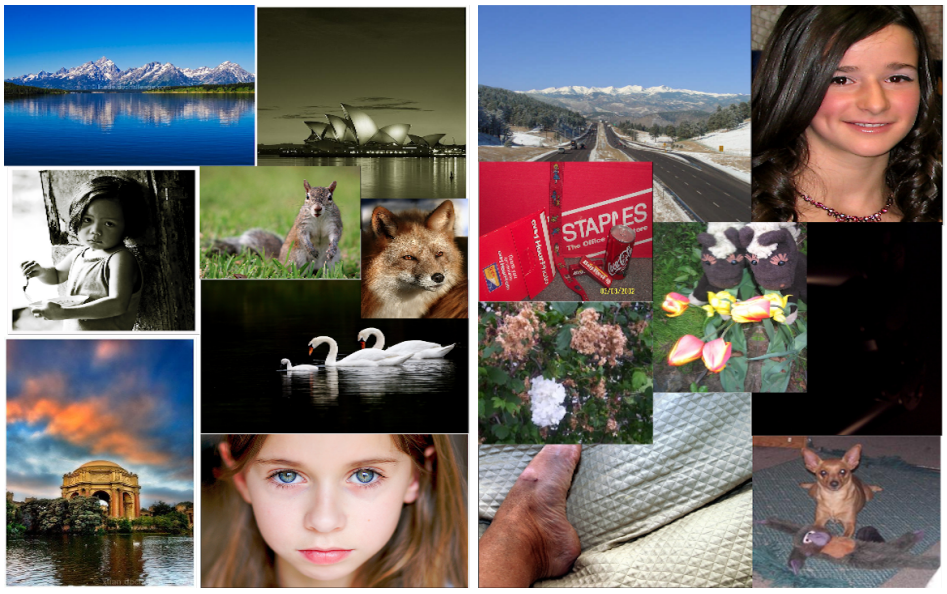
[3]The accuracies of RAPID are from the RDCNN results in Table 3 [110].

Table 4.3: The accuracy comparison of different methods for rating prediction.

	RAPID	BFCN	DCM-WP	DCM-WA	DCM
$\delta = 0$	74.46%	70.20%	73.54%	74.03%	76.80%
$\delta = 1$	73.70%	68.10%	72.23%	73.72%	76.04%

highly correlated to aesthetics judgements. The DCM is jointly tuned from end to end, which is different from RAPID whose style column only acts as a static regularization. We also notice a gain of nearly 3% of DCM over DCM-WA, which verifies the effectiveness of our proposed augmentation approaches.

In [121], a linear classifier was trained on fisher vectors computed from color and SIFT descriptors. Under the same aesthetic quality categorization setting, the baselines reported by [121] were 66.7% when $\sigma = 0$, and 67.0% when $\sigma = 1$, falling far behind both DCM and RAPID.



(a) (b)

Figure 4.3: Classification examples of the DCM model: (a) high-quality; (b) low-quality ($\delta = 0$).

To qualitatively analyze the results, we display eight images correctly classified by DCM to be high-quality when $\delta = 0$, in Figure 4.3 (a), and eight correctly classified low-quality images in Figure 4.3 (b). The images ranked high in terms of aesthetics typically present salient foreground objects, low depth of field, proper composition, and color harmony. In contrast, low-quality images are at least de-

(a) (b)

Figure 4.4: How contexts and emotions could alter the aesthetic judgment. (a) Incorrectly classified examples ($\delta = 0$) due to semantic contents; (b) High-variance examples (correctly predicted by DCM), which have nonconventional styles or subjects.

fected in one aspect. For example, the top left image has no focused foreground object, while the bottom right one suffers from a messy layout. Regarding the top right "girl" portrait in Fig 4.3 (b), the original comments on DPChallenge.com show that people rated it low because of the noticeable detail loss caused by noise reduction post-processing, as well as the unnatural "plastic-like" lights on her hair.

More interestingly, Figure 4.4 (a) lists two **failure** examples of DCM. The left image in Figure 4.4 (a) depicts a waving glowstick captured by time-lapse photography. The image itself has no appealing composition or colors, and is thus identified by DCM to be low-quality. However, the DPChallenge raters/commenters were amazed by the angel shape and rated it very favorably due to the creative idea. The right image, in contrast, is a high-quality portrait, on which DBN confidently agrees. However, it was associated with the "Rectangular" challenge topic on DPChallenge, and was rated low because this targeted theme was overshadowed by the woman. The failure examples manifest the huge subjectivity and sensitivity of human aesthetic judgement.

4.1.6 Conclusion

In this section, we draw inspiration from the knowledge abstracted from human visual perception and neuroaesthetics, and formulate the Deep Chatterjee's Machine (DCM). The biologically inspired, task-specific architecture of DCM leads to better performance than other state-of-the-art models with the same or higher parameter capacity. Since it has been observed in Figure 4.4 that emotions and

contexts could alter the aesthetic judgment, we plan to take the two factors into account for a more comprehensive framework.

4.2 Brain Encoding using Deep Models: An Exploratory Study

4.2.1 Background and Motivation

In this part, we explore the applicability of deep networks to the novel application domain of brain encoding [83]. *Brain encoding* refers to the challenging task to predict the brain activity, e.g., blood oxygenation level dependent (BOLD) responses, from the stimuli. Lately, [84] developed a two-stage cascaded *second-order contrast* (**SOC**) model, that accepts a grayscale image as input and predicts BOLD responses in early visual cortex as output. The SOC model has a cascade architecture, consisting of two stages of linear and nonlinear operations. The first stage involves well-established computations - local oriented filters and divisive normalization - whereas the second stage involves novel computations - compressive spatial summation (a form of normalization) and a variance-like nonlinearity that generates selectivity for second-order contrast. The SOC model has only eight controlling parameters: it heavily relies on specific nonlinear computations, that are summarized from neuroscience expertise. The philosophy behind SOC is that because data are limited, parameters are precious and we have to incorporate very specific computations into models.

In this section, we try to apply more parameterized deep models. Such models could be more flexible, by allowing the data to inform the model as to what types of computations are necessary. One major obstacle arises from the fact that training data are extremely limited. It is also infeasible to artificially increase the data volume, such as generating "synthetic" data or perform data augmentation. Therefore, the classical "data-driven" setting for deep learning, as well as many popular training techniques, does not directly apply here.

4.2.2 Dataset

We refer to Kendrick Kay et al.'s publicly available datasets of BOLD responses in visual cortex[4], measured by functional magnetic resonance imaging (fMRI) in human subjects. Specifically, we adopt their stimulus set 2, stimulus set 3, (response) dataset 4, and (response) dataset 5.

All stimuli are band-pass filtered grayscale images. Following [84], we resize them to 150×150 pixels. Stimulus sets 2 and 3 consist of **156** and **35** distinct stimuli, respectively. The responses at a total of 200 voxels are recorded. On each voxel, a scalar fMRI measurement was measured given each input stimulus. Note that each voxel needs to train a separate brain encoding model. Dataset 4 consists of one person's responses to stimulus set 2, while dataset 5 has the same person's responses to stimulus set 3. Details about the datasets could be found at [84].

Our goal is to train a regression-type model using stimulus set 2 and dataset 4 (as the *training set*). The model is used to generate predictions and be evaluated on stimulus set 3 and dataset 5 (as the *testing set*). Obviously, it is an ill-conditioned "small data" problem.

4.2.3 Model and Experiment

To design a well-adapted architecture that learns with very limited data, it is necessary to incorporate task-specific domain expertise. Since we consider the visual stimuli (structured natural images) as the input, it is straightforward to choose fully convolutional networks [82] as the computational model, and to solve brain encoding as a regression problem. Fully convolutional networks also cost fewer parameters than typical deep models with fully-connected layers. For brain encoding, The SOC model [84] further suggested that the encoding in the human brain would go through some type of *spatial summation*, followed by compressive power-law function (*compressive nonlinearity*). That motivates us to try the sigmoid neuron and average pooling, and compare their effects with the popular ReLu neuron and max pooling, in this specific scenario.

We construct five different deep models for comparison:

- **Model I:** 1-hidden-layer fully convolutional network, with one convolutional layer configured by: channel number = 16, filter size = 3, stride =

[4] `http://kendrickkay.net/socmodel/`

Table 4.4: The averaged R^2 performance comparison of different models for brain encoding.

Model	I	II	III	IV	V	SOC
R^2	82.6027	88.0655	87.4333	87.7403	88.0646	87.7628

2, zero padding = 2, followed by a global average pooling operator.[5] The output is further concatenated with average pooling operator, followed by a mean square error (MSE) loss.

- **Model II:** 2-hidden-layer fully convolutional network, by adding the second convolutional layer with channel number = 8, filter size = 3, stride = 1, zero padding = 1. All other configurations are identical to Model I.
- **Model III:** 3-hidden-layer fully convolutional network, by adding the third convolutional layer, whose configuration remains the same as the second one's. All others are identical to Model II.
- **Model IV:** replacing the sigmoid neurons in Model II with the RELU neurons, while leaving all others unchanged.
- **Model V:** replacing the last average pooling operator, in Model II with max-pooling, while leaving all others unchanged.

We initialize the first layer using PCA, and the remaining layers (if any) using layer-wise pre-training [10], to carefully ensure that those models converge properly. A earning rate of 0.01 and a momentum of 0.9 are applied. The model accuracy is quantified as the percentage of variance explained (R^2) in the measured response amplitudes by the cross-validated predictions of the response amplitudes (see page. 14, [84] for definitions). R^2 ranges between [0, 100]: the higher R^2 is, the more accurate the model is.

The preliminary experiment has found encouraging results, as in Table 4.4. In terms of *averaged* R^2 performance across 200 voxels, the SOC baseline reaches 87.7028, which is very competitive and shows neuroscience expertise to be powerful. By adding one more layer, Model II gains a sharp advantage of 5.4% over Model I, and also outperforms the SOC slightly. However, increasing more

[5]The global average pooling operator is inserted for fusing all channels into one feature map. It should be distinguished from the following pooling operator, which performs feature selection over the feature map.

hardly brings any further benefit, as evidenced by the slight performance drop from Model II to III. That reflects the potential overfitting problem due to the limited training data, and calls for the collection of larger datasets.

A comparison between Models II and IV reminds us that the sigmoid neurons potentially suit the brain encoding scenario better than the popular choice of ReLU neuron. We may view it as a success of learning from neuroscience; however, we realize that sigmoid neurons are more likely to suffer computational concerns ("saturation") when the networks grow deeper. That could be partially alleviated by pre-training.

On the other hand, choosing either max or average pooling operator does not affect the performance much in this experiments. Previous literature [126] suggests that the sensory processing in the brain suggests a sparse coding strategy over a highly over-complete basis, when finding stimuli that effectively activate the neurons. Therefore, we conjecture that the max pooling operator, which introduces sparsity, brings in performance benefits too, which may also find neuroscience grounds in the brain encoding process.

4.2.4 Remarks and Discussions

The above experiments, although restricted by the training data availability, have shown promise of the task-specific deep architectures designed from neuroscience expertise. While preliminary conclusions were drawn, several open questions remain for our future investigation, to list a few:

- If more training data can be collected, will a deeper architecture be useful? Does it really cost thousands of layers to model a brain network?
- Will sigmoid always outperform ReLU? If so, how to resolve the computational difficulties of the former, i.e., the saturation phenomenon of compressive nonlinearity?
- Is sparsity also an indispensable part of brain encoding? Will a mixed utility of average pooling and max pooling boost the model performance further?
- Is there really a clear mapping between the neural network layers and the brain hierarchy? In other words, to what extent could the neuroscience expertise guide the deep architecture design?

CHAPTER 5

CONCLUSION AND FUTURE RESEARCH

The research topics discussed throughout this dissertation share the same underlying theme: developing task-specific and interpretable feature learning models. A majority of my research is focused on bridging traditional learning models that emphasize problem-specific reasoning, and deep models that allow for larger learning capacity. Several concrete model examples are presented, to reveal how the analytic tools in the classical optimization problems can be translated to guide the architecture design and performance analysis of deep models. As a result, those models demonstrate improved performance, intuitive interpretation, and efficiency. The dissertation also discussed the potential synergy between feature learning, and cognitive science together with neuroscience. For describing and estimating human preferences, deep models can also be customized by the knowledge of the cognitive and neural underpinnings of human perception.

Furthermore, there are a wide variety of promising research directions that I wish to pursue further. As the dissertation established certain analytical correspondence relationships between deep learning and sparse coding, the next step aims to apply a similar methodology to more classical "shallow" models, where important priors and assumptions beyond sparsity could be incorporated either implicitly in feature engineering or explicitly as model constraints, such as the linear additivity of signals, the three-dimensional geometry of objects, and statistical assumptions such as conditional independence, latent variable structure, and low-rank covariances.

In addition to deriving more novel model architectures, I also have the ambition to connect the analysis of deep models to the rich theoretical results obtained in "shallow" models. The translation of those mature analytical tools is likely to boost the badly-needed behavior interpretations and performance guarantees for deep models. For example, one of my ongoing projects interprets a conventional deep model as multiple one-step truncated sparse coding models stacked hierarchically. In that way, I study its convergence properties by taking advantage of

the analytic tools that have been found effective in studying the theoretical performances of sparse coding models.

To apply feature learning techniques to the emerging "Big Data" scenarios, one needs to combat extra difficulties such as large volumes, high dimensions, imbalance between classes, heterogeneous sources, weakly-structured or unstructured data, ambiguous labeling, noises, incompleteness, and rich contexts. Such are the challenges and roadblocks for the development of new applicable optimization algorithms, where I have a keen interest.

Whereas deep learning has already been known to be analogous to brain mechanisms, there has been little progress to combine scientific findings and learning-based models into an integrative framework. Starting from the preliminary work in Chapter 4, I am looking forward to more systematically investigating the synergy among feature learning, cognitive science and neuroscience. More psychological and neuroscience mechanisms are expected to be modeled as the domain expertise, and to be combined into the design of relevant deep models.

Finally, fast-growing biomedical and healthcare data have encompassed multiple scales ranging from molecules, to individuals, to populations and have connected various entities in healthcare systems (providers, pharma, payers) with increasing bandwidth, depth, and resolution. Those data are becoming an enabling resource for accelerating basic science discoveries and facilitating evidence-based clinical solutions. Meanwhile, the sheer volume and complexity of the data present major barriers toward their translation into effective clinical actions. It thus also appeals to me to apply cutting-edge feature learning techniques to tackle real-world medical and healthcare problems.

REFERENCES

[1] https://en.wikipedia.org/wiki/Frame_rate/.

[2] M. Adlers, *Sparse Least Squares Problems with Box Constraints*. Citeseer, 1998.

[3] M. Aharon, M. Elad, and A. Bruckstein, "K-svd: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Transactions on Signal Processing (TSP)*, vol. 54, no. 11, pp. 4311–4322, 2006.

[4] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 33, no. 5, pp. 898–916, 2011.

[5] E. A. Ayele and S. Dhok, "Review of proposed high efficiency video coding (hevc) standard," *International Journal of Computer Applications*, vol. 59, no. 15, pp. 1–9, 2012.

[6] D. Barber and F. V. Agakov, "Kernelized infomax clustering," in *Advances in Neural Information Processing Systems (NIPS)*, 2005, pp. 17–24.

[7] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[8] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *Journal of machine learning research*, vol. 7, no. 11, pp. 2399–2434, 2006.

[9] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[10] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, "Greedy layer-wise training of deep networks," *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[11] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific Belmont, 1999.

[12] S. Bhattacharya, R. Sukthankar, and M. Shah, "A framework for photo-quality assessment and enhancement based on visual aesthetics," in *ACM Conference on Multimedia*. ACM, 2010, pp. 271–280.

[13] S. K. Bidhendi, A. S. Shirazi, N. Fotoohi, and M. M. Ebadzadeh, "Material classification of hyperspectral images using unsupervised fuzzy clustering methods," in *Signal-Image Technologies and Internet-Based System, 2007. SITIS'07. Third International IEEE Conference on*. IEEE, 2007, pp. 619–623.

[14] C. Biernacki, G. Celeux, and G. Govaert, "Assessing a mixture model for clustering with the integrated completed likelihood," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 22, no. 7, pp. 719–725, 2000.

[15] H. Bischof and A. Leonardis, "Finding optimal neural networks for land use classification," *IEEE transactions on Geoscience and Remote Sensing*, vol. 36, no. 1, pp. 337–341, 1998.

[16] C. M. Bishop, "Pattern recognition," *Machine Learning*, vol. 128, 2006.

[17] T. Blumensath and M. E. Davies, "Iterative thresholding for sparse approximations," *Journal of Fourier Analysis and Applications*, vol. 14, no. 5-6, pp. 629–654, 2008.

[18] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[19] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs the method of paired comparisons," *Biometrika*, vol. 39, no. 3-4, pp. 324–345, 1952.

[20] K. Bredies and M. Holler, "A total variation-based jpeg decompression model," *SIAM Journal on Imaging Sciences*, vol. 5, no. 1, pp. 366–393, 2012.

[21] A. Brooms, "Stochastic approximation and recursive algorithms with applications, 2nd edn by hj kushner and gg yin," *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, vol. 169, no. 3, pp. 654–654, 2006.

[22] L. Bruzzone, M. Chi, and M. Marconcini, "A novel transductive svm for semisupervised classification of remote-sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44, no. 11, pp. 3363–3373, 2006.

[23] G. Camps-Valls and L. Bruzzone, "Kernel-based methods for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 43, no. 6, pp. 1351–1362, 2005.

[24] G. Camps-Valls, L. Gómez-Chova, J. Calpe-Maravilla, J. D. Martín-Guerrero, E. Soria-Olivas, L. Alonso-Chordá, and J. Moreno, "Robust support vector method for hyperspectral data classification and knowledge discovery," *IEEE Transactions on Geoscience and Remote sensing*, vol. 42, no. 7, pp. 1530–1542, 2004.

[25] G. Camps-Valls, L. Gomez-Chova, J. Muñoz-Marí, J. Vila-Francés, and J. Calpe-Maravilla, "Composite kernels for hyperspectral image classification," *IEEE Geoscience and Remote Sensing Letters*, vol. 3, no. 1, pp. 93–97, 2006.

[26] E. J. Candes, M. B. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted l1 minimization," *Journal of Fourier Analysis and Applications*, vol. 14, no. 5-6, pp. 877–905, 2008.

[27] C. J. Cela-Conde, L. Agnati, J. P. Huston, F. Mora, and M. Nadal, "The neural foundations of aesthetic appreciation," *Progress in Neurobiology*, vol. 94, no. 1, pp. 39–48, 2011.

[28] H. Chang, M. K. Ng, and T. Zeng, "Reducing artifacts in jpeg decompression via a learned dictionary," *IEEE Transactions on Signal Processing (TSP)*, 2014.

[29] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 119–128.

[30] A. Chatterjee, "Prospects for a cognitive neuroscience of visual aesthetics," *Bulletin of Psychology and the Arts*, vol. 4, no. 2, pp. 55–60, 2003.

[31] ——, "Neuroaesthetics: a coming of age story," *Journal of Cognitive Neuroscience*, vol. 23, no. 1, pp. 53–62, 2011.

[32] G. Chen, "Deep learning with nonparametric clustering," *ArXiv preprint arXiv:1501.03084*, 2015.

[33] Y. Chen, N. M. Nasrabadi, and T. D. Tran, "Hyperspectral image classification using dictionary-based sparse representation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 49, no. 10, pp. 3973–3985, 2011.

[34] Y.-C. Chen, C. S. Sastry, V. M. Patel, P. J. Phillips, and R. Chellappa, "In-plane rotation and scale invariant clustering using dictionaries," *IEEE Transactions on Image Processing (TIP)*, vol. 22, no. 6, pp. 2166–2180, 2013.

[35] B. Cheng, B. Ni, S. Yan, and Q. Tian, "Learning to photograph," in *ACM Conference on Multimedia*. ACM, 2010, pp. 291–300.

[36] B. Cheng, J. Yang, S. Yan, Y. Fu, and T. S. Huang, "Learning with l1 graph for image analysis," *IEEE Transactions on Image Processing (TIP)*, vol. 19, no. 4, pp. 858–866, 2010.

[37] I. Choi, S. Kim, M. S. Brown, and Y.-W. Tai, "A learning-based approach to reduce jpeg artifacts in image matting," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2013, pp. 2880–2887.

[38] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "Nus-wide: a real-world web image database from national university of singapore," in *Proceedings of the ACM international conference on image and video retrieval*. ACM, 2009, p. 48.

[39] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of Operations Research*, vol. 153, no. 1, pp. 235–256, 2007.

[40] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[41] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *Journal of Machine Learning Research*, vol. 2, pp. 265–292, 2002.

[42] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[43] Z. L. Z. W. Da JIAO and L. Cheng, "Kernel clustering algorithm," *Chinese Journal of Computers*, vol. 6, p. 004, 2002.

[44] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Transactions on Image Processing (TIP)*, vol. 16, no. 8, pp. 2080–2095, 2007.

[45] B. Dai and B. Hu, "Minimum conditional entropy clustering: A discriminative framework for clustering." in *Asian Conference on Machine Learning (ACML)*, 2010, pp. 47–62.

[46] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Studying aesthetics in photographic images using a computational approach," in *European Conference on Computer Vision (ECCV)*. Springer, 2006, pp. 288–301.

[47] M. Davies and N. Mitianoudis, "Simple mixture model for sparse overcomplete ica," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 151, no. 1, pp. 35–43, 2004.

[48] S. M. Davis, D. A. Landgrebe, T. L. Phillips, P. H. Swain, R. M. Hoffer, J. C. Lindenlaub, and L. F. Silva, "Remote sensing: The quantitative approach," *New York, McGraw-Hill International Book Co.*, vol. 1, 1978.

[49] M. Denil and N. de Freitas, "Recklessly approximate sparse coding," *ArXiv preprint arXiv:1208.0959*, 2012.

[50] S. Dhar, V. Ordonez, and T. L. Berg, "High level describable attributes for predicting aesthetics and interestingness," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2011, pp. 1657–1664.

[51] 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 preprint arXiv:1310.1531*, 2013.

[52] C. Dong, Y. Deng, C. C. Loy, and X. Tang, "Compression artifacts reduction by a deep convolutional network," *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[53] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *European Conference on Computer Vision (ECCV)*. Springer, 2014, pp. 184–199.

[54] Z. Dong, X. Shen, H. Li, and X. Tian, "Photo quality assessment with dcnn that understands image well," in *MultiMedia Modeling*. Springer, 2015, pp. 524–535.

[55] D. L. Donoho, "De-noising by soft-thresholding," *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 613–627, 1995.

[56] D. L. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via l1 minimization," *Proceedings of the National Academy of Sciences*, vol. 100, no. 5, pp. 2197–2202, 2003.

[57] D. L. Donoho, M. Vetterli, R. A. DeVore, and I. Daubechies, "Data compression and harmonic analysis," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2435–2476, 1998.

[58] K. Duan, S. S. Keerthi, W. Chu, S. K. Shevade, and A. N. Poo, "Multi-category classification by soft-max combination of binary classifiers," in *International Workshop on Multiple Classifier Systems*. Springer, 2003, pp. 125–134.

[59] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. John Wiley & Sons,, 1999.

[60] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image Processing (TIP)*, vol. 15, no. 12, pp. 3736–3745, 2006.

[61] G. Field and E. Chichilnisky, "Information processing in the primate retina: circuitry and coding," *Annual Review of Neuroscience*, vol. 30, pp. 1–30, 2007.

[62] A. Foi, V. Katkovnik, and K. Egiazarian, "Pointwise shape-adaptive dct for high-quality denoising and deblocking of grayscale and color images," *IEEE Transactions on Image Processing (TIP)*, 2007.

[63] J.-J. Fuchs, "Spread representations," in *Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2011, pp. 814–817.

[64] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *International Conference on Very Large Data Bases (VLDB)*, vol. 99, 1999, pp. 518–529.

[65] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *International Conference on Machine Learning (ICML)*, 2011, pp. 513–520.

[66] L. Gomez-Chova, G. Camps-Valls, J. Munoz-Mari, and J. Calpe, "Semisupervised image classification with laplacian support vector machines," *IEEE Geoscience and Remote Sensing Letters*, vol. 5, no. 3, pp. 336–340, 2008.

[67] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2011.

[68] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *International Conference on Machine Learning (ICML)*, 2010, pp. 399–406.

[69] R. Gross, I. Matthews, J. Cohn, T. Kanade, and S. Baker, "Multi-pie," *Image and Vision Computing*, vol. 28, no. 5, pp. 807–813, 2010.

[70] S. Gu, L. Zhang, W. Zuo, and X. Feng, "Projective dictionary pair learning for pattern classification," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 793–801.

[71] Y. Gu and K. Feng, "L1-graph semisupervised learning for hyperspectral image classification," in *2012 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2012, pp. 1401–1404.

[72] C. Gulcehre, K. Cho, R. Pascanu, and Y. Bengio, "Learned-norm pooling for deep feedforward and recurrent neural networks," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2014, pp. 530–546.

[73] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 1735–1742.

[74] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5353–5360.

[75] J. R. Hershey, J. L. Roux, and F. Weninger, "Deep unfolding: Model-based inspiration of novel deep architectures," *ArXiv preprint arXiv:1409.2574*, 2014.

[76] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[77] C. Huang, L. Davis, and J. Townshend, "An assessment of support vector machines for land cover classification," *International Journal of remote sensing*, vol. 23, no. 4, pp. 725–749, 2002.

[78] L. Itti, C. Koch, E. Niebur *et al.*, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 20, no. 11, pp. 1254–1259, 1998.

[79] J. Jancsary, S. Nowozin, and C. Rother, "Loss-specific training of non-parametric image restoration models: A new state of the art," in *European Conference on Computer Vision (ECCV)*. Springer, 2012, pp. 112–125.

[80] Z. Jiang, Z. Lin, and L. S. Davis, "Learning a discriminative dictionary for sparse coding via label consistent k-svd," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2011, pp. 1697–1704.

[81] C. Jung, L. Jiao, H. Qi, and T. Sun, "Image deblocking via sparse representation," *Signal Processing: Image Communication*, vol. 27, no. 6, pp. 663–677, 2012.

[82] K. Kavukcuoglu, P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. L. Cun, "Learning convolutional feature hierarchies for visual recognition," in *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[83] K. N. Kay, T. Naselaris, R. J. Prenger, and J. L. Gallant, "Identifying natural images from human brain activity," *Nature*, vol. 452, no. 7185, pp. 352–355, 2008.

[84] K. N. Kay, J. Winawer, A. Rokem, A. Mezer, and B. A. Wandell, "A two-stage cascade model of bold responses in human visual cortex," *PLOS Computational Biology*, vol. 9, no. 5, p. e1003079, 2013.

[85] Y. Ke, X. Tang, and F. Jing, "The design of high-level features for photo quality assessment," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1. IEEE, 2006, pp. 419–426.

[86] D. Kim, S. Sra, and I. S. Dhillon, "Tackling box-constrained optimization via a new projected quasi-newton approach," *SIAM Journal on Scientific Computing*, vol. 32, no. 6, pp. 3548–3563, 2010.

[87] K. Konda, R. Memisevic, and D. Krueger, "Zero-bias autoencoders and the benefits of co-adapting features," *ArXiv preprint arXiv:1402.3337*, 2014.

[88] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[89] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[90] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 1042–1050.

[91] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[92] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. SIAM, 1974, vol. 161.

[93] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 9–48.

[94] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," *ArXiv preprint arXiv:1409.5185*, 2014.

[95] C.-P. Lee and C.-J. Lin, "A study on l2-loss (squared hinge-loss) multiclass svm," *Neural Computation*, vol. 25, no. 5, pp. 1302–1323, 2013.

[96] H. Lee, A. Battle, R. Raina, and A. Y. Ng, "Efficient sparse coding algorithms," in *Advances in Neural Information Processing Systems (NIPS)*, 2006, pp. 801–808.

[97] K. Lee, D. S. Kim, and T. Kim, "Regression-based prediction for blocking artifact reduction in jpeg-compressed images," *IEEE Transactions on Image Processing (TIP)*, vol. 14, no. 1, pp. 36–48, 2005.

[98] J. Li, J. M. Bioucas-Dias, and A. Plaza, "Semisupervised hyperspectral image segmentation using multinomial logistic regression with active learning," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 11, pp. 4085–4098, 2010.

[99] ——, "Spectral–spatial hyperspectral image segmentation using subspace multinomial logistic regression and markov random fields," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 3, pp. 809–823, 2012.

[100] W.-J. Li, S. Wang, and W.-C. Kang, "Feature learning based deep supervised hashing with pairwise labels," *ArXiv preprint arXiv:1511.03855*, 2015.

[101] X. Li, K. Zhang, and T. Jiang, "Minimum entropy clustering and applications to gene expression analysis," in *Computational Systems Bioinformatics Conference (CSB)*. IEEE, 2004, pp. 142–151.

[102] Y.-F. Li, I. W. Tsang, J. T. Kwok, and Z.-H. Zhou, "Tighter and convex maximum margin clustering," in *International Conference on Artificial Intelligence and Statistics*, 2009, pp. 344–351.

[103] M. Lin, Q. Chen, and S. Yan, "Network in network," *ArXiv preprint arXiv:1312.4400*, 2013.

[104] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 2074–2081.

[105] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *International Conference on Machine Learning (ICML)*, 2011.

[106] X. Liu, G. Cheung, X. Wu, and D. Zhao, "Inter-block consistent soft decoding of jpeg images with sparsity and graph-signal smoothness priors," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2015, pp. 1628–1632.

[107] X. Liu, X. Wu, J. Zhou, and D. Zhao, "Data-driven sparsity-based restoration of jpeg-compressed images in dual transform-pixel domain," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 5171–5178.

[108] L. Lovász and M. Plummer, *Matching Theory*. American Mathematical Society, 2009, vol. 367.

[109] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision (IJCV)*, vol. 60, no. 2, pp. 91–110, 2004.

[110] X. Lu, Z. Lin, H. Jin, J. Yang, and J. Z. Wang, "Rapid: Rating pictorial aesthetics using deep learning," in *ACM Conference on Multimedia*. ACM, 2014, pp. 457–466.

[111] W. Luo, X. Wang, and X. Tang, "Content-based photo quality assessment," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 2206–2213.

[112] Y. Lyubarskii and R. Vershynin, "Uncertainty principles and vector quantization," *IEEE Transactions on Information Theory*, vol. 56, no. 7, pp. 3491–3501, 2010.

[113] J. Mairal, F. Bach, and J. Ponce, "Task-driven dictionary learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 34, no. 4, pp. 791–804, 2012.

[114] L. Marchesotti, F. Perronnin, D. Larlus, and G. Csurka, "Assessing the aesthetic quality of photographs using generic image descriptors," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 1784–1791.

[115] J. Masci, A. M. Bronstein, M. M. Bronstein, P. Sprechmann, and G. Sapiro, "Sparse similarity-preserving hashing," *ArXiv preprint arXiv:1312.5479*, 2013.

[116] J. Masci, D. Migliore, M. M. Bronstein, and J. Schmidhuber, "Descriptor learning for omnidirectional image matching," in *Registration and Recognition in Images and Videos*. Springer, 2014, pp. 49–62.

[117] A. Mathur and G. Foody, "Multiclass and binary svm classification: Implications for training and classification users," *IEEE Geoscience and Remote Sensing Letters*, vol. 5, no. 2, pp. 241–245, 2008.

[118] H. N. Mhaskar and C. A. Micchelli, "How to choose an activation function," in *Advances in Neural Information Processing Systems (NIPS)*, 1994, pp. 319–326.

[119] H. Müller, W. Müller, D. M. Squire, S. Marchand-Maillet, and T. Pun, "Performance evaluation in content-based image retrieval: overview and proposals," *Pattern Recognition Letters*, 2001.

[120] J. Muñoz-Marí, F. Bovolo, L. Gómez-Chova, L. Bruzzone, and G. Camp-Valls, "Semisupervised one-class support vector machines for classification of remote sensing data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 8, pp. 3188–3197, 2010.

[121] N. Murray, L. Marchesotti, and F. Perronnin, "Ava: A large-scale database for aesthetic visual analysis," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 2408–2415.

[122] J. J. Nassi and E. M. Callaway, "Parallel processing strategies of the primate visual system," *Nature Reviews Neuroscience*, vol. 10, no. 5, pp. 360–372, 2009.

[123] S. A. Nene, S. K. Nayar, H. Murase *et al.*, "Columbia object image library (coil-20)," Columbia University Technical report CUCS-005-96, Tech. Rep., 1996.

[124] A. Y. Ng, M. I. Jordan, Y. Weiss *et al.*, "On spectral clustering: Analysis and an algorithm," *Advances in Neural Information Processing Systems (NIPS)*, vol. 2, pp. 849–856, 2002.

[125] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision (IJCV)*, 2001.

[126] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?" *Vision Research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[127] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression Standard*. Springer Science & Business Media, 1993.

[128] D.-S. Pham and S. Venkatesh, "Joint learning and dictionary construction for pattern recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2008, pp. 1–8.

[129] A. Plaza, J. A. Benediktsson, J. W. Boardman, J. Brazile, L. Bruzzone, G. Camps-Valls, J. Chanussot, M. Fauvel, P. Gamba, A. Gualtieri *et al.*, "Recent advances in techniques for hyperspectral image processing," *Remote Sensing of Environment*, vol. 113, pp. S110–S122, 2009.

[130] J. D. Power, A. L. Cohen, S. M. Nelson, G. S. Wig, K. A. Barnes, J. A. Church, A. C. Vogel, T. O. Laumann, F. M. Miezin, B. L. Schlaggar *et al.*, "Functional network organization of the human brain," *Neuron*, vol. 72, no. 4, pp. 665–678, 2011.

[131] F. Ratle, G. Camps-Valls, and J. Weston, "Semisupervised neural networks for efficient hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 5, pp. 2271–2282, 2010.

[132] J. A. Richards and J. Richards, *Remote sensing digital image analysis*. Springer, 1999, vol. 3.

[133] V. Roth and T. Lange, "Feature selection in clustering problems," in *Advances in Neural Information Processing Systems (NIPS)*, 2003, pp. 1–8.

[134] R. Rothe, R. Timofte, and L. Van, "Efficient regression priors for reducing image compression artifacts," in *IEEE International Conference on Image Processing (ICIP)*. IEEE, 2015, pp. 1543–1547.

[135] G. A. Rousselet, S. J. Thorpe, and M. Fabre-Thorpe, "How parallel is visual processing in the ventral pathway?" *Trends in Cognitive Sciences*, vol. 8, no. 8, pp. 363–370, 2004.

[136] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural Computation*, vol. 20, no. 10, pp. 2526–2563, 2008.

[137] D. Saad, "Online algorithms and stochastic approximations," *Online Learning*, 1998.

[138] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[139] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2003.

[140] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge university press, 2004.

[141] H. R. Sheikh, Z. Wang, L. Cormack, and A. C. Bovik, "Live image quality assessment database release 2," 2005.

[142] M.-Y. Shen and C.-C. J. Kuo, "Real-time compression artifact reduction via robust nonlinear filtering," in *International Conference on Image Processing (ICIP)*, vol. 2. IEEE, 1999, pp. 565–569.

[143] T. Sim, S. Baker, and M. Bsat, "The cmu pose, illumination, and expression (pie) database," in *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*. IEEE, 2002, pp. 46–51.

[144] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, "A sparse-group lasso," *Journal of Computational and Graphical Statistics*, vol. 22, no. 2, pp. 231–245, 2013.

[145] P. Sprechmann, A. Bronstein, and G. Sapiro, "Learning efficient sparse and low rank models," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2015.

[146] P. Sprechmann, R. Litman, T. B. Yakar, A. M. Bronstein, and G. Sapiro, "Supervised sparse analysis and synthesis operators," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 908–916.

[147] P. Sprechmann and G. Sapiro, "Dictionary learning and sparse coding for unsupervised clustering," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2042–2045.

[148] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[149] P. B. Stark and R. L. Parker, "Bounded-variable least-squares: An algorithm and applications," *Computational Statistics*, vol. 10, pp. 129–129, 1995.

[150] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua, "Ldahash: Improved matching with smaller descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 34, no. 1, pp. 66–78, 2012.

[151] C. Studer, T. Goldstein, W. Yin, and R. G. Baraniuk, "Democratic representations," *ArXiv preprint arXiv:1401.3420*, 2014.

[152] X. Sun, Q. Qu, N. M. Nasrabadi, and T. D. Tran, "Structured priors for sparse-representation-based hyperspectral image classification," *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 7, pp. 1235–1239, 2014.

[153] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International Conference on Machine Learning (ICML)*, 2013, pp. 1139–1147.

[154] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[155] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision, 1998. Sixth International Conference on*. IEEE, 1998, pp. 839–846.

[156] A. M. Treisman and G. Gelade, "A feature-integration theory of attention," *Cognitive Psychology*, vol. 12, no. 1, pp. 97–136, 1980.

[157] G. Trigeorgis, K. Bousmalis, S. Zafeiriou, and B. Schuller, "A deep semi-nmf model for learning hidden representations," in *International Conference on Machine Learning (ICML)*, 2014, pp. 1692–1700.

[158] J. K. Tsotsos and A. Rothenstein, "Computational models of visual attention," *Scholarpedia*, vol. 6, no. 1, p. 6201, 2011.

[159] Y. Wang, Y.-X. Wang, and A. Singh, "Clustering consistent sparse subspace clustering," *ArXiv preprint arXiv:1504.01046*, 2015.

[160] Y. Wang, D. Wipf, Q. Ling, W. Chen, and I. Wassail, "Multi-task learning for subspace segmentation," in *International Conference on Machine Learning (ICML)*, 2015.

[161] Y. Wang, S. Chen, and Z.-H. Zhou, "New semi-supervised classification method based on modified cluster assumption," *IEEE Transactions on Neural Networks and Learning Systems (NIPS)*, vol. 23, no. 5, pp. 689–702, 2012.

[162] Z. Wang, S. Chang, J. Zhou, M. Wang, and T. S. Huang, "Learning a task-specific deep architecture for clustering," *SIAM Conference on Data Mining (SDM)*, 2016.

[163] Z. Wang, Q. Ling, and T. Huang, "Learning deep ℓ_0 encoders," *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.

[164] Z. Wang, N. M. Nasrabadi, and T. S. Huang, "Semisupervised hyperspectral classification using task-driven dictionary learning with laplacian regularization," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 3, pp. 1161–1173, 2015.

[165] Z. Wang, Y. Yang, S. Chang, J. Li, S. Fong, and T. S. Huang, "A joint optimization framework of sparse coding and discriminative clustering," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[166] Z. Wang, D. Liu, J. Yang, W. Han, and T. Huang, "Deep networks for image super-resolution with sparse prior," *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[167] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing (TIP)*, vol. 13, no. 4, pp. 600–612, 2004.

[168] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[169] D. P. Wipf and B. D. Rao, "l0-norm minimization for basis selection," in *Advances in Neural Information Processing Systems (NIPS)*, 2004, pp. 1513–1520.

[170] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 31, no. 2, pp. 210–227, 2009.

[171] O. Wu, W. Hu, and J. Gao, "Learning to predict the perceived visual quality of photos," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 225–232.

[172] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2014.

[173] L. Xu, C. Lu, Y. Xu, and J. Jia, "Image smoothing via l 0 gradient minimization," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 6. ACM, 2011, p. 174.

[174] L. Xu, J. Neufeld, B. Larson, and D. Schuurmans, "Maximum margin clustering," in *Advances in Neural Information Processing Systems (NIPS)*, 2004, pp. 1537–1544.

[175] H. Yang, "A back-propagation neural network for mineralogical mapping from aviris data," *International Journal of Remote Sensing*, vol. 20, no. 1, pp. 97–110, 1999.

[176] J. Yang, Z. Wang, Z. Lin, X. Shu, and T. Huang, "Bilevel sparse coding for coupled feature spaces," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 2360–2367.

[177] J. Yang, K. Yu, and T. Huang, "Supervised translation-invariant sparse coding," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2010, pp. 3517–3524.

[178] Y. Yang, Z. Wang, J. Yang, J. Wang, S. Chang, and T. S. Huang, "Data clustering by laplacian regularized l1-graph," in *AAAI Conference on Artificial Intelligence*, 2014.

[179] C. Yim and A. C. Bovik, "Quality assessment of deblocked images," *IEEE Transactions on Image Processing (TIP)*, vol. 20, no. 1, pp. 88–98, 2011.

[180] G. Yuan and B. Ghanem, "l0tv: A new method for image restoration in the presence of impulse noise," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5369–5377.

[181] M. D. Zeiler, G. W. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 2018–2025.

[182] G. Zhang, Z. Jiang, and L. S. Davis, "Online semi-supervised discriminative dictionary learning for sparse representation," in *Asian Conference on Computer Vision*. Springer, 2012, pp. 259–273.

[183] B. Zhao, F. Wang, and C. Zhang, "Efficient maximum margin clustering via cutting plane algorithm," in *SIAM Conference on Data Mining (SDM)*. SIAM, 2008, pp. 751–762.

[184] ——, "Efficient multiclass maximum margin clustering," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1248–1255.

[185] M. Zheng, J. Bu, C. Chen, C. Wang, L. Zhang, G. Qiu, and D. Cai, "Graph regularized sparse coding for image representation," *IEEE Transactions on Image Processing (TIP)*, vol. 20, no. 5, pp. 1327–1336, 2011.