ABSTRACT

Title of dissertation: LEARNING ALONG THE EDGE OF DEEP NEURAL NETWORKS

Maya Kabkab
Doctor of Philosophy, 2018

Dissertation directed by: Professor Rama Chellappa
Department of Electrical and Computer Engineering

While Deep Neural Networks (DNNs) have recently achieved impressive results on many classification tasks, it is still unclear why they perform so well and how to properly design them. It has been observed that while training and testing deep networks, some ideal conditions need to be met in order to achieve impressive performance. In particular, an abundance of training samples is required. These training samples should be lossless, perfectly labeled, and spanning various classes in a balanced way. A lot of empirical results suggest that deviating from such ideal conditions can severely affect the performance of DNNs.

In this dissertation, we analyze each of these individual conditions to understand their effects on the performance of deep networks. Furthermore, we devise mitigation strategies when the ideal conditions may not be met.

We, first, investigate the relationship between the performance of a convolutional neural network (CNN), its depth, and the size of its training set. Designing a CNN is a challenging task and the most common approach to picking the right

architecture is to experiment with many parameters until a desirable performance is achieved. We derive performance bounds on CNNs with respect to the network parameters and the size of the available training dataset. We prove a sufficient condition —polynomial in the depth of the CNN— on the training database size to guarantee such performance. We empirically test our theory on the problem of gender classification and explore the effect of varying the CNN depth, as well as the training distribution and set size. Under i.i.d. sampling of the training set, we show that the incremental benefit of a new training sample decreases exponentially with the training set size.

Next, we study the structure of the CNN layers, by examining the convolutional, activation, and pooling layers, and showing a parallelism between this structure and another well-studied problem: Convolutional Sparse Coding (CSC). The sparse representation framework is a popular approach due to its desirable theoretical guarantees and the successful use of sparse representations as feature vectors in machine learning problems. Recently, a connection between CNNs and CSC was established using a simplified CNN model. Motivated by the use of spatial pooling in practical CNN implementations, we investigate the effect of using spatial pooling in the CSC model. We show that the spatial pooling operations do not hinder the performance and can introduce additional benefits.

Then, we investigate three of the ideal conditions previously mentioned: the availability of vast amounts of noiseless and balanced training data. We overcome the difficulties resulting from deviating from this ideal scenario by modifying the training sampling strategy. Conventional DNN training algorithms sample training

examples in a random fashion. This inherently assumes that, at any point in time, all training samples are equally important to the training process. However, empirical evidence suggests that the training process can benefit from different sampling strategies. Motivated by this objective, we consider the task of adaptively finding optimal training subsets which will be iteratively presented to the DNN. We use convex optimization methods, based on an objective criterion and a quantitative measure of the current performance of the classifier, to efficiently identify informative samples to train on. We propose an algorithm to decompose the optimization problem into smaller per-class problems, which can be solved in parallel. We test our approach on benchmark classification tasks and demonstrate its effectiveness in boosting performance while using even fewer training samples. We also show that our approach can make the classifier more robust in the presence of label noise and class imbalance.

Finally, we consider the case where testing (and potentially training) samples are lossy, leading to the well-known compressed sensing framework. We use Generative Adversarial Networks (GANs) to impose structure in compressed sensing problems, replacing the usual sparsity constraint. We propose to train the GANs in a task-aware fashion, specifically for reconstruction tasks. We show that it is possible to train our model without using any (or much) non-compressed data. We also show that the latent space of the GAN carries discriminative information and can further be regularized to generate input features for general inference tasks. We demonstrate the effectiveness of our method on a variety of reconstruction and classification problems.

LEARNING ALONG THE EDGE OF DEEP NEURAL NETWORKS

by

Maya Kabkab

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

Advisory Committee:
Professor Rama Chellappa, Chair/Advisor
Professor David Jacobs, Dean's Representative
Professor Behtash Babadi
Professor Larry Davis
Professor Min Wu

To my sister Lara, the strongest person I know.

Acknowledgments

It was not too long ago that I did not believe I would reach this stage of writing, let alone submitting, my Ph.D. dissertation. Yet, here I am, drafting its acknowledgement section. While my name may appear as the sole author on the cover, several remarkable individuals have inspired, supported, and contributed to this work. I would like to start by acknowledging their contributions and thanking them for their support and help in the face of adversity. They have kept me going whenever I wanted to give up.

I, first and foremost, would like to thank my advisor, Prof. Rama Chellappa, for his constant belief in my abilities and for the chance to be part of his group. I was lucky to be given the freedom to work on problems I enjoy, guided by his insight and encyclopedic knowledge. Known by many for his extremely successful career in computer vision research, he is admired for that and so much more by his students and colleagues. His unique sense of humor, approachable character, and humanity are what characterize him beyond his undeniable wisdom and success. I thank him for his patience, support, and advice, as well as his commitment to the well-being and professional development of his students. I will forever remember his words "I want students to leave my group in the same mental state they came in with... or a better one!". It is my honor to be the 96^{th} Ph.D. graduate under his supervision.

I am also fortunate to have had the support and encouragement of many people who have routinely gone beyond their duties to help me during my most difficult times. I owe Prof. Steve Marcus a great debt of gratitude for having been my

champion within the department. He never hesitated to offer a listening ear, write letters of recommendation, provide sound advice and feedback, and sometimes even intervene on my behalf. I can say with no hesitation that a large part of my degree completion is thanks to his help and encouragement. I am also grateful to Prof. Min Wu and Dr. Melanie Prange for always being available and willing to listen and help. Last but not least, I would like to thank Dr. Berk Gürakan for always being there for me. His support on both the emotional and technical levels has been unprecedented in my life. I would like to express my gratitude for his persistent effort in trying to understand my research, helping me brainstorm, proofreading my papers, and wiping my worries and anxieties. Without his encouragement, a lot of the work in this dissertation would not have materialized.

I would like to thank Prof. Larry Davis, Prof. Behtash Babadi, Prof. Min Wu, and Prof. David Jacobs for being on my dissertation committee, and for their valuable feedback.

I have greatly benefited from collaborations with Pouya Samangouei, Dr. Azadeh Alavi, and Emily Hand. Some of the work in this dissertation is a joint work with them. I am also grateful for the friendships that resulted from these collaborations.

I am thankful for the administrative help I have received through my Ph.D. journey, particularly for the assistance provided by Ms. Janice Perone, Dr. Melanie Prange, Mr. Bill Churma, Ms. Arlene Schenk, Ms. Maria Hoo, and the entire UMIACS staff.

I would like to thank all members of Prof. Chellappa's group for the friendly

environment, enjoyable group lunches, insightful discussions, and shared conference trips, with a special mention to my office-mate Boyu! Thanks to my DC area group of friends, especially Berk, Cem, Evripidis, Gianluca, Haytham, Kleoniki, Prem, Rebecca, and Sam. Meals, movies, drinks, chats, and walks with you were always a source of energy and enjoyment, and you have enriched my life in so many ways.

Finally, I express my most heartfelt gratitude to my parents Ghassan and Mona, my two sisters Eliane and Lara, my big loving family spread-out all around the globe, and my long-distance friends. Thank you for always standing by me and for your infinite and unconditional love.

This research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA R&D Contract No. 2014-14071600012. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Algorithms

FISTA	Fast Iterative Shrinkage-Thresholding Algorithm
SB	Split Bregman method
SF	Smashed Filters
TwIST	Two-step Iterative Shrinkage-Thresholding algorithm

Datasets

CelebA	CelebFaces Attributes
F-MNIST	Fashion-MNIST
GROUPS	Images of Groups
LFW	Labeled Faces in the Wild

Mathematical concepts

DCT	Discrete Cosine Transform
GD	Gradient Descent
i.i.d.	Independent and Identically Distributed
KL divergence	Kullback-Leibler divergence
LP	Linear Program
p.d.f.	Probability Distribution Function
REC	Restricted Eigenvalue Condition
RIP	Restricted Isometry Property
SDP	Semi-Definite Program
TV	Total Variation
VC dimension	Vapnik-Chervonenkis dimension

Models, methods, and networks

CNN	Convolutional Neural Network
CSC	Convolutional Sparse Coding
CSGAN	Compressed Sensing Generative Adversarial Networks
DCGAN	Deep Convolutional Generative Adversarial Networks
DNN	Deep Neural Network
DSC	Deep Sparse Coding
DSCP	Deep Sparse Coding with Pooling
GAN	Generative Adversarial Networks
LBP	Local Binary Patterns
NN	Nearest Neighbor
ReLU	Rectified Linear Unit
VAE	Variational Auto-Encoder

Other

CPU	Central Processing Unit
GPU	Graphics Processing Unit
NC	(Number of) Non-Compressed training samples
UAV	Unmanned Aerial Vehicle
w.r.t.	With Respect To

Chapter 1: Introduction

1.1 Motivation

In recent years, deep neural network approaches have been widely adopted for machine learning tasks, including classification [1, 2]. However, many important questions remain as open problems, such as why they perform so well, how to properly design them, how they work, and their limitations. In this dissertation, we attempt to obtain a better understanding of deep networks using techniques from statistical learning and sparse coding, and push the limits of these networks by straying away from ideal train-time and inference-time conditions. While we mostly limit ourselves to classification problems, most of the techniques we develop can be extended for other supervised learning tasks such as verification.

In an ideal scenario, a deep network is trained using an *abundance* of training samples. These training samples are *complete* (i.e., not lossy), *perfectly* labeled, and all classes are *balanced* and equally represented. At inference time, *no adversary* can manipulate the input to the network.

1.2 Outline

We begin this dissertation by seeking a deeper understanding of this ideal scenario. We focus on Convolutional Neural Networks (CNNs), which are forms of deep networks most often used for computer vision classification problems [2]. Our objective is to answer the following questions: In the case of CNNs, what is an *abundance* of training samples? How do we a pick a suitable network depth? When designing a CNN, the most common approach is to experiment with the depth (and many other parameters), until a suitable model is found. It is known that if the CNN is *too shallow*, then it may not correctly represent the underlying relationship between the input and its corresponding class (i.e., under-fit). If it is *too deep*, however, it may follow irrelevant properties of the dataset on which it is trained (i.e., over-fit). In Chapter 2 of this dissertation, we derive bounds on the performance of CNNs as a function of the network parameters and the number of available training data points. This can be insightful when trying to design a CNN architecture. While deriving this bound, we assume that the training data is sampled in an i.i.d. fashion, which is usually the case in most practical applications. We show that the number of examples sufficient for a desired generalization performance grows polynomially with the complexity of the network model. This bound can also be used to find the incremental benefit that one new random example adds to the generalization performance. We show that, under i.i.d. sampling, this benefit decreases exponentially with the training set size.

Keeping our focus on investigating the ideal training and inference conditions

for deep networks, we offer a different insight into CNNs in Chapter 3, this time examining the design of their layers. Our work in this chapter is based on the surprising connection between CNNs and Convolutional Sparse Coding (CSC), which was made in [3]. Specifically, it was shown that the *forward pass* of the CNN, which computes the representation of an input vector, is well approximated by a series of CSC steps, which we will define later in the dissertation. However, the CNN model considered in [3] does not include the *spatial pooling* operation, widely used in the best-performing CNN architectures [1, 2]. In this chapter, we investigate the theoretical benefits of adding spatial pooling layers after the CSC steps. Our work bridges the gap between the deep CSC model proposed in [3] and CNN architectures used in practice. In addition to the known benefits (such as translation invariance [4, 5]), we show that inserting pooling layers does not cause loss in performance while decreasing the dimensionality of the involved vectors and introducing additional benefits such as noise suppression and preventing codes from becoming too sparse. This offers some justification to the most commonly used and successful CNN architectures which often consist of stacking convolutional filters, rectified linear unit (ReLU) activations, and spatial (usually max) pooling [6, 7].

As previously mentioned, under the i.i.d. sampling of training examples, we witness an exponential decrease in the incremental benefit of a data point. To counteract this effect it is clear that the i.i.d. sampling requirement needs to be changed. In Chapter 4, we present techniques for a judicious selection of new training examples by exploiting the cumulative knowledge gained by the network from previous examples. Specifically, rather than passively accepting examples

generated by some unknown external distribution, we seek to actively and iteratively find optimal subsets of training examples to present to the network. Our results indicate that careful selection and ordering of training samples can lead to improved performance, compared to sampling training data at random. We also show that our approach makes the classifier more robust in the presence of label noise and class imbalance. This chapter addresses the limitations of deep networks when one or more of the following three ideal conditions is not met: abundance of training data, clean training labels, and class balance. It also develops solutions to overcome these problems.

In Chapter 5, we relax the completeness assumption on the network input, and assume that, at inference time (and potentially training time as well), samples are lossy or *compressed*. This leads us to the field of compressed sensing which has impressive applications, such as rapid magnetic resonance imaging [8], single-pixel camera [9] and UAV systems, among others. The core problem of compressed sensing is that of efficiently reconstructing a signal from an under-determined linear system of noisy measurements. In this chapter, we extend the work in [10], in which a *generative model* was used and the unknown signal was assumed to be the output of this model. We propose to train the generative model specifically for the task of recovering compressed measurements. This makes it task-aware, and improves the compressed sensing performance. Our approach also addresses the case where no or very little non-compressed data is available for training, by complementing the training set with compressed training data.

The last ideal condition for training and deploying a deep network classifier

deals with the absence of an adversary which can manipulate inputs at inference time. In fact, deep neural networks have been shown to be susceptible to *adversarial attacks* [11, 12]. These attacks come in the form of *adversarial examples*: carefully crafted perturbations added to a legitimate input sample. In the context of classification, these perturbations cause the legitimate sample to be misclassified at inference time [11–14]. Such perturbations are often small in magnitude and do not affect human recognition but can drastically change the output of the classifier. An approach very similar to the one adopted in Chapter 5 can be used to diminish the effect of the adversarial perturbation. We can leverage the representative power of Generative Adversarial Networks (GANs) [15] by projecting input images onto the range of the GAN's generator prior to feeding them to the classifier. We expect that legitimate samples will be close to some point in the range of the generator, whereas adversarial samples will be further away from it. Furthermore, "projecting" the adversarial examples onto the range of the generator can have the desirable effect of reducing the adversarial perturbation. The projected output, computed using Gradient Descent (GD), is fed into the classifier instead of the original (potentially adversarially modified) image. This work will not be discussed in this dissertation but we refer the interested reader to [16] for an in-depth discussion of this defense method.

1.3 Contributions

- In Chapter 2, we derive performance bounds on CNNs with respect to the network parameters and the size of the available training dataset.
  - We prove a sufficient condition, polynomial in the depth of the CNN, on the training database size to guarantee a certain performance.
  - We extend the bound to the case where the training and testing distributions are slightly different, and show how it changes as the two distributions diverge.
- In Chapter 3, we investigate the theoretical benefits of spatial pooling layers.
  - We show that spatial pooling layers, while being lossy operations, do not hinder performance.
  - We also demonstrate that the addition of pooling can introduce additional benefits such as reducing the dimensionality of involved vectors, slight translation invariance, and noise suppression.
  - Finally, we show that the pooling layers allow CNNs (and CSC models) to be very deep by preventing codes from becoming too sparse.
- In Chapter 4, we present strategies to make optimal use of the available training data. We introduce an adaptive selection algorithm to choose batches of training samples which meet four criteria: class balance, diversity, representativeness, and classifier uncertainty.

  - We propose a novel class balancing algorithm which uses feedback from the classifier to allot a subset of training samples to each class.
  - We use convex optimization techniques to identify a per-class near-optimal batch which meets the remaining three criteria.
  - We empirically show that this selection method leads to improved performance of CNNs.
  - Our results also suggest that using our selection algorithm can add robustness in the case of class imbalance and label noise in the training data.

- In Chapter 5, we introduce techniques to reconstruct lossy images using GANs.
  - We train a GAN in a task-aware fashion allowing it to be specifically optimized for the reconstruction task. We show that this consistently improves the reconstruction error compared to state-of-the-art methods.
  - We introduce a novel algorithm to train the GAN using a combination of a small number of (or no) non-lossy data and a larger set of lossy training data.
  - We show that the latent space of the GAN can be regularized and used for various supervised inference tasks.

Chapter 2: On the size of convolutional neural networks and generalization performance

2.1 Overview

Convolutional Neural Networks (CNNs) are now widely used for classification problems due to their state-of-the-art performance (see, e.g., [1, 2]). However, one important challenge, which remains an open problem, is how to size them appropriately. In this chapter, we try to address this problem by investigating the relationship between the depth of a CNN and its generalization performance using approaches from statistical learning theory.

Recently, CNNs have drawn much needed attention, and a lot of empirical work has attempted to understand why they perform so well [17, 18] as well as how to properly design them [19, 20]. However, from a theoretical perspective, CNNs are still not completely understood. While theoretical results on deep architectures exist [21–24], they are almost always restricted to feedforward neural networks.

In this chapter, we investigate the effect of CNN depth on its generalization performance. Specifically, we ask the question of how to pick a suitable CNN depth given a training database size. We assume that the examples are drawn according

to an arbitrary, fixed, probability distribution, and that the learning algorithm will produce a CNN which will correctly predict on a substantial fraction of the training set. We are concerned with how the same CNN will perform on unseen (testing) samples, drawn from the same, or a slightly different, distribution. Our work is based on the VC dimension, which was first introduced in [25, 26] and provides a mathematical foundation for answering such questions. We follow an approach similar to [21], which is specific to feedforward networks, but extend it for the case of CNNs. We restrict our study to the problem of binary classification in which the set of possible labels contains only two elements, e.g., 0 and 1.

We show that, if the training and testing sampling distributions are the same, a sufficient condition to guarantee valid generalization is for the CNN training set size to be some constant times d^4 where d is the depth of the convolutional layers. We also show how to generalize the condition when the training and testing distributions are different. We empirically demonstrate that these conditions are sufficient but often not necessary, and examine the behavior of the testing error as we vary the CNN depth, as well as the training distribution and set size.

The chapter is organized as follows: Section 2.2 introduces the CNN model architecture under consideration, Section 2.3 develops the mathematical framework as well as the theoretical results, and finally, Section 4.3 provides experimental results on the binary problem of gender classification.

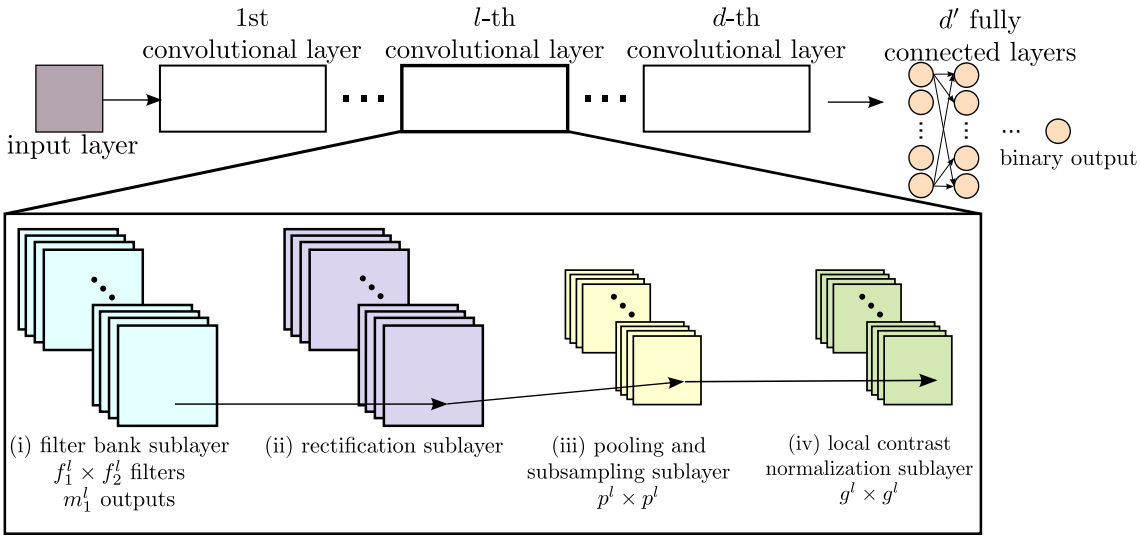

Figure 2.1: Model architecture.

2.2 Model architecture

In this work, we consider a model architecture similar to the one presented in [7]. As shown in Figure 2.1, a CNN of depths (d, d') consists of d convolutional layers and d' fully connected layers. The l-th layer of a CNN is composed of the following:

(i) *a filter bank sublayer*, which takes as input $\mathbf{x}^l$, a 3D array with n_1^l 2D feature maps of size $n_2^l \times n_3^l$ each, and outputs a 3D array with m_1^l 2D feature maps of size $m_2^l \times m_3^l$ each. The size of the output maps is determined by the size $f_1^l \times f_2^l$ of the convolution filters and is given by $m_2^l = n_2^l - f_1^l + 1$ and $m_3^l = n_3^l - f_2^l + 1$. Filter k_{ij}^l connects the i-th input feature map $\mathbf{x}_i^l$ to the j-th output feature

map $\mathbf{y}_j^l$:

$$\mathbf{y}_j^l = a_j^l \cdot \tanh\left(\sum_i k_{ij}^l * \mathbf{x}_i^l\right). \tag{2.1}$$

The filters and the coefficients $\{a_j^l\}$ are trainable parameters.

(ii) *a rectification sublayer*, which only retains positive inputs:

$$\bar{y}_{ijk}^l = \max\left\{0, y_{ijk}^l\right\}. \tag{2.2}$$

(iii) *a pooling and subsampling sublayer*, which keeps the maximum (or the average) from each $p^l \times p^l$ window and outputs $\bar{\bar{\mathbf{y}}}^l$.

(iv) *a local contrast normalization sublayer*, which performs the following operations:

$$v_{ijk}^l = \bar{\bar{y}}_{ijk}^l - \sum_{i,p,q} w_{pq} \cdot \bar{\bar{y}}_{i,j+p,k+q}^l, \tag{2.3}$$

where w is a Gaussian window of size $g^l \times g^l$, then

$$\bar{\bar{y}}_{ijk}^l (= x_{ijk}^{l+1}) = \frac{v_{ijk}^l}{\max\left\{\mu^l, \sigma_{jk}^l\right\}} \tag{2.4}$$

where $\sigma_{jk}^l = \sum_{ipq} w_{pq} \cdot \left(v_{i,j+p,k+q}^l\right)^2$ and $\mu^l = \text{mean}\left(\sigma_{jk}^l\right)$.

The d' fully connected layers have a fixed structure and trainable weights $\mathbf{W}_f$. In the rest of the chapter, we will assume that d' is fixed and study the effect of varying d on the classifier's generalization performance. As mentioned earlier, we restrict

our study to binary classification, i.e., CNNs which implement a function that maps samples from the input domain I, to a boolean value in $\{0, 1\}$.

2.3 Relationship between depth and generalization performance

2.3.1 Problem formulation

In this chapter, we are interested in characterizing how the depth of a CNN affects its generalization performance. Formally, we let $\mathcal{C}^d$ be the set of convolutional neural networks with d convolutional layers, for some fixed values of $\{n_1^l, n_2^l, n_3^l, m_1^l, f_1^l, f_2^l, p^l, g^l\}_{l=1}^d$, as defined in section 2.2 above. This set includes all such CNNs realized by varying the parameters $\{a_j^l, k_{ij}^l\}_{i,j,l} \cup \{\mathbf{W}_f\}$. As with any supervised learning algorithm, a CNN learning algorithm starts with a training set $S = \{x_1, x_2, \ldots, x_{|S|}\} \subseteq I$, assumed to be drawn at random according to a fixed but arbitrary distribution $\mathcal{D}_S$ on the input domain I. The aim of the algorithm is to find a suitable CNN $c \in \mathcal{C}^d$ which agrees with the *ground truth*, or target, hypothesis $h^* : I \to \{0, 1\}$ as much as possible. It is assumed that the true labels of the training samples, i.e., $h^*(x_1), h^*(x_2), \ldots, h^*(x_{|S|})$, are known. As such, the resulting CNN c will have an empirical training error given below:

$$\hat{e}_S(c) \triangleq \frac{1}{|S|} \sum_{i=1}^{|S|} \mathbb{1}\left(h_c(x_i) \neq h^*(x_i)\right), \tag{2.5}$$

where $\mathbb{1}(.)$ is the indicator function and $h_c(.)$ is the boolean function implemented by the CNN c. Clearly, $\hat{e}_S(c)$ is a random variable since the set S is chosen at random.

However, if the learning algorithm is designed properly, $\hat{e}_S(c)$ will tend to be small. This does not, however, provide any guarantee as to how the CNN classifier will perform on test samples. We assume that testing samples are drawn at random according to a distribution $\mathcal{D}_T$. We are thus interested in the average performance of c on these new samples:

$$e_T(c) \triangleq \Pr_{\mathcal{D}_T}\left[h_c(x) \neq h^*(x)\right], \tag{2.6}$$

where x is a random sample picked according to $\mathcal{D}_T$.

2.3.2 Same training and testing distribution

We first look at the case when the training and testing sampling distributions are the same, i.e., $\mathcal{D}_S \sim \mathcal{D}_T$. As previously stated, a CNN c (or its corresponding boolean function $h_c(.)$), which is accurate on the training set (i.e., has small $\hat{e}_S(c)$), might not necessarily be accurate on new examples which are not in the training set, even if the new examples are drawn from the same distribution. In this case, we are interested in performance guarantees on $e_T(c) = e_S(c)$, whenever $\hat{e}_S(c)$ is small. To this end, we first state Lemma 2.1 which computes the *VC dimension* of CNNs of convolutional depth d. The VC dimension of a set of binary functions, is the maximum number m of vectors which can be separated into two classes in all 2^m ways using functions from the set [27].

Lemma 2.1 *Let* $\mathcal{H}^d \triangleq \left\{h_c : I \to \{0,1\} \mid c \in \mathcal{C}^d\right\}$ *be the set of boolean functions*

implementable by all CNNs in $\mathcal{C}^d$*, and* $q(d) \triangleq \sum_{l=1}^{d} m_1^l \cdot (n_2^l - f_1^l + 1) \cdot (n_3^l - f_2^l + 1) \cdot (n_1^l n_2^l n_3^l + m_1^l (g^l)^2 + (p^l)^2)$*. Then, the VC dimension of the class of CNNs defined in section 2.3.1 above satisfies*

$$\text{VCdim}\left(\mathcal{H}^d\right) \leq \alpha \left(d \cdot q(d)\right)^2, \tag{2.7}$$

for some constant α*.*

Proof: The proof of this Lemma, and all other proofs in this dissertation, can be found in the Appendix. ■

We now state the following theorem on the CNN generalization performance guarantees:

Theorem 2.1 *For any* $0 < \delta < 1$*,* $\epsilon > 0$*,* $0 < \gamma \leq 1$*, if* S *is chosen at random according to the distribution* $\mathcal{D}_S$*, such that*

$$|S| \geq \frac{8}{\gamma^2 \epsilon} \max\left\{\ln \frac{8}{\delta},\ 2\alpha \left(d \cdot q(d)\right)^2 \ln \frac{16}{\gamma^2 \epsilon}\right\}, \tag{2.8}$$

then, with probability at least $1 - \delta$*, for every* $c \in \mathcal{C}^d$*, one of the following will hold:*

(i) $\hat{e}_S(c) > (1 - \gamma)\epsilon$*,*

(ii) $e_T(c) = e_S(c) \leq \epsilon,\ \hat{e}_S(c) \leq (1 - \gamma)\epsilon$*.*

Theorem 2.1 implies that if condition (2.8) is met, and if the trained CNN c is such that $\hat{e}_S(c)$ is as small as desired, then we know that, with high probability, c will exhibit good generalization performance. Let $M = \max_{l=1,\ldots,d} \{m_1^l \cdot (n_2^l - f_1^l + 1) \cdot (n_3^l -$

$f_2^l + 1) \cdot (n_1^l n_2^l n_3^l + m_1^l (g^l)^2 + (p^l)^2)\}$, then $q(d) \leq M \cdot d$. From (2.8), we see that, for proper generalization, the training sample size should be larger than $M' \cdot d^4$ where $M' = M^2 \alpha \cdot \frac{16}{\gamma^2 \epsilon} \cdot \ln \frac{16}{\gamma^2 \epsilon}$. Conversely, when designing a CNN, given a fixed training set size $|S|$, we know that the CNN is very likely to exhibit good generalization performance if the depth of the convolutional layers is less than $\sqrt[4]{\frac{|S|}{M'}}$. We also state a converse to Theorem 2.1:

Theorem 2.2 *For any learning algorithm which uses a training sample set S of size*

$$|S| \leq \frac{\text{VCdim}\left(\mathcal{H}^d\right) - 1}{2e\epsilon} \tag{2.9}$$

(where e denotes the base of the natural logarithm), there exists a CNN $c \in \mathcal{C}^d$ and a distribution $\mathcal{D}$ such that the expected error of c (w.r.t. $\mathcal{D}$) is at least ϵ.

2.3.3 Different training and testing distributions

In section 2.3.2 above, we addressed the question of when a CNN is expected to generalize from $|S|$ training examples chosen according to an arbitrary probability distribution $\mathcal{D}_S$, assuming that test examples are drawn from the same distribution. In this section, we relax this assumption and allow the training and testing distributions to be different, $\mathcal{D}_S$ and $\mathcal{D}_T$, respectively. To this end, we define the variation divergence between the two distributions [28]:

$$\tau \triangleq 2 \sup_{B \in \mathcal{B}} \left| \Pr_{\mathcal{D}_S}[B] - \Pr_{\mathcal{D}_T}[B] \right|, \tag{2.10}$$

where $\mathcal{B}$ is the set of measurable subsets under $\mathcal{D}_S$ and $\mathcal{D}_T$. While we allow the two distributions to be different, our hope is that they are not *too* different so that learning from $\mathcal{D}_S$ is still somehow relevant for testing on $\mathcal{D}_T$. We now reformulate Theorem 2.1 for the case when $\tau \neq 0$:

Theorem 2.3 *Let $0 < \delta' < 1$, $\epsilon' > \tau$, $0 < \gamma' \leq 1$. If the training and testing sets are chosen independently at random according to the distributions $\mathcal{D}_S$ and $\mathcal{D}_T$, respectively, such that*

$$|S| \geq \frac{8}{\bar{\gamma}^2 (\epsilon' - \tau)} \max \left\{ \ln \frac{16}{\delta'}, 2\alpha (d \cdot q(d))^2 \ln \frac{16}{\bar{\gamma}^2 (\epsilon' - \tau)} \right\}, \tag{2.11}$$

where

$$\bar{\gamma} = \gamma' \cdot \left(1 + \frac{\tau}{\epsilon' - \tau} \right) - \frac{\tau}{\epsilon' - \tau}, \tag{2.12}$$

then, with probability at least $1 - \delta'$, for every $c \in \mathcal{C}^d$, one of the following will hold:

(i) $\hat{e}_S(c) > (1 - \gamma')\epsilon'$,

(ii) $e_T(c) \leq \epsilon'$, $\hat{e}_S(c) \leq (1 - \gamma')\epsilon'$.

Note that Theorem 2.3 requires that $\epsilon' > \tau$. As mentioned earlier, we are interested in the case when τ is small so that the learning is still useful. If $\tau \ll \epsilon'$, then $\bar{\gamma} \approx \gamma'$ and (2.8) and (2.11) are very close. When τ increases, so does the lower bound on $|S|$. This is to be expected, as we are looking at learning from and testing on two very different distributions.

2.4 Experimental results

While section 2.3 gives some insight as to how to design CNNs which exhibit desirable generalization performance, it has been shown that neural networks tend to perform well with training sets which are smaller than required by the VC dimension bounds [23]. We therefore attempt to gain a better and more practical understanding of the problem by designing experiments for gender classification of face images. To this end, we use three different datasets: Images of Groups (GROUPS) [29], Labeled Faces in the Wild (LFW) [30], and Facetracer [31]. We resize face images to 64×64 and normalize them using histogram equalization to correct changes in brightness. We then use mean-subtraced normalized face images to train CNNs of convolutional depths 3, 4, and 5. Once the CNN is trained, we classify new face images by resizing and normalizing them, then applying the learned model to them. We use the Caffe framework [32] to train and test the CNNs.

2.4.1 Method

For each depth $d = 3, 4, 5$, we select uniform random subsets of varying sizes from each training dataset. Since, as noted in Section 2.3.2, a sufficient training sample size which guarantees good generalization is proportional to d^4, we choose the random training subsets to have sizes $|S| = \beta \cdot d^4$ for different values of β. Then, for each depth, dataset, and training subset size, we train a CNN (starting from a random weight initialization) until we reach a training error $\hat{e}_S(c) < 0.05$. We then test the resulting CNN on a testing set T in order to estimate $e_T(c)$. For the

case when the testing and training distributions are the same, we perform 5-fold cross-validation using the protocol specified in [33] for LFW and GROUPS, and five random splits for Facetracer. We also perform cross-dataset testing, training on subsets of one dataset and testing on the other two.

2.4.2 Architectures

As mentioned in section 2.2, each convolutional layer of the CNN is composed of a filter bank sublayer, a rectification sublayer, a pooling and subsampling sublayer, and a local contrast normalization sublayer. The rectification sublayers have no parameters. All pooling sublayers are max-pooling and use 3×3 windows. All local contrast normalization sublayers use 5×5 windows, except for the first one, which uses 7×7 windows. The first layer's filter bank sublayer consists of a 15×15 convolution mask applied every 3 pixels, resulting in 96 feature maps. The second filter bank sublayer has 5×5 convolution filters with 256 output maps. The third sublayer uses 3×3 kernels with 384 feature map outputs. When needed, the fourth and fifth filter bank sublayers also use 3×3 kernels, with 512 and 384 output maps, respectively. The convolutional layers are followed by three fully connected layers. The first two have 4096 outputs and are each followed by rectification and a 50% dropout. The last fully connected layer has two outputs. We do not attempt to optimize the architecture of the CNNs and keep it fixed throughout the experiments, only varying the convolutional depth d.

2.4.3 Results

Since the designed CNNs have different training errors, comparing their testing accuracies would not be very informative. Instead, we consider the difference between the testing and training errors. When dataset D1 is used for training and dataset D2 for testing, we denote this difference by $\Delta_{\text{D1,D2}}$.

2.4.3.1 Same training and testing distribution

In the case of the same training and testing distribution, we take the average across the five cross-validation tests. In general, and as expected, we notice that $\Delta_{\text{D1,D1}}$ decreases with the training set size. For instance, Figure 2.2 plots, for depths $d = 3, 4, 5$, $\Delta_{\text{LFW,LFW}}$ and $\Delta_{\text{GROUPS,GROUPS}}$ vs. the training set size $|S|$ (in logscale). We note that, when plotted against $|S|$, $\Delta_{\text{LFW,LFW}}$ behaves similarly for depths 3 and 4, and the CNNs actually achieve good generalization performance for relatively small training set sizes. For example, to have $\Delta_{\text{LFW,LFW}} \leq 0.05$, $|S|$ should only be greater than about 1500. This is much smaller than the bound given in Theorem 2.1 which is actually very large (in fact, even for $d = 1$, $q(1)$ is larger than the total number of images in GROUPS and Facetracer). It also seems to be the same for both $d = 3$ and $d = 4$, which is contrary to what was expected. For $d = 5$, slight over-fitting seems to take place, and larger training set sizes are needed to achieve similar generalization performance as in shallower networks. As seen in the bottom plot, we observe a similar behavior with GROUPS but the over-fitting is apparent starting from $d = 4$. As previously mentioned, while shown to be tight in Theorem 2.2, bounds based on

the VC dimension tend to be very large as they provide generalization performance guarantees regardless of the underlying probability distribution on the training and testing examples, and of the training algorithm used [34]. In fact, Figure 2.3 shows that while the CNN performance does generally improve with larger training sets, other aspects, especially the sample distribution, have a considerable effect. For example, the results seem to suggest that CNNs perform better on LFW gender classification than on GROUPS gender classification. The training algorithm is also important as it can restrict the set of realizable CNNs to a subset of $\mathcal{C}^d$. Our training algorithm uses dropout in the fully connected layers. Dropout is a very well known technique to reduce overfitting in deep neural networks [35]. However, CNNs with dropout and without dropout have the same VC dimension and therefore share the same bounds in Theorem 2.1. Since dropout has become almost standard in state-of-the-art CNN implementations, we chose to only carry out experiments using it. However, we naturally expect the over-fitting behavior to be much more prominent for deep CNNs which do not use dropout.

2.4.3.2 Different training and testing distributions

Theorem 2.3 suggests that more training samples are needed for cross-dataset testing in order to achieve the same generalization performance compared to when the training and testing samples have the same distribution. This is shown to be clearly the case in Figure 2.8. In the top figure, we see that, for depth 3, to achieve $\Delta_{\text{D1,D2}} < 0.2$, for D2 = Facetracer, we need $|S|$ to be greater than 105, 1000, and

1300, for D1 = Facetracer, LFW, and GROUPS, respectively. Figure 2.8 also shows that training using the LFW dataset seems to be more "relevant" for testing on Facetracer. This suggests that the variation divergence τ between the underlying distributions of Facetracer images and LFW images could be smaller than that between the distributions of Facetracer and GROUPS images. However, τ cannot be accurately estimated from finite samples of distributions [28]. We therefore seek a different approach to quantify the distance between the distributions. We consider the method proposed in [36] to estimate the KL divergence between distributions based on k-th nearest neighbor distances. The KL divergence is a non-symmetric measure of the difference between two probability distributions. According to [36], given $\{X_1, \ldots X_m\}$ and $\{Y_1, \ldots, Y_p\}$ n-dimensional samples drawn according to two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively, the KL divergence estimate is given by:

$$\hat{D}(\mathcal{D}_1||\mathcal{D}_2) = \frac{n}{m}\sum_{i=1}^{m} \ln \frac{\nu_k(i)}{\rho_k(i)} + \ln \frac{p}{m-1}, \tag{2.13}$$

where $\nu_k(i)$ is the distance between X_i and its k-th nearest neighbor in $\{Y_j\}$, and $\rho_k(i)$ is the distance between X_i and its k-th nearest neighbor in $\{X_j\}_{j\neq i}$. The choice of k trades off bias and variance. While it is true that the number of images available is relatively small compared to their dimension ($64 \times 64 \times 3$) and therefore, the KL divergence estimates are not very accurate, we notice that both $\hat{D}(\mathcal{D}_{\text{Facetracer}}||\mathcal{D}_{\text{LFW}})$ and $\hat{D}(\mathcal{D}_{\text{LFW}}||\mathcal{D}_{\text{Facetracer}})$ are consistently smaller (by a factor of around 3) than $\hat{D}(\mathcal{D}_{\text{Facetracer}}||\mathcal{D}_{\text{GROUPS}})$ and $\hat{D}(\mathcal{D}_{\text{GROUPS}}||\mathcal{D}_{\text{Facetracer}})$ for different values of k ranging from 1 to 10. This difference could explain why, when tested on Facetracer, CNNs

trained using LFW perform better than those trained using GROUPS.

In the bottom plot of Figure 2.8, we notice an over-fitting trend for the cross-dataset case at depth 4. This is in contrast with the findings when the training and testing samples have the same distribution. We investigate this on a different dataset (LFW) and across depths 3, 4 and 5. The results are shown in Figure 2.11. In the top figure, we see that the generalization performance tends to become worse as the depth increases, especially for models trained on GROUPS. In the bottom figure, the x-axis is changed to β (where $|S| = \beta \cdot d^4$) and we notice that for large β (> 10), models trained on GROUPS behave similarly across depths. This means that, if to achieve a certain generalization performance, a training set size $\beta \cdot 3^4$ is needed for CNNs of depth 3, then approximately $\beta \cdot 4^4$ and $\beta \cdot 5^4$ training samples are needed to achieve the same level of performance for CNNs of depths 4 and 5, respectively. It seems that, in this case, the number of samples needed for good generalization scales with d^4 as predicted by the theoretical bound (albeit with a smaller multiplicative constant). We found similar trends when testing on the GROUPS and Facetracer datasets as shown in the following figures.

2.5 Concluding remarks

In this chapter, we extended various statistical learning theorems to characterize the relationship between the depth of a CNN, the size of the training set, and the generalization performance. We proved that whenever the training and testing sampling distributions are the same, if the training set size is some constant times

d^4, then the CNN will, with high probability, exhibit good generalization. We also showed that this bound increases when the training and testing distributions are different, and characterized it as a function of the variation divergence between the distributions. We then implemented deep CNNs for the problem of gender recognition on three well-known datasets. We empirically demonstrated that over-fitting tends to occur for very deep networks, which will require larger training sets to achieve generalization performance similar to shallower versions. This is especially the case when the training and testing distributions are different.

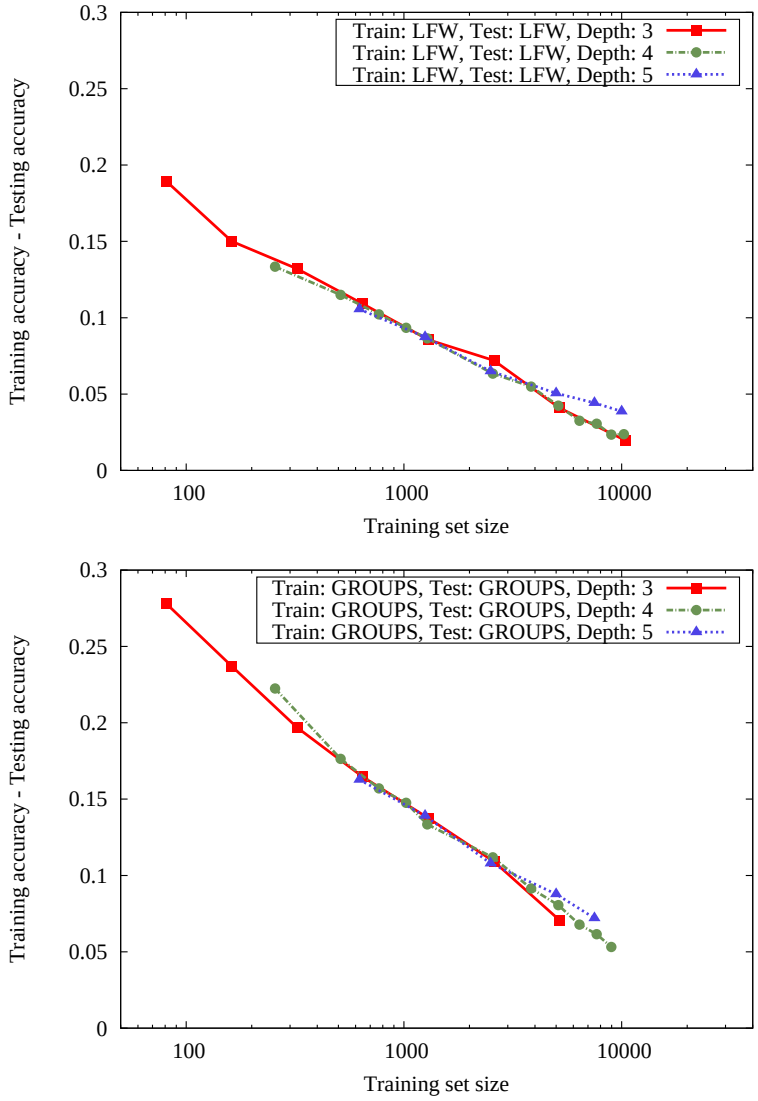

Figure 2.2: Generalization performance of CNNs trained and tested on LFW (top) and GROUPS (bottom).

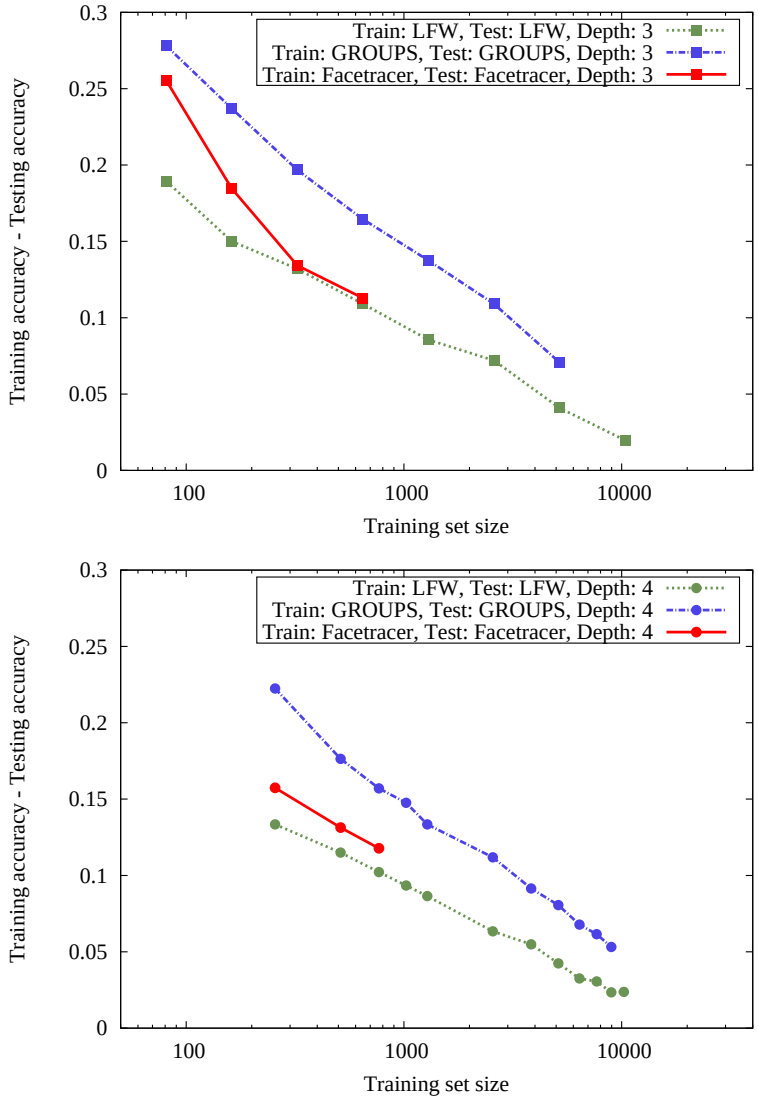

Figure 2.3: Generalization performance of CNNs of depths 3 (top) and 4 (bottom) trained and tested on the same datasets.

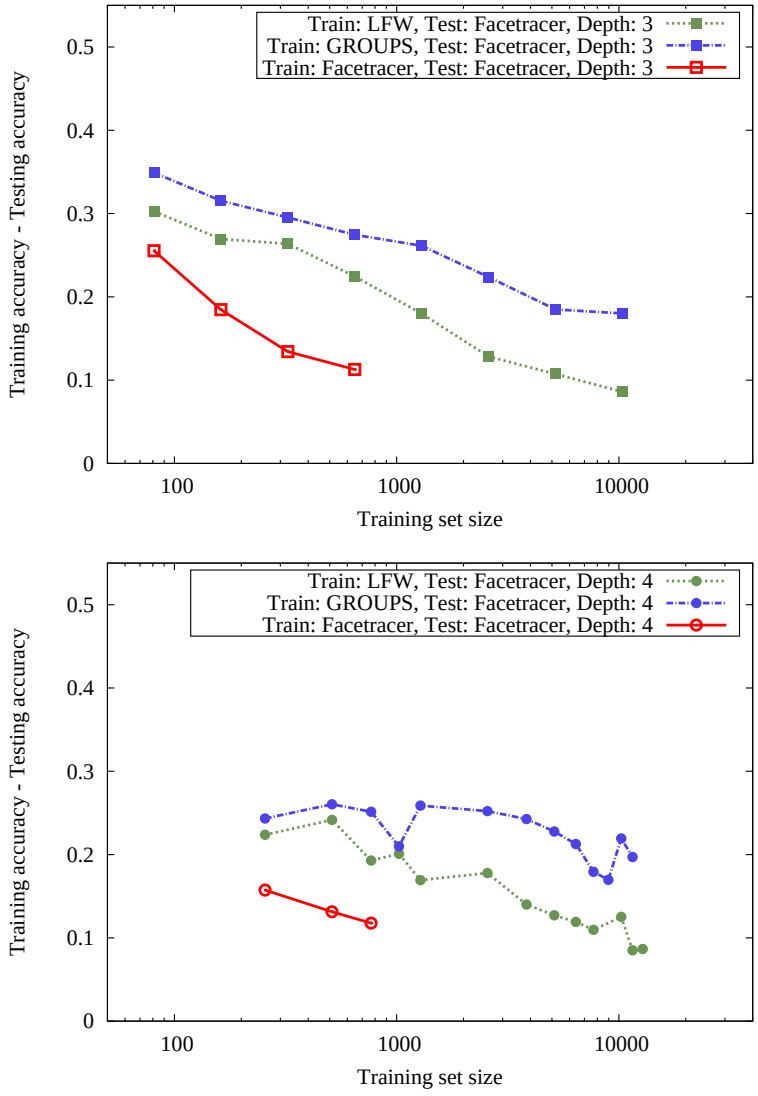

Figure 2.4: Generalization performance of CNNs of depths 3 (top) and 4 (bottom) tested on Facetracer and trained on different datasets.

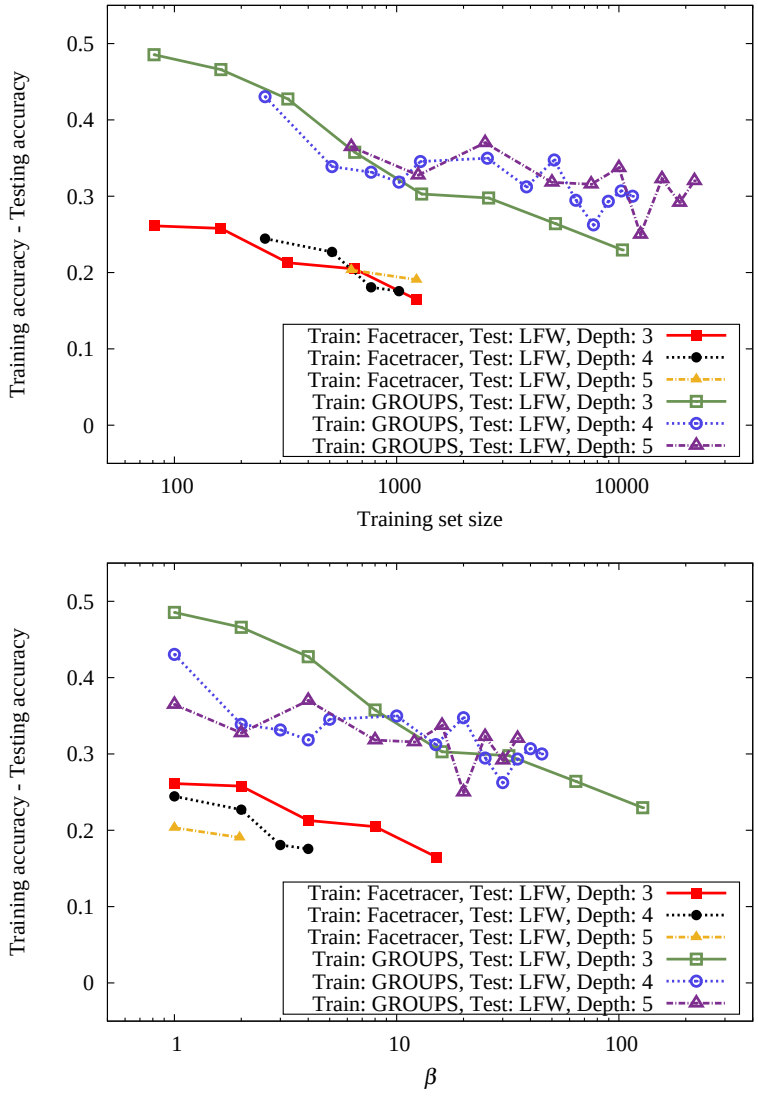

Figure 2.5: Generalization performance of CNNs tested on LFW and trained on different datasets.

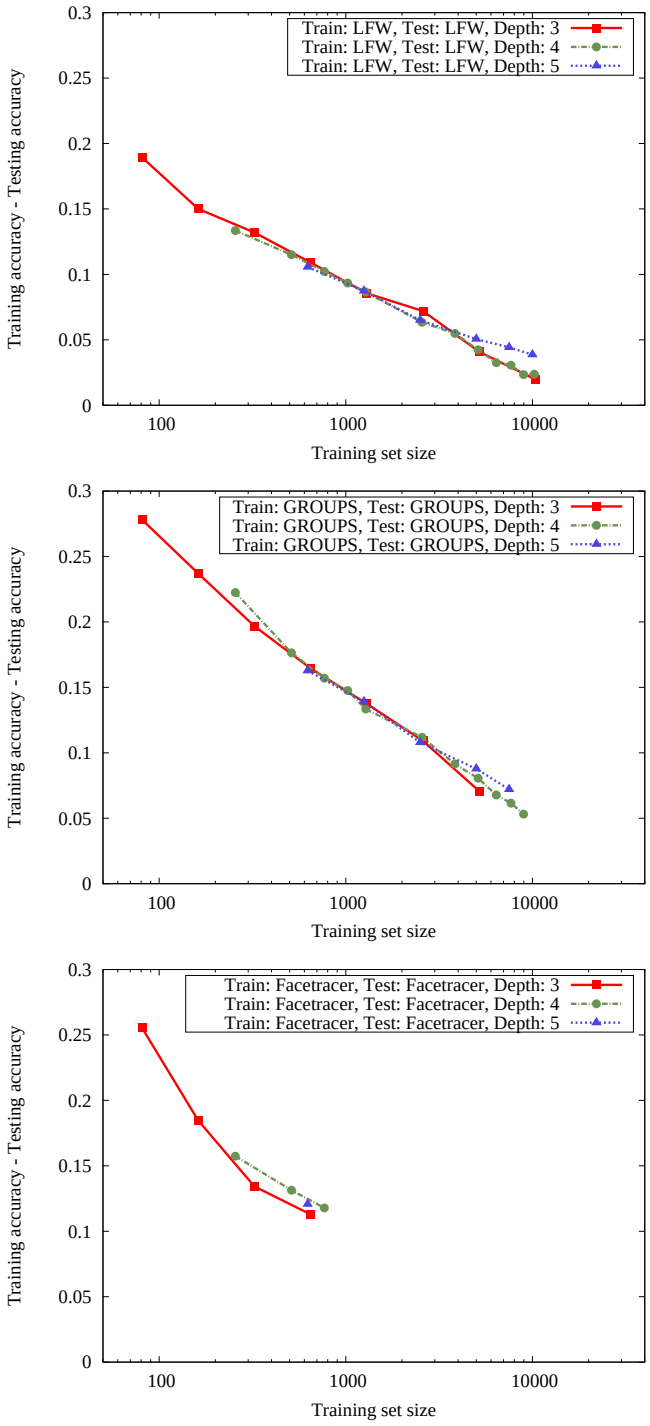

Figure 2.6: Generalization performance of CNNs trained and tested on the same datasets.

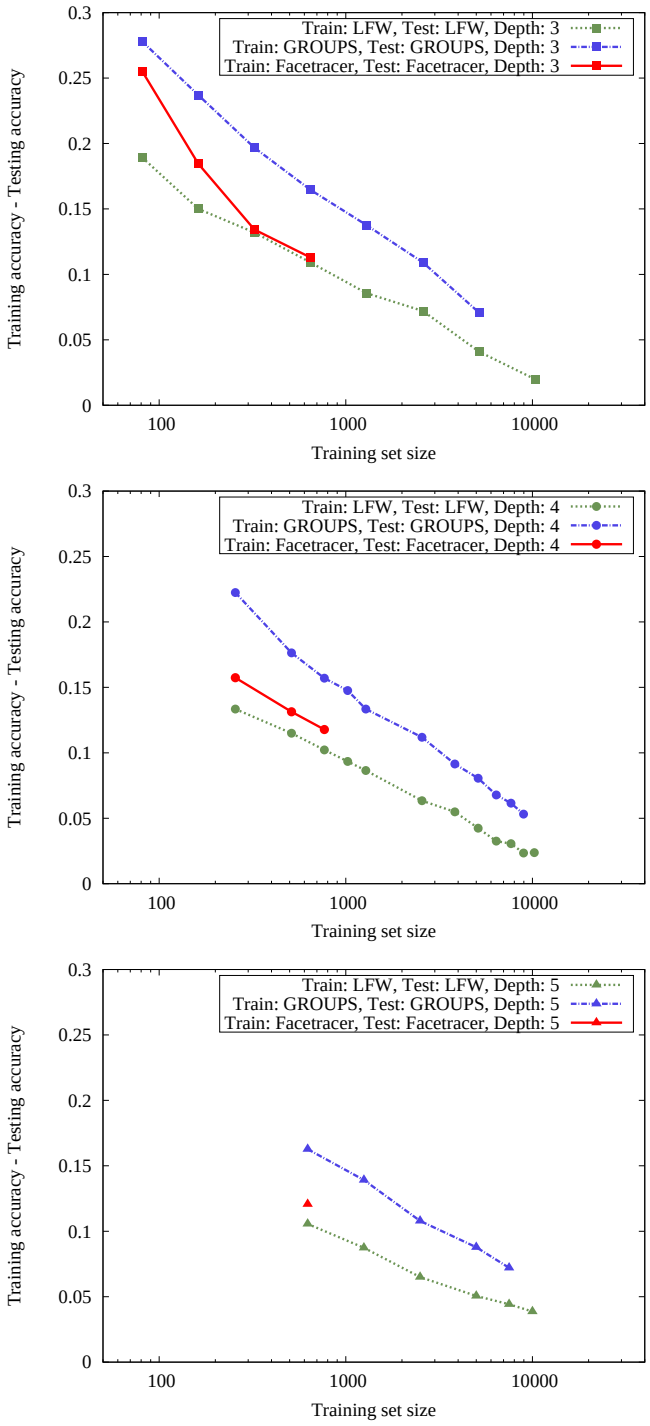

Figure 2.7: Generalization performance of CNNs of depths 3, 4, and 5 trained and tested on the same datasets.

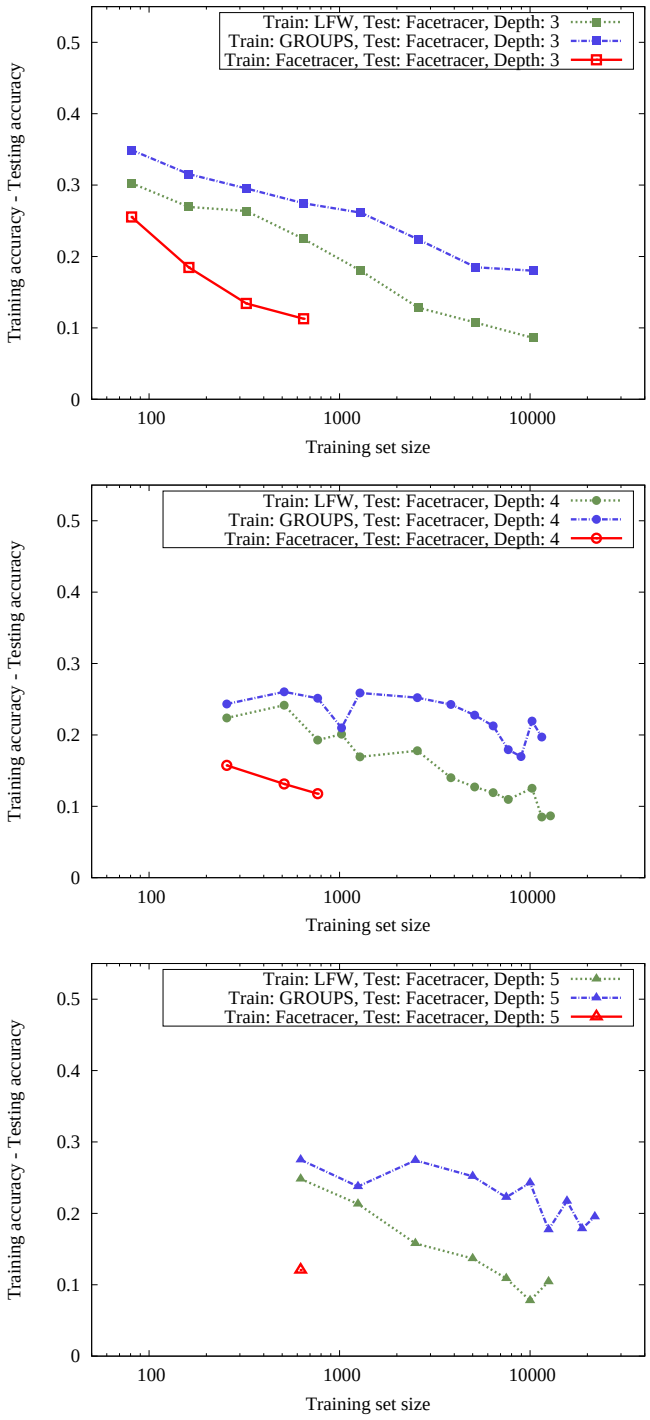

Figure 2.8: Generalization performance of CNNs of depths 3, 4, and 5 tested on Facetracer and trained on different datasets.

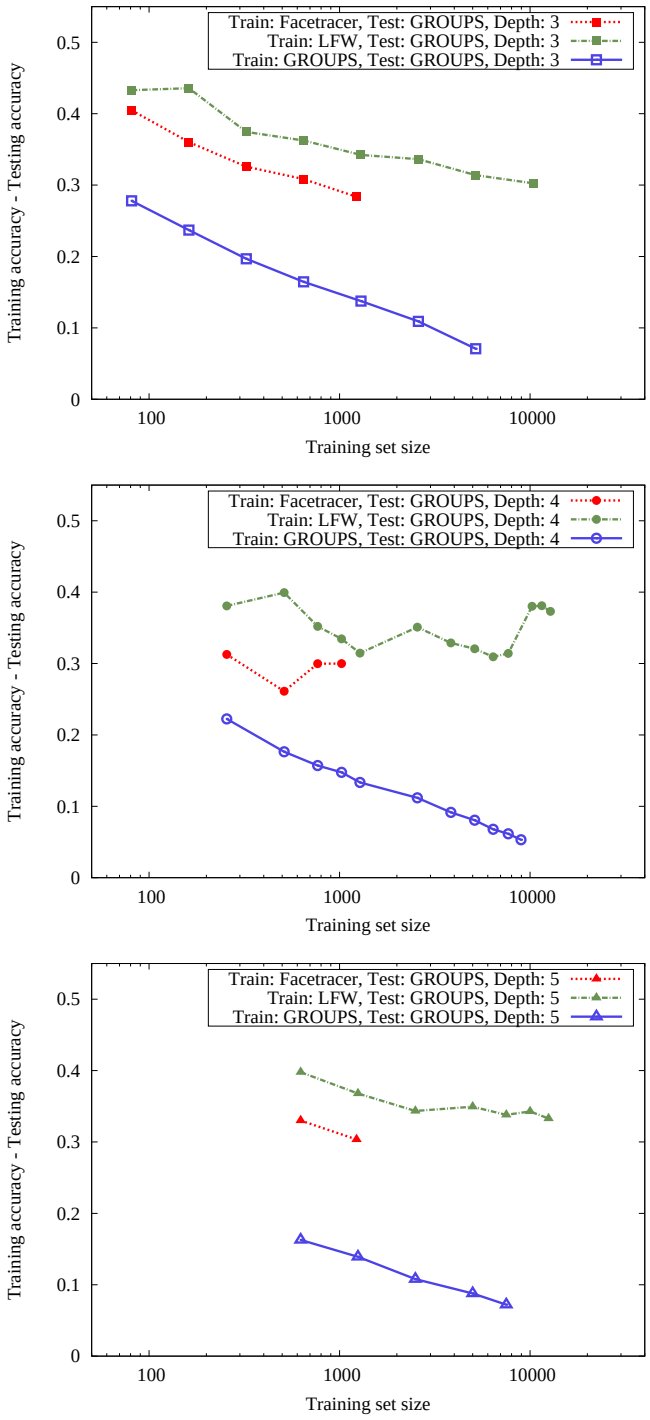

Figure 2.9: Generalization performance of CNNs of depths 3, 4, and 5 tested on GROUPS and trained on different datasets.

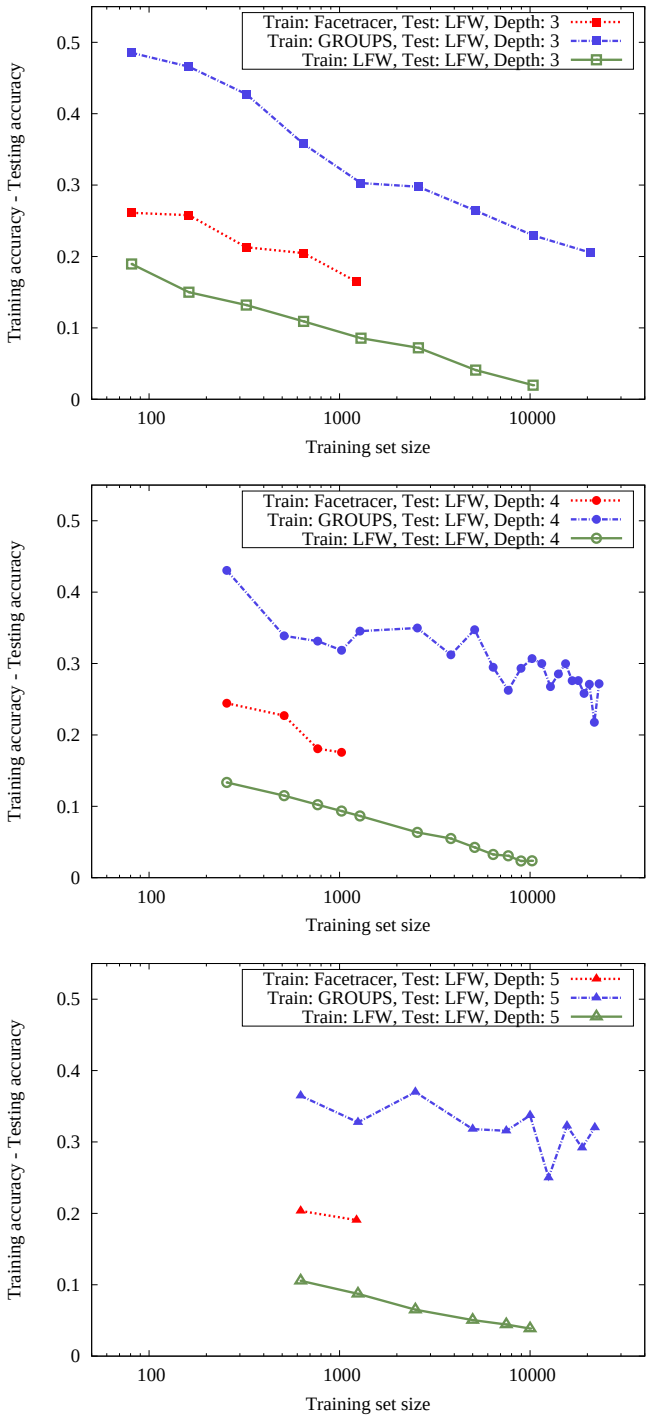

Figure 2.10: Generalization performance of CNNs of depths 3, 4, and 5 tested on LFW and trained on different datasets.

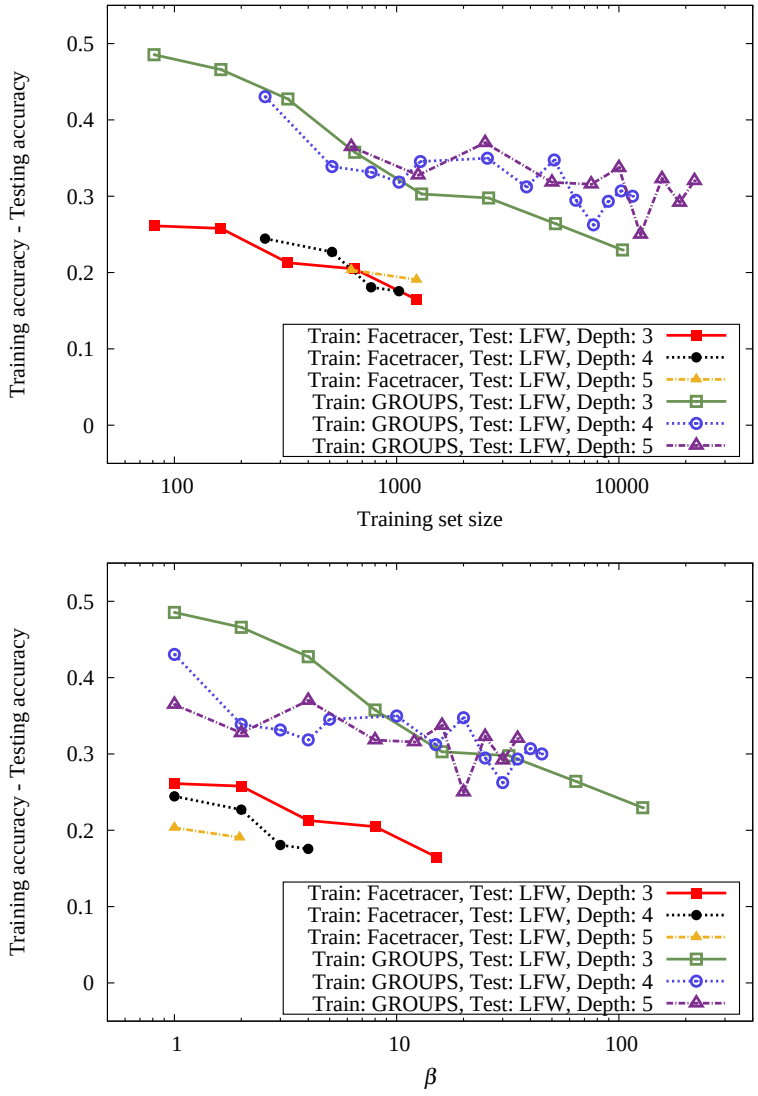

Figure 2.11: Generalization performance of CNNs tested on LFW and trained on different datasets.

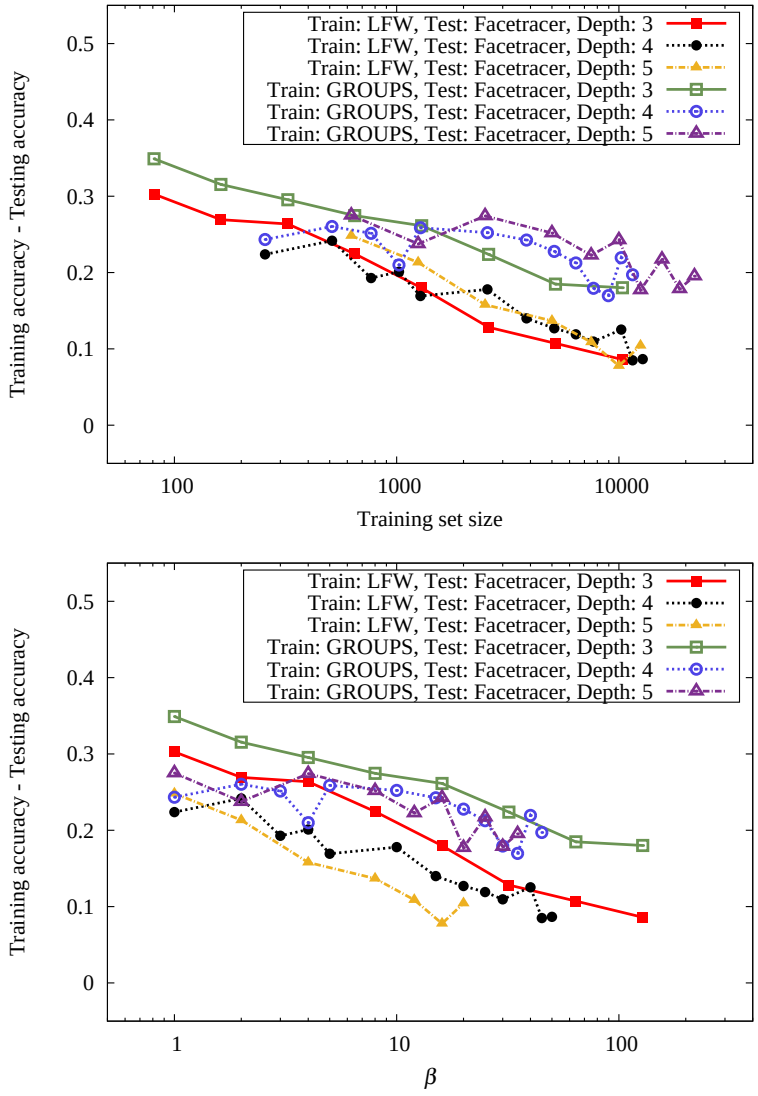

Figure 2.12: Generalization performance of CNNs tested on Facetracer and trained on different datasets.

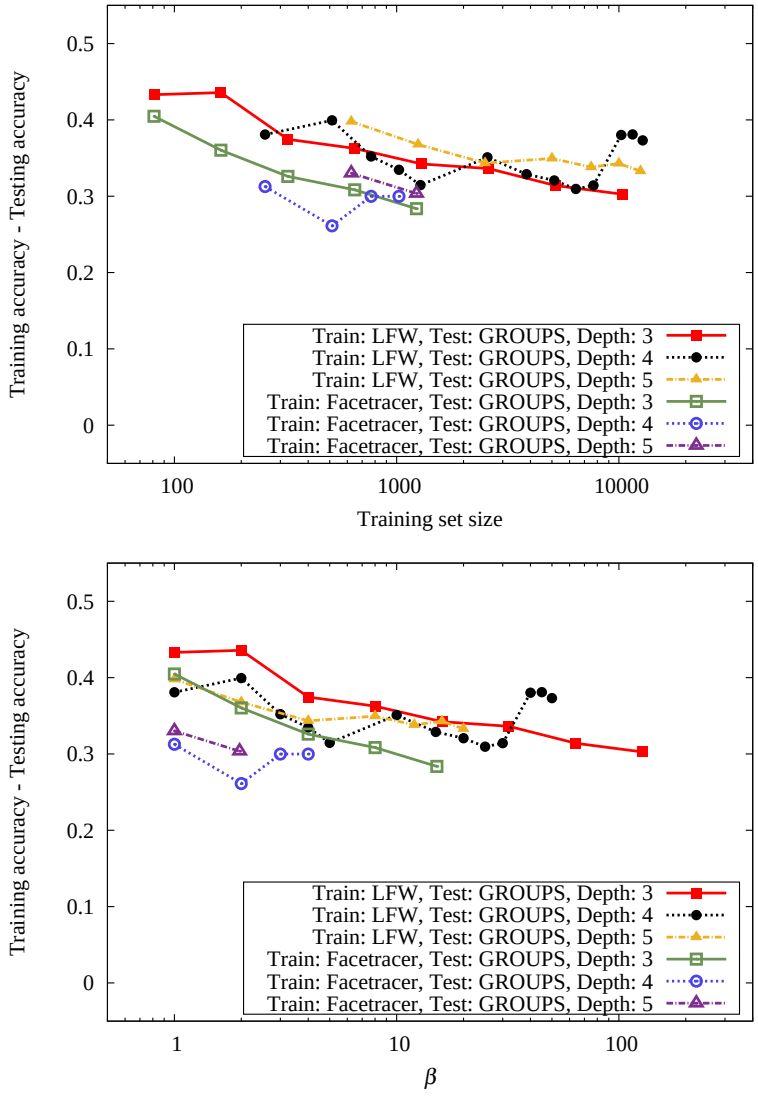

Figure 2.13: Generalization performance of CNNs tested on GROUPS and trained on different datasets.

Chapter 3: The case for spatial pooling in deep convolutional sparse coding

3.1 Overview

In this chapter, we continue to investigate the ideal training and inference conditions for Convolutional Neural Networks (CNNs), this time examining the design of their layers based on a connection with the Convolutional Sparse Coding (CSC) model.

The sparse coding framework has been a very popular choice for a signal model, where one seeks sparse representations for high dimensional signals, on the basis of a global dictionary. Given a vector $\mathbf{X} \in \mathbb{R}^N$, and a *dictionary* $\mathbf{D} \in \mathbb{R}^{N \times M}$, the sparse representation problem can be formulated as finding a sparse vector $\mathbf{\Gamma} \in \mathbb{R}^M$ such that $\mathbf{X} = \mathbf{D}\mathbf{\Gamma}$. In other words, the vector $\mathbf{X}$ can be written as a linear combination of a few columns (or *atoms*) of $\mathbf{D}$. The sparse coding problem attempts to recover the sparsest such representation of $\mathbf{X}$, for a fixed dictionary $\mathbf{D}$ [37–39]:

$$\min_{\mathbf{\Gamma}} \|\mathbf{\Gamma}\|_0 \quad \text{s.t.} \quad \mathbf{D}\mathbf{\Gamma} = \mathbf{X}, \tag{3.1}$$

where $\|\mathbf{\Gamma}\|_0$, the ℓ_0 pseudo-norm, counts the number of non-zero elements in $\mathbf{\Gamma}$.

These sparse representations are often used as *features* for various machine learning tasks [40].

The traditional sparse coding framework has been extensively studied in the literature [39, 41]. Theoretical guarantees to the uniqueness of the solution to (3.1) are given in terms of properties of the dictionary $\mathbf{D}$ [37], such as its *mutual coherence* defined as:

$$\mu(\mathbf{D}) = \max_{i \neq j} \frac{|\mathbf{d}_i^T \mathbf{d}_j|}{\|\mathbf{d}_i\|_2 \cdot \|\mathbf{d}_j\|_2}, \tag{3.2}$$

where $\mathbf{d}_i$ is the i-th column of $\mathbf{D}$. However, even if the solution to (3.1) is unique, finding it remains NP-hard. Approximate solutions are therefore often sought [42–44]. In addition, the model is usually extended to also allow for noise and modeling errors, leading to the following formulation:

$$\min_{\mathbf{\Gamma}} \|\mathbf{\Gamma}\|_0 \quad \text{s. t.} \quad \|\mathbf{D}\mathbf{\Gamma} - \mathbf{X}\| \leq \epsilon. \tag{3.3}$$

In many applications, especially with large-dimensional vectors, using one unstructured dictionary is unfeasible. Attempts to resolve this problem include dividing the input into patches of smaller size, and finding the sparse code for each patch [45]. However, such methods fail to represent the underlying relationship between different patches of the same vector. The CSC model is a recent and promising solution to this problem [46–48].

The CSC model imposes a structure on the global dictionary $\mathbf{D}$, requiring it

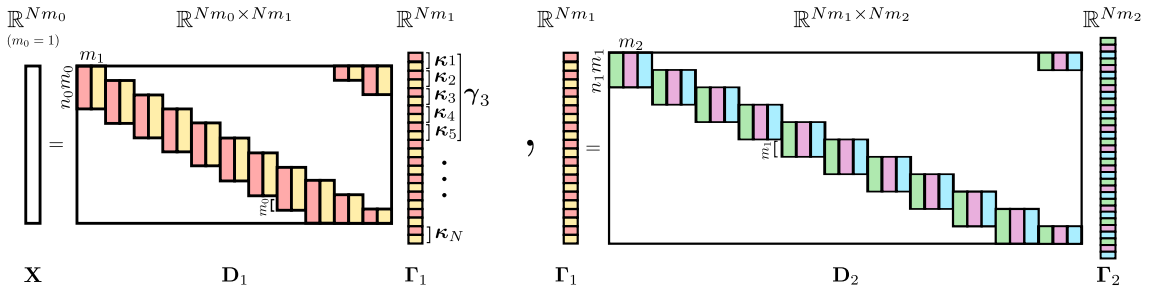

Figure 3.1: Deep convolutional sparse coding. Here, $\mathbf{D}_i$ has a local dictionary with m_i atoms, of length $h_i = n_{i-1}m_{i-1}$ each.

to consist of shifted versions of the same local dictionary of size $h \times m$, as shown in Figure 3.1(left). We refer to such a dictionary $\mathbf{D}$ as a *convolutional* dictionary. According to the CSC model, a vector $\mathbf{X} \in \mathbb{R}^N$ can be written as $\mathbf{X} = \mathbf{D}\boldsymbol{\Gamma}$, where $\mathbf{D} \in \mathbb{R}^{N \times Nm}$ is a convolutional dictionary, and $\boldsymbol{\Gamma} \in \mathbb{R}^{Nm}$ is the resulting sparse code. Here, m is the number of atoms in the *local* dictionary, and h is the length of these atoms. Refer to Figure 3.1(left) for a visual example.

Recently, it was shown in [49, 50] that the CSC model has desirable theoretical guarantees, both in the noiseless and noisy regimes. This was done by introducing a new sparsity measure, the stripe-sparsity, which captures local sparsity properties of a vector. This measure is appropriate under the CSC model which operates locally.

Another seemingly unrelated line of research, which has recently gained popularity, is deep learning and, in particular, CNNs, due to their remarkable performance on complex machine learning tasks [1, 2], often surpassing human performance. We refer the reader to [5] for detailed information on CNNs. Like sparse coding, CNNs aim to find suitable representations for high dimensional inputs, to be used in machine learning problems. However, unlike sparse coding, not much is known about

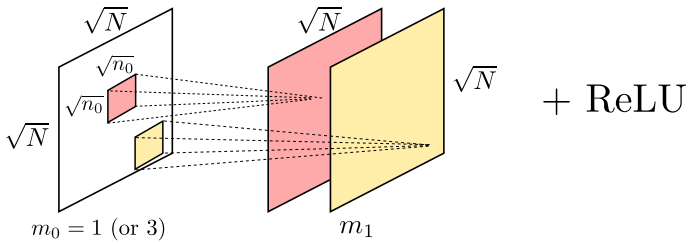

Figure 3.2: One layer of a CNN forward pass. An input image is convolved with a number of filters, followed by a ReLU activation.

their theoretical properties. While some theoretical results on deep architectures exist [24, 51, 52], further work is necessary to obtain a complete and fundamental understanding of why they perform so well. Towards this goal, a connection between CNNs and CSC was recently made in [3]. Specifically, it was shown that the *forward pass* of the CNN, which computes the representation of an input vector, is well approximated by a series of CSC steps. The forward pass of the CNN model considered in [3] is illustrated in Figure 3.2.

In this chapter, we extend the CNN model used in [3] to include the *spatial pooling* operation, originally inspired by models of the visual cortex [53], and widely used in the best-performing CNN architectures [1, 2]. Then, we investigate the theoretical benefits of doing so, and find that, in addition to the known benefits (such as translation invariance [4, 5]), the addition of pooling layers does not affect the CNN performance while decreasing the dimensionality of the involved vectors. The pooling layers also introduce additional benefits such as noise suppression and preventing codes from becoming too sparse.

3.2 Problem formulation

For the sake of completeness, we repeat definitions given in [49, 50], and refer the reader to these two papers for an in-depth discussion. Due to the structure of the convolutional dictionary, the sparse code $\mathbf{\Gamma}$ consists of N independent parts, each of length m. We denote each such part as $\boldsymbol{\kappa}_i$, the *local* sparse vector, as shown in Figure 3.1(left).

Definition 3.1 *A stripe $\boldsymbol{\gamma}_i$ is defined as a group of $2h-1$ adjacent sparse vectors $\boldsymbol{\kappa}_j$ of length m, centered at location $\boldsymbol{\kappa}_i$ [49]. h is the size of atoms in the local dictionary.*

Definition 3.2 *The stripe-sparsity pseudo-norm of a global sparse vector $\mathbf{\Gamma}$ is given by [49]:*

$$\|\mathbf{\Gamma}\|_{0,\infty}^{\mathbf{s}} = \max_i \|\boldsymbol{\gamma}_i\|_0. \tag{3.4}$$

The stripe-sparsity or ($\ell_{0,\infty}$) counts the number of non-zero elements in the densest stripe $\boldsymbol{\gamma}_i$ of $\mathbf{\Gamma}$, and is thus a local measure of sparsity.

We consider a CNN with L consecutive convolutional layers. [3] shows that, under certain conditions, the output of each layer of the CNN with input $\mathbf{X}$ is an approximation to the sparse codes $\{\mathbf{\Gamma}_i\}_{i=1}^{L}$ given by:

$$\begin{aligned} &\text{Find} \quad \{\mathbf{\Gamma}_i\}_{i=1}^{L} \quad \text{s. t.} \\ &\mathbf{X} = \mathbf{D}_1\mathbf{\Gamma}_1, \qquad \|\mathbf{\Gamma}_1\|_{0,\infty}^{\mathbf{s}} \leq \lambda_1 \end{aligned} \tag{3.5}$$

$$\begin{aligned}
&\boldsymbol{\Gamma}_1 = \mathbf{D}_2\boldsymbol{\Gamma}_2, \qquad \|\boldsymbol{\Gamma}_2\|_{0,\infty}^{\mathbf{s}} \leq \lambda_2 \\
&\quad \vdots \\
&\boldsymbol{\Gamma}_{L-1} = \mathbf{D}_L\boldsymbol{\Gamma}_L, \quad \|\boldsymbol{\Gamma}_L\|_{0,\infty}^{\mathbf{s}} \leq \lambda_L,
\end{aligned}$$

where $\{\lambda_i\}_{i=1}^L$ are scalar parameters. The atoms of the local dictionary of $\mathbf{D}_i$ are the filters of the i-th CNN layer.

The problem in (3.5) is a layered CSC problem, which essentially computes a sparse representation $\boldsymbol{\Gamma}_1$ of the input $\mathbf{X}$ based on a convolutional dictionary $\mathbf{D}_1$, and then in turn computes the sparse representation of $\boldsymbol{\Gamma}_1$ based on another convolutional dictionary $\mathbf{D}_2$, and so on. This process is illustrated in Figure 3.1, with the dimensions of vectors and dictionaries as denoted in the figure. Hereafter, we refer to the problem in (3.5) as *deep (convolutional) sparse coding* with parameters $\boldsymbol{\lambda}$, denoted by DSC($\boldsymbol{\lambda}$).

We now define the pooling operation. We will consider two types of pooling, namely average- and max-pooling. Let $\mathbf{X} = \mathbf{D}\boldsymbol{\Gamma}$, for a vector $\mathbf{X} \in \mathbb{R}^N$ and convolutional dictionary $\mathbf{D}$ with m atoms in its local dictionary. As previously mentioned, $\boldsymbol{\Gamma} \in \mathbb{R}^{Nm}$ can be written as $\boldsymbol{\Gamma} = [\boldsymbol{\kappa}_1^T, \boldsymbol{\kappa}_2^T, \ldots, \boldsymbol{\kappa}_N^T]^T$.

Definition 3.3 *The max-pooling operation, $\mathcal{M}_{b,s}(\cdot)$, and the average-pooling operation, $\mathcal{A}_{b,s}(\cdot)$, with pooling block size $b \in \mathbb{Z}^+$ and stride $s \in \mathbb{Z}^+$, are defined as:*

$$\left(\mathcal{M}_{b,s}(\boldsymbol{\Gamma})\right)_k \triangleq \max_{Qs+1 \leq i \leq Qs+b} \left(\boldsymbol{\kappa}_i\right)_R, \tag{3.6}$$

$$\left(\mathcal{A}_{b,s}(\mathbf{\Gamma})\right)_k \triangleq \frac{1}{b} \sum_{Qs+1 \le i \le Qs+b} \left(\boldsymbol{\kappa}_i\right)_R, \tag{3.7}$$

where $i, k \in \mathbb{Z}^+$,

$$Q = \left\lfloor \frac{k-1}{m} \right\rfloor, \quad R = (k-1) \ (mod\ m) + 1, \tag{3.8}$$

$\left(\boldsymbol{\kappa}_i\right)_R$ *is the* R*-th element of* $\boldsymbol{\kappa}_i$*, and* $1 \le k \le m\left(\frac{N-b}{s}+1\right)$.[1]

This process combines nearby features using a commutative operation. It is described in Figure 3.3, for $m = 3$, $N = 8$, $b = 4$, and $s = 2$. Pooling, as used in CNNs, is illustrated in Figure 3.4. These two operations are equivalent.

We propose to add a pooling operation after every CSC step in (3.5). Specifically, we will analyze properties of the following problem:

$$\begin{aligned}
&\text{Find} \quad \{\mathbf{\Gamma}_i, \mathbf{P}_i\}_{i=1}^{L} \quad \text{s. t.} \\
&\mathbf{X} = \mathbf{D}_1\mathbf{\Gamma}_1, \qquad \mathbf{P}_1 = \underset{b_1,s_1}{\text{Pool}}(\mathbf{\Gamma}_1), \quad \|\mathbf{\Gamma}_1\|_{0,\infty}^{\mathbf{s}} \le \lambda_1 \\
&\mathbf{P}_1 = \mathbf{D}_2\mathbf{\Gamma}_2, \qquad \mathbf{P}_2 = \underset{b_2,s_2}{\text{Pool}}(\mathbf{\Gamma}_2), \quad \|\mathbf{\Gamma}_2\|_{0,\infty}^{\mathbf{s}} \le \lambda_2 \\
&\quad \vdots \\
&\mathbf{P}_{L-1} = \mathbf{D}_L\mathbf{\Gamma}_L, \qquad \mathbf{P}_L = \underset{b_L,s_L}{\text{Pool}}(\mathbf{\Gamma}_L), \quad \|\mathbf{\Gamma}_L\|_{0,\infty}^{\mathbf{s}} \le \lambda_L,
\end{aligned} \tag{3.9}$$

where $\text{Pool}_{b,s}(\cdot)$ is either $\mathcal{M}_{b,s}(\cdot)$ or $\mathcal{A}_{b,s}(\cdot)$ as defined in (3.6) and (3.7), respectively. Compared with the problem in (3.5), every step of our proposed method will find

[1] We assume $\frac{N-b}{s}$ is an integer. If not, we either ignore the extra elements or "pad" $\mathbf{\Gamma}$ by appropriate values.

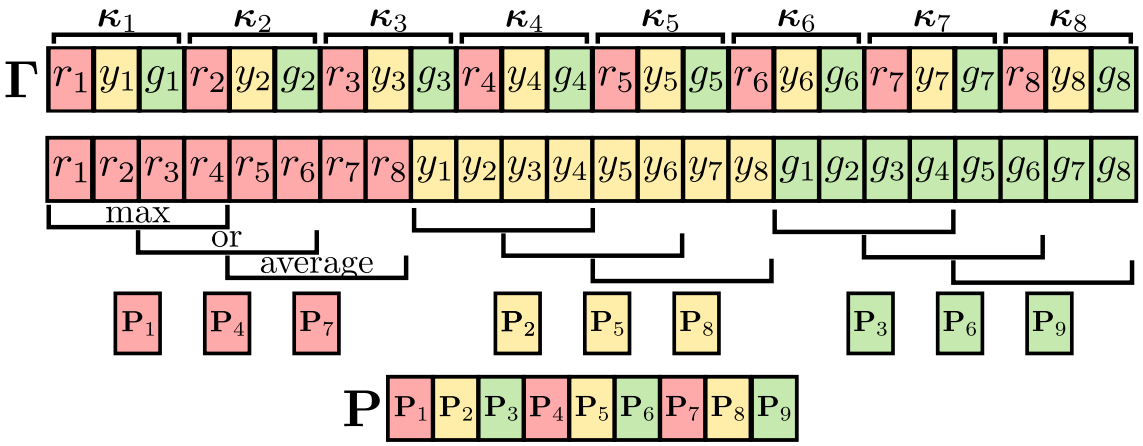

Figure 3.3: Spatial pooling in CSC.

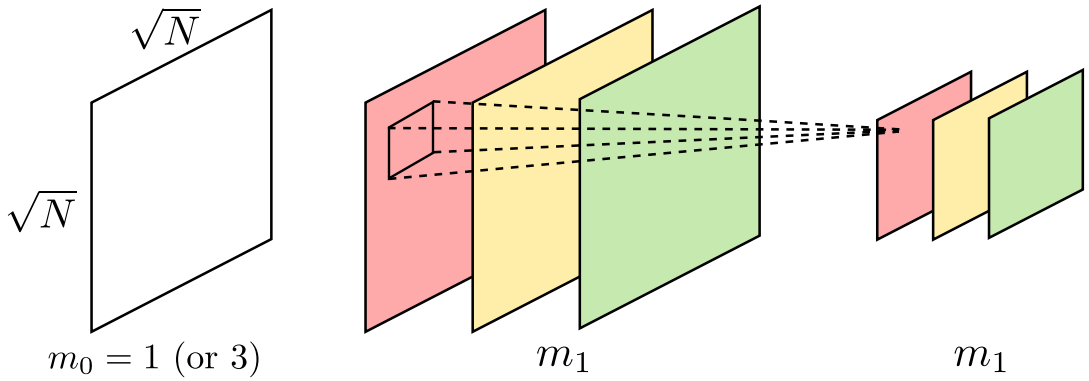

Figure 3.4: Spatial pooling in CNNs.

sparse representations of the *pooled* version of the previous sparse representation. Note that the size of the dictionaries is also smaller than in (3.5), to match the size of the pooled vectors. We refer to the problem in (3.9) as *deep (convolutional) sparse coding with pooling* with parameters $\boldsymbol{\lambda}$, $\mathbf{b}$, and $\mathbf{s}$, denoted by DSCP$(\boldsymbol{\lambda}, \mathbf{b}, \mathbf{s})$. We do not enforce pooling after every layer, as the choice $b_i = 1, s_i = 1$ ensures there is no pooling at the i-th step.

3.3 Results

3.3.1 Uniqueness and stability of DSCP

We start by investigating the uniqueness and stability properties of DSCP($\boldsymbol{\lambda}, \mathbf{b}, \mathbf{s}$). Given a vector $\mathbf{X}$ and convolutional dictionaries $\{\mathbf{D}_i\}_{i=1}^L$, it is desirable that there be a unique solution $\{\boldsymbol{\Gamma}_i^*, \mathbf{P}_i^*\}_{i=1}^L$ to (3.9). This is especially important since the sparse codes are often used as *features* for classification or regression tasks. If the solution to (3.9) is not unique, the same signal could be represented by more than one different features, therefore resulting in a different classification or regression output depending on which solution is adopted.

Theorem 3.1 *Assume that, for some vector* $\mathbf{X}$*, the DSCP*($\boldsymbol{\lambda}, \mathbf{b}, \mathbf{s}$) *model in (3.9) admits a solution* $\{\boldsymbol{\Gamma}_i^*, \mathbf{P}_i^*\}_{i=1}^L$*. If, for all* $1 \leq i \leq L$*,*

$$\|\boldsymbol{\Gamma}_i^*\|_{0,\infty}^{\mathbf{s}} \leq \lambda_i < \frac{1}{2}\left(1 + \frac{1}{\mu(\mathbf{D}_i)}\right), \tag{3.10}$$

then, the solution $\{\boldsymbol{\Gamma}_i^*, \mathbf{P}_i^*\}_{i=1}^L$ *is unique.*

Besides uniqueness, another desirable property is the stability of the solution when the input is contaminated with bounded noise. Let $\mathbf{Y} = \mathbf{X} + \mathbf{E}$, where $\mathbf{X}$ satisfies the DSCP($\boldsymbol{\lambda}, \mathbf{b}, \mathbf{s}$) problem, and $\mathbf{E}$ is additive noise. Consider the following problem:

$$
\begin{aligned}
&\text{Find} \quad \{\mathbf{\Gamma}_i, \mathbf{P}_i\}_{i=1}^{L} \qquad \text{s. t.} \\
&\|\mathbf{Y} - \mathbf{D}_1\mathbf{\Gamma}_1\|_2 \leq \epsilon_1, \qquad \mathbf{P}_1 = \underset{b_1,s_1}{\text{Pool}}(\mathbf{\Gamma}_1), \quad \|\mathbf{\Gamma}_1\|_{0,\infty}^{\mathbf{s}} \leq \lambda_1 \\
&\|\mathbf{P}_1 - \mathbf{D}_2\mathbf{\Gamma}_2\|_2 \leq \epsilon_2, \qquad \mathbf{P}_2 = \underset{b_2,s_2}{\text{Pool}}(\mathbf{\Gamma}_2), \quad \|\mathbf{\Gamma}_2\|_{0,\infty}^{\mathbf{s}} \leq \lambda_2 \\
&\quad \vdots \\
&\|\mathbf{P}_{L-1} - \mathbf{D}_L\mathbf{\Gamma}_L\|_2 \leq \epsilon_L, \ \mathbf{P}_L = \underset{b_L,s_L}{\text{Pool}}(\mathbf{\Gamma}_L), \ \|\mathbf{\Gamma}_L\|_{0,\infty}^{\mathbf{s}} \leq \lambda_L.
\end{aligned} \tag{3.11}
$$

We denote this problem by DSCP$^{\boldsymbol{\epsilon}}(\boldsymbol{\lambda}, \mathbf{b}, \mathbf{s})$ and show that its solution is not *too different* from that of DSCP$(\boldsymbol{\lambda}, \mathbf{b}, \mathbf{s})$.

Theorem 3.2 *Suppose a vector* $\mathbf{X}$ *satisfies the DSCP*$(\boldsymbol{\lambda}, \mathbf{b}, \mathbf{s})$ *model in (3.9), but is contaminated with noise* $\mathbf{E}$*, where* $\|\mathbf{E}\|_2 \leq \epsilon$*, resulting in* $\mathbf{Y} = \mathbf{X} + \mathbf{E}$*. Suppose* $\{\mathbf{\Gamma}_i^*, \mathbf{P}_i^*\}_{i=1}^{L}$ *solves the problem in (3.9) and* $\{\hat{\mathbf{\Gamma}}_i, \hat{\mathbf{P}}_i\}_{i=1}^{L}$ *solves the problem in (3.11). If*

$$
\|\mathbf{\Gamma}_i^*\|_{0,\infty}^{\mathbf{s}} \leq \lambda_i < \frac{1}{2}\left(1 + \frac{1}{\mu(\mathbf{D}_i)}\right), \ \ \forall\, 1 \leq i \leq L, \tag{3.12}
$$

$$
\epsilon_1 = \epsilon, \quad \epsilon_i^2 = \frac{4\epsilon_{i-1}^2}{1 - (2\|\mathbf{\Gamma}_i^*\|_{0,\infty}^{\mathbf{s}} - 1)\mu(\mathbf{D}_i)} \ \ \forall\, i \geq 2, \tag{3.13}
$$

then, for all $1 \leq i \leq L$*,*

$$
\|\mathbf{P}_i^* - \hat{\mathbf{P}}_i\|_2^2 \leq \|\mathbf{\Gamma}_i^* - \hat{\mathbf{\Gamma}}_i\|_2^2 \leq \epsilon_{i+1}^2. \tag{3.14}
$$

If max-pooling is used, it is further required that $s_i \geq b_i, \ \forall\, 1 \leq i \leq L$*.*

In the proof of Theorem 3.2, we will make use of the following lemmas.

Lemma 3.1 *Let* $\mathbf{X}$ *and* $\hat{\mathbf{X}}$ *be two vectors in* $\mathbb{R}^N$*, and* $\mathbf{P} = \mathcal{M}_{b,s}(\mathbf{X})$*,* $\hat{\mathbf{P}} = \mathcal{M}_{b,s}(\hat{\mathbf{X}})$ *their max-pooled versions, with block size* b *and stride* $s \geq b$*. Then,* $\|\mathbf{P} - \hat{\mathbf{P}}\|_2 \leq \|\mathbf{X} - \hat{\mathbf{X}}\|_2$*.*

Lemma 3.2 *Let* $\mathbf{X}$ *and* $\hat{\mathbf{X}}$ *be two vectors in* $\mathbb{R}^N$*, and* $\mathbf{P} = \mathcal{A}_{b,s}(\mathbf{X})$*,* $\hat{\mathbf{P}} = \mathcal{A}_{b,s}(\hat{\mathbf{X}})$ *their average-pooled versions, with pooling block size* b *and stride* s*. Then,* $\|\mathbf{P} - \hat{\mathbf{P}}\|_2 \leq \|\mathbf{X} - \hat{\mathbf{X}}\|_2$*.*

Theorem 3.2 shows that the DSCP model is stable under bounded noise. This is, again, important for classification tasks where it is desirable that the representation does not change much when the input is affected by minor noise. Adding the pooling layers does not affect the stability of the codes. In fact, in most cases, the pooling layers will help suppress noise propagating to the next layer. This is because the inequalities in Lemmas 3.1 and 3.2 are strict except for specific edge cases.[2]

3.3.2 Stability of the CNN forward pass with pooling

As previously mentioned, a connection between the CNN forward pass (without pooling) and the DSC problem was made in [3]. We now consider a CNN with (optional) pooling after every convolutional layer. The i-th layer of the CNN with input $\hat{\mathbf{P}}_{i-1}$ computes the following output: $\hat{\mathbf{\Gamma}}_i = \mathcal{S}_{\beta_i}(\mathbf{D}_i^T \hat{\mathbf{P}}_{i-1})$, followed by $\hat{\mathbf{P}}_i = \text{Pool}_{b_i,s_i}(\hat{\mathbf{\Gamma}}_i)$, where $\mathcal{S}_{\beta_i}(\cdot)$ is a thresholding operation with parameter β_i, applied element-wise and defined in (3.15). The multiplication of the input by $\mathbf{D}_i^T$ is equivalent to how the input is convolved with various filters in CNNs (the filters are

[2]For instance, in the case of max-pooling, $\|\mathbf{P} - \hat{\mathbf{P}}\|_2 = \|\mathbf{X} - \hat{\mathbf{X}}\|_2$ if $\mathbf{X}$ and $\hat{\mathbf{X}}$ only differ in the maximum element of each pooling block.

the atoms of the local dictionary). $\mathcal{S}_{\beta_i}(\cdot)$ is chosen to resemble a two-sided ReLU activation function [54]. With abuse of notation, for a real-valued z, we define

$$\mathcal{S}_{\beta_i}(z) = \begin{cases} z + \beta_i, & z < -\beta_i \\ 0, & -\beta_i \leq z \leq \beta_i \\ z - \beta_i, & \beta_i < z. \end{cases} \tag{3.15}$$

By extending [3, Theorem 10], we are able to show that, under some conditions on $\|\mathbf{\Gamma}_i^*\|_{0,\infty}^{\mathbf{s}}$ and β_i, the solution to (3.9), $\mathbf{\Gamma}_i^*$, and $\hat{\mathbf{\Gamma}}_i$ have the same support and the difference between them is bounded. We omit the proof and technical details for the sake of brevity. It is seen that the addition of the spatial pooling operations does not change the stability properties of the CNN.

3.3.3 Sparsity bounds

In the DSC model, the objective is to find sparse representations of already sparse vectors. It is natural to wonder whether performing this task L times successively is feasible, i.e., whether the sparse codes will keep admitting sparse representations [55]. In the following theorem, we show that, under some condition on the dictionaries, the sparse codes will get sparser and sparser with every layer. We first define the *normalized sparsity* of a vector (and a matrix), which will allow us to compare the sparsity of vectors with different lengths.

Definition 3.4 *For a vector* $\mathbf{X}$*, let the normalized sparsity be the ratio of the* ℓ_0

pseudo-norm of $\mathbf{X}$ *to its length, i.e.,*

$$\mathcal{F}(\mathbf{X}) \triangleq \frac{\|\mathbf{X}\|_0}{\dim(\mathbf{X})}. \tag{3.16}$$

Definition 3.5 *For a matrix* $\mathbf{M}$*, let the normalized sparsity be the ratio of the maximum number of non-zero elements in a column of* $\mathbf{M}$ *to the number of rows in* $\mathbf{M}$*. This is equivalent to the largest normalized sparsity of any of its columns, i.e.,*

$$\mathcal{F}(\mathbf{M}) \triangleq \max_{\mathbf{X} \in \text{Col}(\mathbf{M})} \mathcal{F}(\mathbf{X}). \tag{3.17}$$

Theorem 3.3 *Suppose* $\mathbf{X}$ *satisfies the DSC*($\boldsymbol{\lambda}$) *model in (3.9) and the conditions of [3, Theorem 10] are met. If, for all* $1 \leq i \leq L$*, the convolutional dictionaries satisfy* $\mathcal{F}(\mathbf{D}_i^T) \leq \frac{1}{Nm_{i-1}}$*, then* $\mathcal{F}(\boldsymbol{\Gamma}_i) \leq \mathcal{F}(\boldsymbol{\Gamma}_{i-1})$*.*

Theorem 3.3 implies that the normalized sparsity of the sparse codes decreases with the number of layers. While the normalized sparsity assumption on the dictionaries might seem too strict, we note that the rows of the dictionaries are very sparse due to the convolutional structure and the fact that the local dictionary is shifted by m_i rows at a time, as seen in Figure 3.1. In addition, the inequalities involved in the proof being typically loose, the theorem can still be satisfied under much looser constraints. The implication is that not too many layers can be used in the DSC model, before the codes become too sparse (or all zero) for any practical use. Adding spatial pooling operations "unsparsifies" the sparse codes, thus only retaining important information and allowing a very deep representation.

3.4 Concluding remarks

In this chapter, we investigated the effect of adding spatial pooling operations to deep CSC problems. We showed that this addition does not interfere with the uniqueness and stability properties of the original problem. Furthermore, we showed that, while allowing more compact representations, pooling may in fact introduce benefits to the overall system in terms of noise suppression and, most importantly, preventing the codes from becoming too sparse and vanishing.

Chapter 4: Quality over quantity: Active selection strategies for improved performance of CNNs

4.1 Overview

In this chapter, we develop a novel sampling technique to counteract the exponential decrease in the incremental benefit of a new additional training sample. We also relax some of the ideal training scenarios previously mentioned, and show that our technique helps mitigate the effect of lack of enough training data, class imbalance in the training set, and noisy training labels.

Currently, the best performing deep networks have many hidden layers and an extremely large number of trainable parameters, therefore requiring vast amounts of training data [1, 56, 57]. This raises the question of whether all this data is really necessary and, perhaps most importantly, whether all training samples are equally valuable in the learning process. In fact, [58] suggests that guiding a classifier by presenting training samples in an order of increasing difficulty can speed up learning and result in convergence to a better local minimum. Furthermore, modern large-scale training sets often include redundant or noisy samples which could cause learning bias. In this chapter, we address the problem of adaptively selecting training

samples to reduce the effect of learning bias and improve generalization performance.

Related work. This problem, sometimes referred to as *exemplar* or *active selection*, has been studied in the literature. Starting with a given set of labeled examples, active selection aims to identify a subset to use for training, while leveraging information obtained from the classifier trained on previous selections. One simple approach [59] repeatedly presents the same example if the network error exceeds a threshold. In [60], this problem is addressed in the context of feedforward neural networks. The authors propose a sequential method to select one training sample at a time such that, when added to the previous set of examples, it results in the largest decrease in a squared error estimate criterion. A similar objective is considered in [61] based on pattern informativeness – a measure of a sample's influence on the classifier output. However, most active selection techniques in the literature introduce one training sample at a time (as opposed to a batch) and use estimation techniques which are not applicable to high-dimensional inputs. This makes them not suitable for modern deep learning tasks, and especially computer vision applications.

A closely related approach is *active learning* which starts with an unlabeled set of examples and sequentially identifies critical samples to label and train on [62–67]. It is shown that a classifier trained on a carefully chosen subset can sometimes outperform one that is trained on all the available data. In contrast with active learning, active selection assumes a fully supervised setting where all training samples are labeled and available a priori.

Contributions. In this chapter, we present strategies to make optimal use of available training data by adaptively selecting batches of training samples which will be iteratively presented to the classifier. We are interested in incrementally training a deep neural network, using batches of training data carefully selected to meet four criteria: class balance, diversity, representativeness, and classifier uncertainty. The class balance criterion utilizes the a priori knowledge of labels to ensure that all classes are appropriately present in the new training batch. We propose a novel class balancing algorithm which uses immediate feedback from the classifier to allot a subset of training samples to each class based on the average classifier performance on that class. Diversity and representativeness are distance-based measures aiming to reduce redundancy while maximizing the quality of selected samples. Such measures have been used in active learning [68, 69], subset selection [70, 71], and clustering [72]. Finally, the classifier uncertainty criterion favors samples that the classifier has not yet properly learnt, thus driving it to explore unvisited parts of the input space. We combine the last three criteria and use optimization techniques from [73, 74] to identify a near-optimal batch to train on at every iteration. We apply our methods on various classification problems. Our results indicate that careful selection and ordering of training samples can lead to improved performance, compared to sampling training data at random.

The rest of the chapter is organized as follows. The problem formulation is stated in Sections 4.2.1 - 4.2.5 and the proposed solution and algorithm in Sections 4.2.6 and 4.2.7. Experimental results, comparing our method to random sampling, are presented in Section 4.3. In Section 4.4, we discuss the computational complexity

of our method as well as potential improvements.

4.2 Problem statement

We assume we are given a fixed classifier architecture, and a set of labeled training data points: $\mathcal{X} = \bigcup_{k=1}^{L} \mathcal{X}_k$, where $\mathcal{X}_k = \{X_{1,k}, X_{2,k}, \ldots, X_{N_k,k}\}$ are the training samples belonging to class k, $N_k = |\mathcal{X}_k|$ is the number of training samples from class k, and L is the number of classes. At each time t, we select a subset $\mathcal{B}^t \subset \mathcal{X}$, such that the classifier (which has previously been trained on $\mathcal{B}^{t-1}$) exhibits good generalization performance when trained on $\mathcal{B}^t$.

To this end, we formulate a criterion for selecting new training examples which serves the following objectives:

(O1) The samples in $\mathcal{B}^t$ must be such that the classifier is *uncertain* about classifying them (or *certain* but *wrong* in its classification).

(O2) $\mathcal{B}^t$ should have a *balanced* selection from all classes.

(O3) $\mathcal{B}^t$ should be sufficiently *diverse*.

(O4) $\mathcal{B}^t$ should be *representative* of $\mathcal{X}$.

We will mathematically formulate each of these objectives in the following sections.

4.2.1 Classifier uncertainty and error

We assume that, at time t, the classifier produces L outputs for each training sample $X_{i,k}$ from class k, denoted by

$$p^t(X_{i,k}) = [\quad p_1^t(X_{i,k}),\quad p_2^t(X_{i,k}),\quad \ldots,\quad p_L^t(X_{i,k})\quad], \tag{4.1}$$

where $p_l^t(X)$ is interpreted as the classifier's estimate of the probability that $X \in \mathcal{X}$ belongs to class l, and satisfies $p_l^t(X) \geq 0\ \forall l$, and $\sum_{l=1}^{L} p_l^t(X) = 1$. In order to quantify classifier uncertainty and error we define:

$$c^t(X_{i,k}) = -\sum_{l=1}^{L} \log p_l^t(X_{i,k}) \cdot \left[\beta^t \cdot \mathbb{1}\left[l = k\right](1 - \beta^t) \cdot p_l^t(X_{i,k})\right], \tag{4.2}$$

where $\beta^t \in [0, 1]$ is a chosen parameter. We can interpret $c^t(X_{i,k})$ in two ways. First, $c^t(X_{i,k})$ can be seen as a weighted sum of an error term: $-\sum_{l=1}^{L} \mathbb{1}\left[l = k\right] \cdot \log p_l^t(X_{i,k}) = -\log p_k^t(X_{i,k})$ and an entropy term: $-\sum_{l=1}^{L} p_l^t(X_{i,k}) \log p_l^t(X_{i,k})$. These two terms correspond to the correctness of the classifier's decision and the uncertainty in this decision, therefore satisfying objective (O1) above. Second, $c^t(X_{i,k})$ can be interpreted as a bootstrapping technique to overcome possible label noise [75], in which case $\beta^t \mathbb{1}\left[l = k\right] + (1 - \beta^t) p_l^t(X_{i,k})$ is a weighted "correct label" and $c^t(X_{i,k})$ represents the cross-entropy between $p^t(X_{i,k})$ and this weighted label. $c^t(\cdot)$ being low on a given sample means that the classifier has enough information about this sample. In order to present the classifier with informative samples, we would therefore like

to pick samples where $c^t(\cdot)$ is large.

4.2.2 Class balance

At each time t, we would like to select a total of M^t samples, distributed among all classes in a balanced way. However, it might be counter-intuitive to simply impose that all classes be equally represented in the subset $\mathcal{B}^t$, as the current classifier may be performing very well on some of them. Therefore, we assign a budget M_k^t to each class depending on the average performance on this class. This can be measured by $c_k^t = \frac{1}{N_k} \sum_{i=1}^{N_k} c^t(X_{i,k})$, where $c^t(X_{i,k})$ is defined in (4.2). The larger c_k^t is, more samples are assigned to class k. An objective function of $\sum_{k=1}^{L} c_k^t \cdot M_k^t$ would result in the trivial solution of assigning all the budget M^t to the class with the largest uncertainty score, and would contradict the class balancing requirement. We therefore use a logarithmic objective function and formulate the problem as follows:

$$\begin{aligned} \max_{M_k^t \in \mathbb{Z}^+} \quad & \sum_{k=1}^{L} \log\left(1 + \alpha \cdot c_k^t \frac{M_k^t}{M^t}\right) \\ \text{s. t.} \quad & \sum_{k=1}^{L} M_k^t \leq M^t; \;\; M_k^t \leq |\mathcal{X}_k|, \end{aligned} \tag{4.3}$$

where $\alpha > 1$ sets the sensitivity of the method (the smaller α, the larger the effect of differences in c_k^t) . This problem arises in information theory, in allocating power to a set of communication channels [76, Section 9.4]. We use a similar formulation since M_k^t represents the budget allocated to the k^{th} class (channel), and $1/c_k^t$ is akin to channel quality. There exists a very efficient solution to this convex optimization

problem, known as the water-filling algorithm [77, Section 5.5], where we interpret water levels as the number of samples allocated to each class. Our formulation differs from the standard formulation due to the addition of the last constraint (which ensures that we do not allocate more samples than available in the pool $\mathcal{X}_k$). Another difference is that the feasible set in (4.3) is the set of non-negative integers.

Theorem 4.1 *The modified water-filling problem in (4.3) can be solved using Algorithm 4.1.*

Algorithm 4.1 Integer water-filling algorithm with caps

1: Sort the base levels $\frac{M^t}{\alpha c_k^t}$ in ascending order and take the ceiling of the base levels $\lceil \frac{M^t}{\alpha c_k^t} \rceil$.
2: Place "caps" at $\lceil \frac{M^t}{\alpha c_k^t} \rceil + |\mathcal{X}_k|$.
3: **repeat**
4: Fill with one water unit at a time proceeding from left to right without exceeding any cap.
5: **until** M^t water units are used or all empty spaces are filled.

"Caps" enforce the $M_k^t \leq |\mathcal{X}_k|$ constraints. Each water unit corresponds to one training sample being assigned to a class. An illustration of the algorithm is found in Figure 4.1 for a budget $M^t = 10$. The numbers on the water units show the order in which they have been assigned. Because of the balanced selection of budgets $\{M_k^t\}$, this formulation addresses the class balance objective (O2).

Remark 4.1 *Objectives (O3) and (O4) are only meaningful when considered as intraclass rather than globally. Two images from different classes trivially meet the diversity criterion but cannot be representative of each other. Since we are considering supervised learning settings, we can leverage the label information and focus on finding*

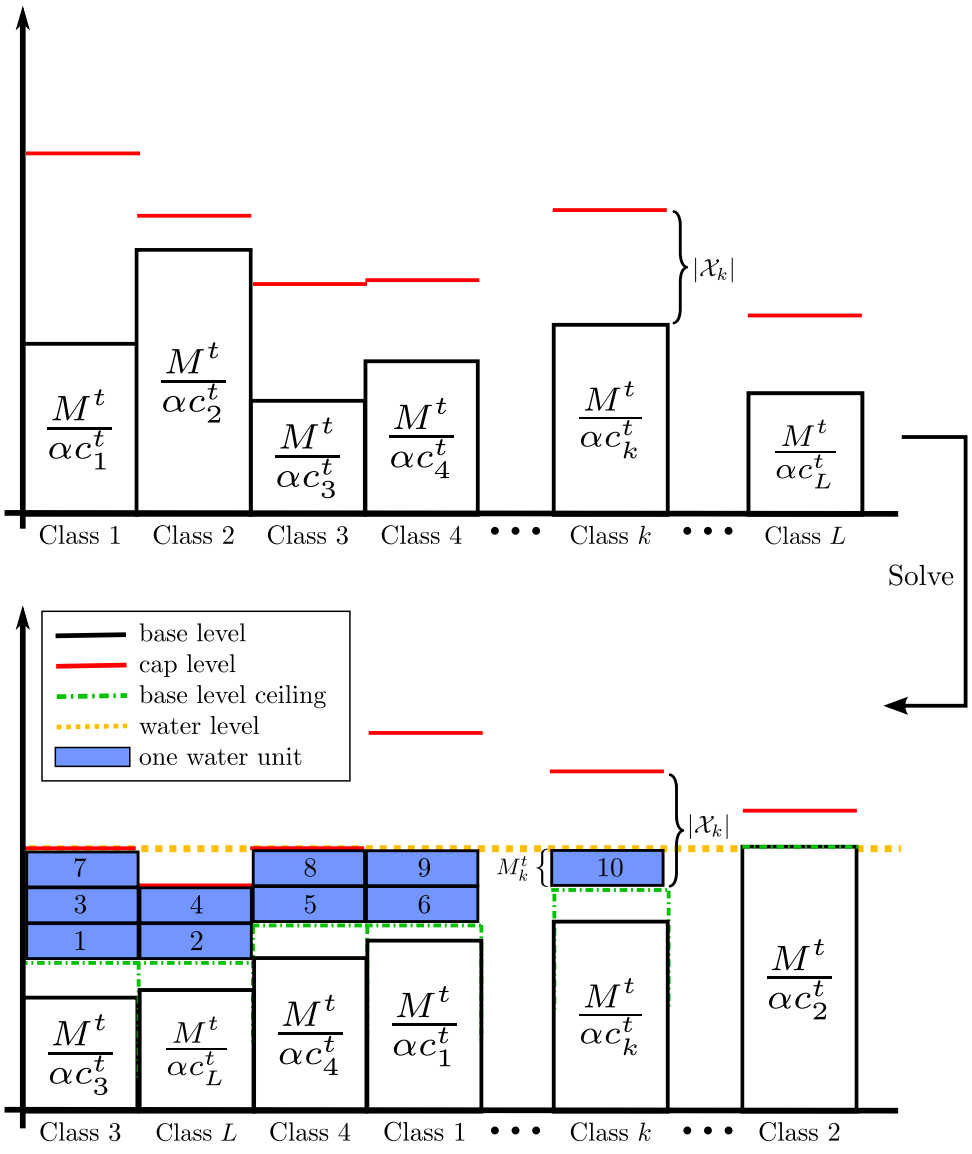

Figure 4.1: Integer water-filling with caps.

a diverse representative subset of each class separately. The budget selection algorithm in Theorem 4.1 allows us to do so by distributing our original budget M^t amongst the various classes. We can therefore solve L independent problems: one for each class. We drop the class subscript k and assume that we would like to select a subset $\mathcal{B}^t$ from a pool of samples $\mathcal{X}$, where all the samples belong to the same class. For notational convenience, we also drop the time superscript t, with the understanding that this procedure will be repeated at every time step.

4.2.3 Subset diversity

As per (O3), we would like to select a *diverse* subset, i.e., one that does not have too much redundancy. To this end, we assume we have a distance metric $d(\cdot, \cdot)$ such that $d(X_i, X_j)$ represents the distance between samples X_i and X_j. This can, for example, be the Euclidean distance between X_i and X_j, or the Euclidean distance between their feature vectors, in some pre-defined feature space. In order to maximize diversity, we seek to **maximize** the average distance between all selected samples, i.e., find $\mathcal{B}$ such that:

$$\frac{1}{M^2} \sum_{X \in \mathcal{B}} \sum_{X' \in \mathcal{B}} d(X, X') \tag{4.4}$$

is maximized,[1] where M is the budget allocated by the water-filling algorithm. Let $N = |\mathcal{X}|$, the training set size of the class under consideration. We introduce a

[1]Other objective functions can be formulated such as maximizing the minimum distance between selected samples. While guaranteeing less redundancy, such objective functions are more difficult to solve.

binary variable $\mathbf{s} \in \{0,1\}^N$, such that $s_i = 1$ if $X_i \in \mathcal{B}$, and $s_i = 0$ otherwise. We also group all the distances in a matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ such that $\mathbf{D}_{ij} = d(X_i, X_j)$. As such, the objective in (4.4) can be re-written as

$$\max_{\mathbf{s} \in \{0,1\}^N} \frac{1}{M^2} \mathbf{s}^\intercal \mathbf{D} \mathbf{s}. \tag{4.5}$$

This problem formulation ensures that the chosen samples are sufficiently distant from each other.

4.2.4 Subset representativeness

Per (O4), we would also like to select a *representative* subset $\mathcal{B}$, i.e., the non-selected samples must be well represented by the set $\mathcal{B}$. To this end, we seek to **minimize** the average distance between selected and non-selected samples. As before, this can be re-written as

$$\min_{\mathbf{s} \in \{0,1\}^N} \frac{1}{M(N-M)} (\mathbf{1} - \mathbf{s})^\intercal \mathbf{D} \mathbf{s}, \tag{4.6}$$

where $\mathbf{1}$ is the vector of all ones in $\mathbb{R}^N$.

4.2.5 Joint formulation

As previously mentioned, once the sub-problem of allocating budgets to each class has been solved, we seek to solve L independent problems of finding a diverse, representative subset over which the classifier performs poorly. We therefore combine

the subset diversity, representativeness, and uncertainty criteria. We define the vector $\mathbf{c} \triangleq [c(X_1), \ldots, c(X_N)]^\intercal$ where $c(\cdot)$ is as defined in Section 4.2.1. To make the quantities comparable, we normalize $\mathbf{D}$ and $\mathbf{c}$ such that all their elements lie in $[0, 1]$. We denote the normalized quantities by $\tilde{\mathbf{D}}$ and $\tilde{\mathbf{c}}$ respectively. Our objective function is:

$$\max_{\mathbf{s}\in\{0,1\}^N} \quad \lambda_d \cdot \underbrace{\frac{1}{M^2}\mathbf{s}^\intercal\tilde{\mathbf{D}}\mathbf{s}}_{\text{diversity}} \quad - \quad \lambda_r \cdot \underbrace{\frac{1}{M(N-M)}(\mathbf{1}-\mathbf{s})^\intercal\tilde{\mathbf{D}}\mathbf{s}}_{\text{representativeness}} \quad + \quad \lambda_u \cdot \underbrace{\frac{1}{M}\tilde{\mathbf{c}}^\intercal\mathbf{s}}_{\text{uncertainty}}, \tag{4.7}$$

where $\lambda_d, \lambda_r, \lambda_u \geq 0$ are parameters which dictate the relative importance of each criterion. We need to add the constraint that $|\mathcal{B}| = M$, where M is the budget allocated by the water-filling algorithm. The joint optimization problem for each class is therefore:

$$\begin{aligned} \min_{\mathbf{s}\in\{0,1\}^N} \quad & -\lambda_d \cdot \frac{1}{M}\mathbf{s}^\intercal\tilde{\mathbf{D}}\mathbf{s} + \lambda_r \cdot \frac{1}{N-M}(\mathbf{1}-\mathbf{s})^\intercal\tilde{\mathbf{D}}\mathbf{s} - \lambda_u \cdot \tilde{\mathbf{c}}^\intercal\mathbf{s} \\ \text{s.t.} \quad & \mathbf{1}^\intercal\mathbf{s} = M. \end{aligned} \tag{4.8}$$

It is important to note that the division of our problem into L independent sub-problems provides many advantages. First, formulating the problem on the entire training dataset would require a very large distance matrix $\mathbf{D}$ which would, in most cases, need excessive storage and be computationally prohibitive. Second, the L sub-problems are completely independent and can run in parallel, thus reducing computation time.

4.2.6 Proposed solution

The problem in (4.8) is not convex for two reasons: (i) the set $\{0,1\}^N$ is finite and therefore not convex, and (ii) the objective function is generally not convex. We change the constraint $\mathbf{1}^\intercal\mathbf{s} = M$ to its equivalent $(\mathbf{1}^\intercal\mathbf{s} - M)^2 = 0$ (as this guarantees zero duality gap [73]) and make the change of variable $\mathbf{x} = 2\mathbf{s} - \mathbf{1}$, where $\mathbf{x} \in \{-1,1\}^N$. Let

$$\mathbf{A} \triangleq -\left(\frac{\lambda_d}{4M} + \frac{\lambda_r}{4(N-M)}\right)\tilde{\mathbf{D}}, \tag{4.9}$$

$$\mathbf{b} \triangleq -\frac{\lambda_d}{2M}\tilde{\mathbf{D}}^\intercal\mathbf{1} - \frac{\lambda_u}{2}\tilde{\mathbf{c}}. \tag{4.10}$$

Lemma 4.1 *An equivalent problem to (4.8) is given by:*

$$\begin{aligned} \min_{\mathbf{x}\in\{-1,1\}^N} \quad & \mathbf{x}^\intercal\mathbf{A}\mathbf{x} + \mathbf{b}^\intercal\mathbf{x} \\ \text{s.t.} \quad & (\mathbf{1}^\intercal\mathbf{x} - 2M + N)^2 = 0. \end{aligned} \tag{4.11}$$

This problem is known as *constrained binary quadratic programming* and is NP-hard [73]. We seek an efficient relaxation to this problem.

Theorem 4.2 *The solution* $\mathbf{x}^*$ *to (4.11) can be well-approximated by*

$$\hat{\mathbf{x}}^* = -\frac{1}{2}\left(\mathbf{A} + \mu^*\mathbf{1}\mathbf{1}^\intercal + \gamma^*\mathbf{I}\right)^\dagger\left(\mathbf{b} - 2\mu^*(2M-N)\mathbf{1}\right), \tag{4.12}$$

where $(\cdot)^\dagger$ *denotes the pseudo-inverse,* $\mathbf{I}$ *denotes the identity matrix in* $\mathbb{R}^{N\times N}$*, and* μ^*, γ^* *are the solution to the following semi-definite program (SDP):*

$$\begin{aligned} &\max_{\mu,\gamma,\tau\in\mathbb{R}} \quad (2M-N)^2\mu - \gamma N - \tau \\ &\text{s. t.} \begin{pmatrix} \tau & \frac{1}{2}(\mathbf{b} - 2\mu(2M-N)\mathbf{1})^\intercal \\ \frac{1}{2}(\mathbf{b} - 2\mu(2M-N)\mathbf{1}) & \mathbf{A} + \mu\mathbf{1}\mathbf{1}^\intercal + \gamma\mathbf{I} \end{pmatrix} \succeq 0 \end{aligned} \tag{4.13}$$

We select the samples corresponding to the largest M entries in $\hat{\mathbf{x}}^*$.

4.2.7 Overall algorithm

Algorithm 4.2 Proposed algorithm.

1: Set $t \leftarrow 0$
2: Initialize classifier
3: Test initial classifier on training data to obtain uncertainty levels $\{\mathbf{c}_k^0\}_{k=1}^L$
4: **repeat**
5: Using $\{\mathbf{c}_k^t\}$, apply Algorithm 4.1 to obtain class budgets $\{M_k^t\}$
6: **for all** $k \in \{1, \ldots, L\}$ **do**
7: (in parallel) Solve (4.12) with $M = M_k^t$, $N = N_k^t$, to obtain class batch selection $\mathcal{B}_k^t$
8: **end for**
9: Set $\mathcal{B}^t \leftarrow \cup_{k=1}^L \mathcal{B}_k^t$
10: Resume classifier training on $\mathcal{B}^t$
11: Test resulting classifier on training data to obtain new uncertainty levels $\{\mathbf{c}_k^{t+1}\}$
12: Set $t \leftarrow t + 1$
13: **until** Stopping criterion met

In this section, we explain the overall batch subset selection pipeline. We start from a classifier with a fixed architecture. First, we test this classifier on the entire pool of training examples and compute the initial average uncertainty levels $\{c_k^0\}_{k=1}^L$.

Then, at every time step t, we use $\{c_k^t\}_{k=1}^L$ to obtain a class-specific budget using Algorithm 4.1 and solve (4.12) and (4.13) independently for each class, resulting in a new selected batch. We resume training on the union of all selected batches. At each time step, all candidate samples have a chance to be selected, i.e., previously selected examples are not removed from the set of candidates. We iterate until a stopping criterion is met. The overall algorithm is described in Algorithm 4.2 and illustrated in Figure 4.2.

4.3 Experiments

In this section, we test the proposed method on several real-world classification tasks. We compare our approach to the random selection of training samples, i.e., ordinary training algorithms. Our formulation does not assume a specific classifier structure. However, we will illustrate our results on deep neural networks as they are the current state-of-the-art. We use the Caffe framework [32] for the implementation of Convolutional Neural Networks (CNN) as well as the SDPA framework [78] to solve the SDP problem in (4.13). We calculate distances between samples based on the Local Binary Patterns (LBP) features [79].

For each of these experiments, unless otherwise specified, we start from a randomly initialized CNN with a fixed architecture, and apply Algorithm 4.2.

We do not employ any type of data pre-processing or augmentation techniques which are widely used to achieve state-of-the-art performance, since these methods are not the focus of this work. Instead, we choose to focus on the effect of our training

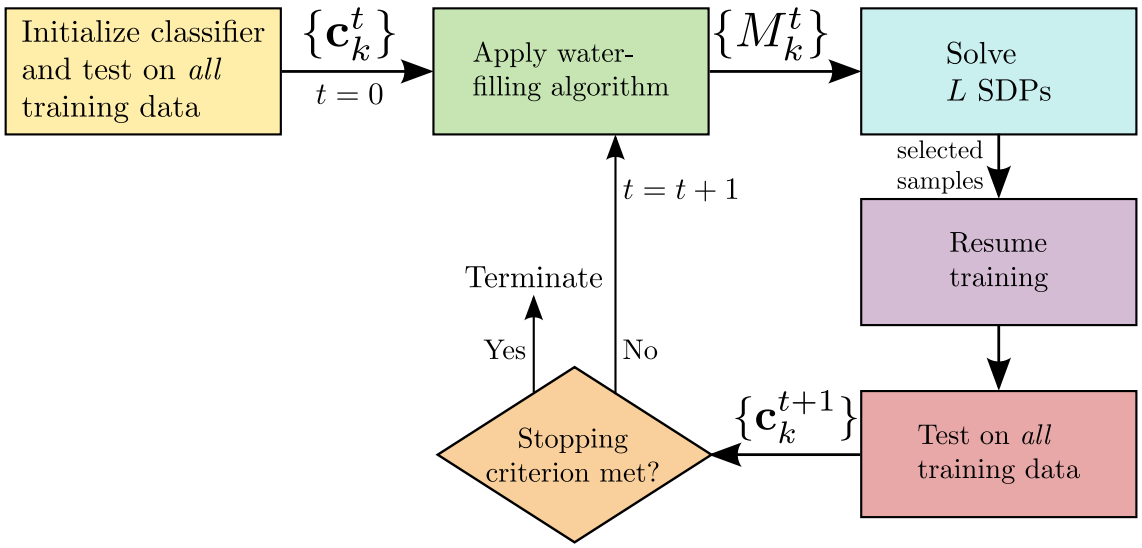

Figure 4.2: Proposed algorithm.

set selection method on the generalization performance, compared to picking the training samples in a random fashion. As our training is incremental, we add dropout layers [35] whenever necessary to combat the problem of catastrophic forgetting in deep neural networks. Catastrophic forgetting refers to the inability of a learning method to preserve previously learnt information when exclusively trained on new data [80, 81].

4.3.1 MNIST digit recognition

We start our experiments with examples from the well-known MNIST digit recognition dataset. We use the LeNet architecture [82] and run our experiments using a randomly selected subset of the MNIST dataset consisting of 1000 images from each class. We use a total budget of 50 training images per one loop of our algorithm (see Figure 4.2), i.e., for all t, $\sum_{k=1}^{L} M_k^t = 50$.

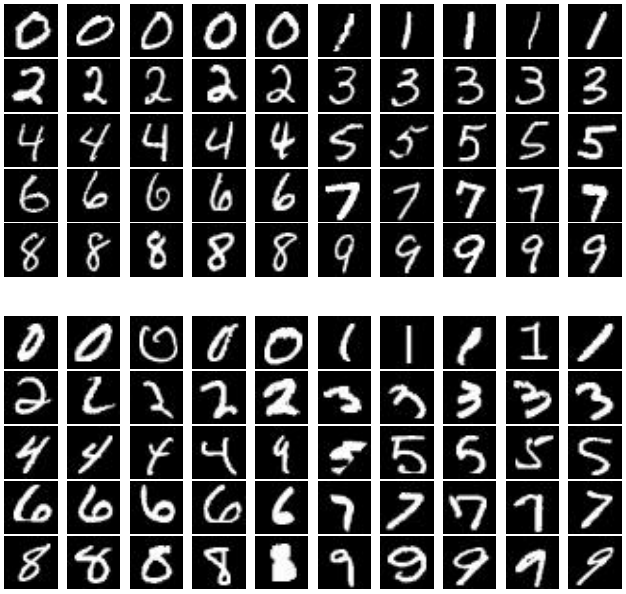

Figure 4.3: Top: Selected samples when $\lambda_r = 20\lambda_d$. Bottom: Selected samples when $\lambda_r = \lambda_d$.

4.3.1.1 Effect of λ_d vs. λ_r

We first illustrate the effect of the weights λ_d, λ_r, defined in (4.7), on the selection process. We set $\lambda_u = 0$. Figure 4.3 shows the selected samples when $\lambda_r = 20\lambda_d$ (top) and $\lambda_r = \lambda_d$ (bottom). When λ_r is large, more representative samples are chosen, as seen in Figure 4.3, top. When $\lambda_d = \lambda_r$, more diverse samples are chosen. This validates our initial objective formulation in (4.7).

4.3.1.2 Performance on clean data

We compare our method of adaptively selecting training batches to the baseline of random selection. In [58], it is suggested that introducing "easier" samples first and gradually increasing the difficulty results in improved performance. Our method enables us to implement such an approach by modifying the relative values of λ_d, λ_r,

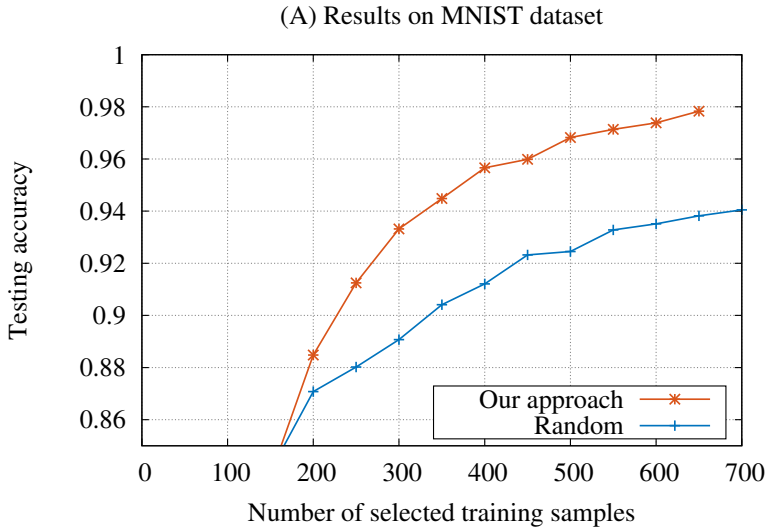

Figure 4.4: Results on MNIST dataset with clean labels.

and λ_u over time. We achieve this by keeping λ_d fixed, and starting with $\lambda_r = 10\lambda_d$ and $\lambda_u = 0$. Picking a large λ_r puts more weight on the representativeness term in (4.7) and thus ensures that outliers are not picked. We gradually decrease λ_r and increase λ_u in order to allow for more difficult examples to be sampled. We present our findings in Figure 4.4. Our approach outperforms random selection by a margin of 4%. Furthermore, the number of samples required by our proposed method to reach a target performance level is much smaller than random sampling. For instance, for a target testing accuracy of 94%, around 700 samples are needed for random as opposed to less than 350 samples for our approach.

We now assess the quality of the local minimum obtained with our method. From Figure 4.4, our method achieved around 98% accuracy using 650 selected samples. Using these *same* 650 images, we train a CNN from scratch using random sampling. This results in a testing accuracy of 96.3%, inferior to the one obtained by

our method (but superior to the 94% accuracy obtained using a random sampling of 650 images out of the original pool of 10,000 training images). These two CNNs have seen the exact same data, the only difference being the order in which it was introduced. This validates the claim that adaptive selection of training data guides the neural network towards a better local optimum. Our algorithm has selected easier training samples in the first few iterations, and more difficult samples later on, as dictated by the change in weights $\lambda_d, \lambda_r, \lambda_u$.

4.3.1.3 Performance on data with noisy labels

We now study the performance of our algorithm in the presence of label noise. In 20% and 30% of the training samples, we randomly change the correct label to one of the 9 incorrect labels. The results are shown in Figures 4.5(B) and 4.5(C). In the presence of label noise, our approach was able to out-perform random selection by more than 5%. To achieve this, we decrease the diversity weight λ_d and adopt a more "cautious" approach by increasing λ_u at a slower pace. This results in a slower but safer update of the network. In fact, the total number of noisy training images chosen by our algorithm for the case of 20% label noise is 93 images by the 12^{th} loop (or 6.5% of the picked images), whereas random sampling obviously picks around 20% noisy samples.

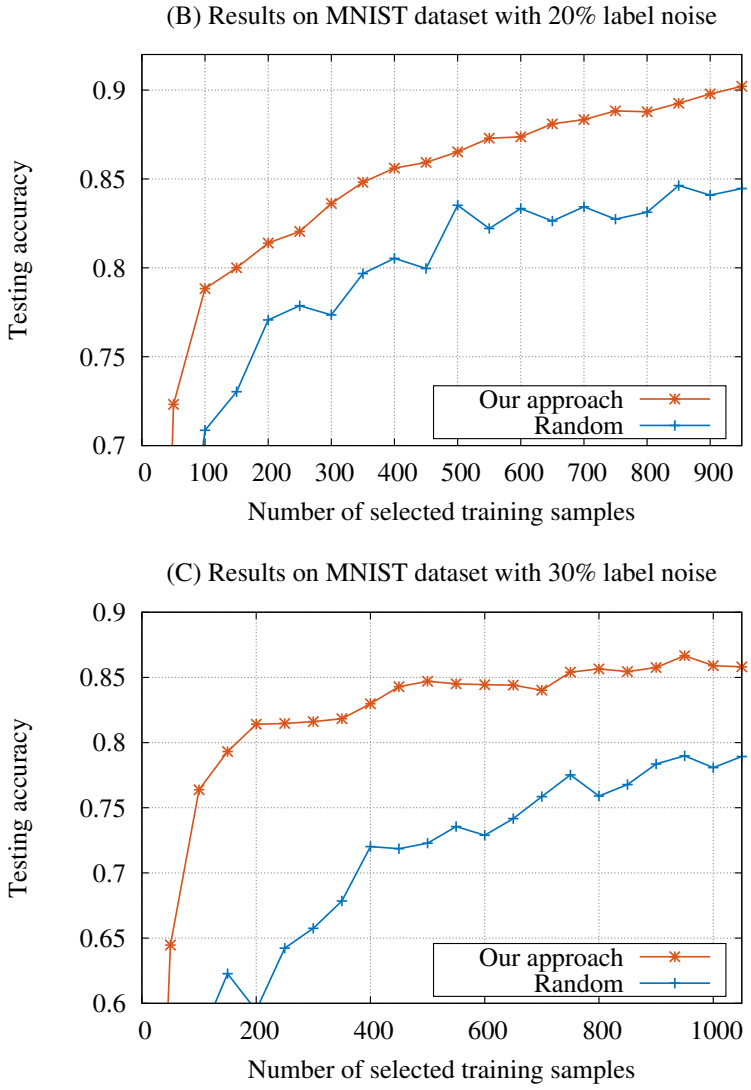

Figure 4.5: Results on MNIST dataset with 20% (top) and 30% (bottom) label noise.

4.3.1.4 Data imbalance

Finally, we test our method on a scenario where there is a significant data imbalance between different classes. This can happen when acquiring labeled data for some classes is considerably more difficult than for others. We artificially introduce data imbalance by picking 4 classes at random and reducing their training set size to between 10 and 20 images per class. Our approach achieves 90.14% testing accuracy after only 9 loops of the algorithm (i.e., 450 picked samples), while random sampling achieves 86.91% using the entire training set. We are thus able to boost the performance by over 3% while only using a fraction of the available samples. In our algorithm, picked samples are not removed from the pool of available training images, thus allowing the network to revisit certain training samples if required. This is especially crucial in the case of data imbalance since random selection has very low probability of selecting images from the down-sampled classes. In contrast, in our method the number of selected samples from each class depends on how well the classifier has learnt that class. Figure 4.6 shows the number of classification mistakes made by a CNN trained with our algorithm and with random selection. Classes 1, 3, 6, and 7 were significantly down-sampled. Our approach allows the CNN to perform well on these classes compared to random sampling.

4.3.2 SUN397 scene recognition

Our second set of experiments is on the SUN397 scene recognition dataset [83]. The dataset consists of 108,754 images from 397 scene categories. We use a randomly

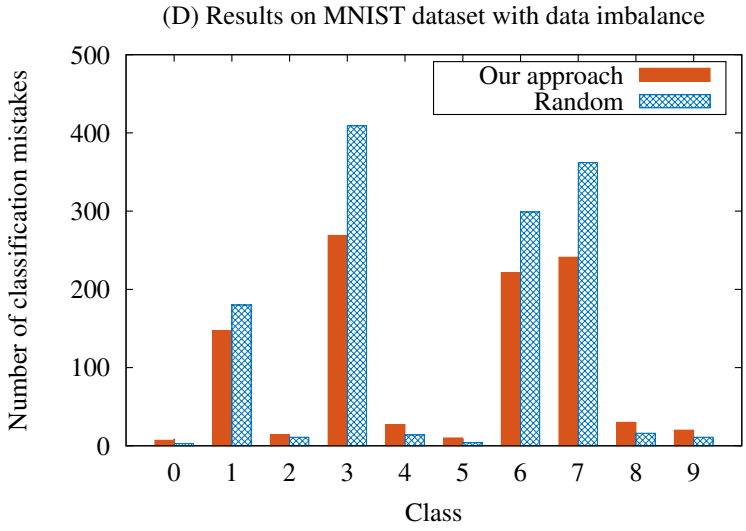

Figure 4.6: Results on MNIST dataset with class imbalance.

initialized CNN with the "AlexNet" architecture [2] as provided by Caffe [32]. We choose a budget of 2000 images per loop and perform five-fold cross-validation using 5 random splits. No data pre-processing was performed except for resizing all images to 227×227.

As before, we keep λ_d fixed, and start with $\lambda_r > \lambda_d$ and $\lambda_u = 0$. With time, we decrease λ_r and increase λ_u, thus putting more emphasis on sample diversity and classifier feedback. In Figure 4.7, we show examples of images from 9 different classes selected by our algorithm. We notice that examples selected at the beginning of the training process (images to the left) are representative of their respective classes. The images to the right, selected later in the process, are more "difficult" and less typical examples of the classes. For example, for class "Operating room", our algorithm picked an image which includes a crowd of people. This is not typically associated with an operating room and makes it significantly more difficult to classify than the

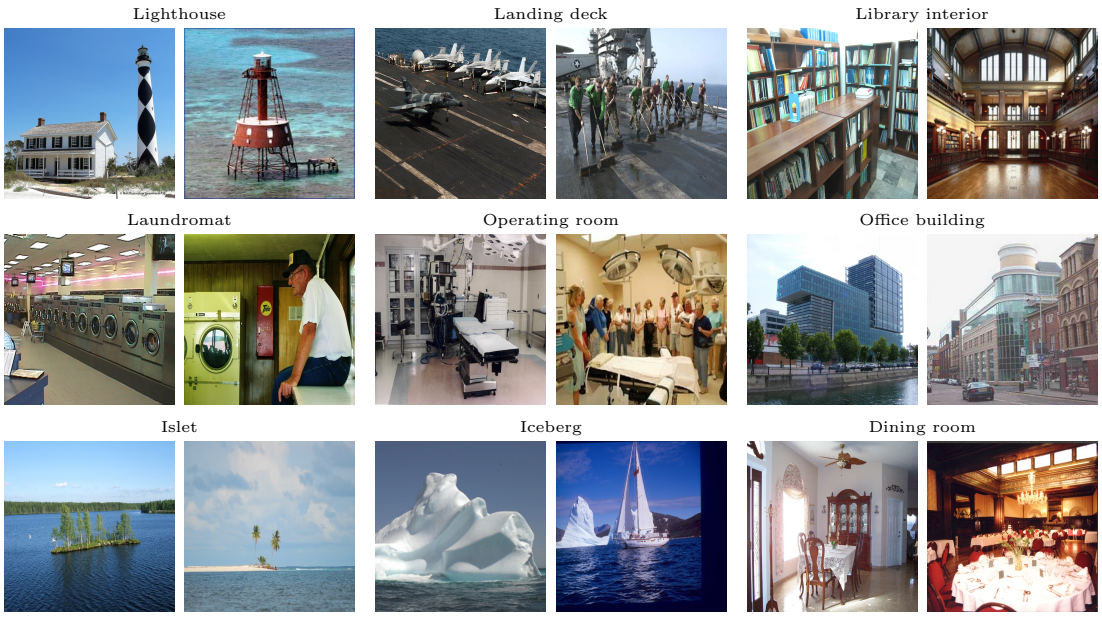

Figure 4.7: Examples of images from various classes of SUN397 picked by our algorithm at the beginning of training (left) and 75% through the training process (right).

picture to its left.

The classification results are reported in Table 4.1. Using our active selection algorithm, we are able to boost the performance on SUN397 by over 10%. Better state-of-the-art results are reported on the SUN397 dataset using AlexNet [84–86]. However, in all of these papers, the networks are trained on much larger datasets such as the 2.5 million scene images from Places [84]. It seems that our method is particularly useful in cases where not a lot of data is available. We also run the same experiments on a subset of SUN397, which we call SUN50, obtained by restricting the dataset to 50 randomly selected classes out of the original 397 scene categories. We see a similar performance boost of over 7%.

4.3.3 ImageNet large-scale object recognition

We test our method on the ImageNet large-scale object recognition dataset (specifically, the ILSVRC2012 classification challenge) [87, 88]. While multiple CNN architectures have been suggested in the literature [1, 2, 20], we consider "AlexNet" [2] as provided by Caffe. The training data consists of around 1.2 million images. State-of-the-art results on this network are obtained by averaging results from multiple crops. As previously mentioned, we do not employ any data augmentation and/or pre-processing techniques and only feed the networks the center crop (227×227). We report testing accuracies on the validation set in Table 4.1.

The results in Table 4.1 clearly show the benefit of the judicial selection of training data. Images introduced later in the learning process affect the weights of the CNN to a lesser extent than those introduced early on, due to the decrease in the learning rate. Therefore, it is intuitive that representative images should be the ones "driving" the network at the early stages of learning. As before, we show examples of such images in Figure 4.8. For each class, images in the left columns are introduced when the learning rate is large, and therefore affect the trained CNN more than images to the right.

4.3.4 VGG Face dataset

In this experiment, we address a face recognition task. We start with a pre-trained CNN in order to illustrate the use of our algorithm for transfer learning, as a fine-tuning sampling strategy. Using the methods and network described in [89],

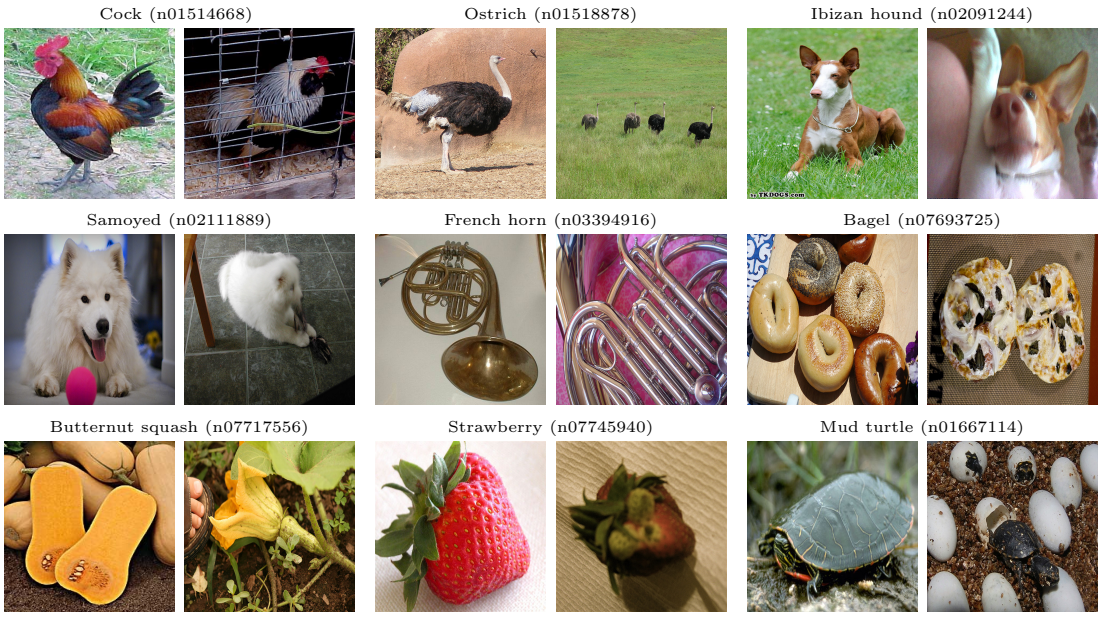

Figure 4.8: Examples of images from various classes of ImageNet picked by our algorithm at the beginning of training (left) and 75% through the training process (right).

a CNN was pre-trained on the CASIA-WebFace dataset [90]. Instead of random initialization, we start with the pre-trained weights for the first 15 layers (up to the fifth pooling layer), and add two randomly initialized fully connected layers joined by a dropout layer. We train and test on the VGG Face dataset [91]. Since CASIA-WebFace and VGG Face have significant subject overlap, we choose 20 of the non-overlapping subjects. The VGG Face dataset consists of a large number of images, out of which a portion has been selected as part of the final curated set. We observe that the non-curated images are considerably more affected by label and bounding box noise. In order to get meaningful test results, we restrict our testing set to the curated images, while training on the entire dataset. We perform five-fold cross-validation using 5 random splits. We choose a budget of 100 training samples per loop of our algorithm.

Figure 4.9 shows some examples of images selected by our algorithm. The top images are chosen in the first loop, when the representativeness score λ_r is large and the uncertainty score λ_u zero. We notice that all chosen samples are frontal, of good quality, and typical of the subjects. The bottom images are chosen much later in the process, after λ_r has considerably decreased and λ_u increased. This time, our method chooses more difficult examples which include extreme poses, obstruction, blur, an additional person, and an unusually young version of the subject.

Dataset	**Architecture and initialization**	**Our approach**	**Random sampling**
MNIST Clean labels	LeNet [82] Random weights	$> \mathbf{97.9}\%$[2]	$> 93.8\%$[2]
MNIST 20% label noise	LeNet Random weights	$> \mathbf{90.0}\%$[3]	$> 84.5\%$[3]
MNIST 30% label noise	LeNet Random weights	$> \mathbf{85.5}\%$[4]	$> 79.0\%$[4]
MNIST Class imbalance	LeNet Random weights	$> \mathbf{90.14}\%$[5]	86.91%
SUN397	AlexNet [2] Random weights	$\mathbf{53.4}\% \pm \mathbf{0.4}$	$42.3\% \pm 0.2$
SUN50	AlexNet Random weights	$\mathbf{68.7}\% \pm \mathbf{0.2}$	$61.1\% \pm 0.3$
ILSVRC2012	AlexNet Random weights	$\mathbf{62.9}\%$	57.1%[6]
VGG Face	As described in [89] Pre-trained	$\mathbf{98.1}\% \pm \mathbf{0.2}$	$94.9\% \pm 0.1$

Table 4.1: Summary of testing accuracies.

We present the performance results of this CNN trained using our algorithm

[2]These experiments were stopped after the selection of 650 samples.
[3]These experiments were stopped after the selection of 950 samples.
[4]These experiments were stopped after the selection of 1050 samples.
[5]This experiment was stopped after the selection of 450 samples.
[6]This result is taken directly from Caffe model documentations.

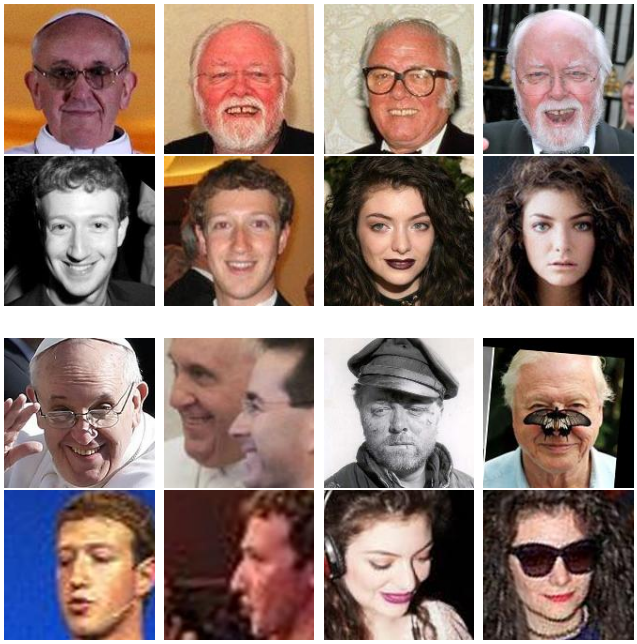

Figure 4.9: Examples of selected samples at the beginning of training (left) and 75% through the training process (right).

and random sampling in Tables 4.1 and 4.2. Using only one loop (i.e., 100 picked images), the testing accuracy increases to 89.69% compared to 80.05% for random sampling. The performance of random sampling saturates at around 94.9%, while our approach achieves 97.15% with only 1600 selected images (which cuts the error in half). Our method eventually achieves 98.09% accuracy vs. 94.92% for random sampling.

# of selected samples	Our approach	Random
0	12.73%	12.45%
100	**89.69%**	80.05%
500	**93.23%**	90.79%
1300	**97.00%**	94.89%
1600	**97.15%**	94.89%

Table 4.2: VGG Face testing accuracies.

4.4 Computational complexity

In this section, we analyze the computational complexity of one loop of Algorithm 4.2. At every loop, the following operations are executed:

- Solve Algorithm 4.1 – line 5 in Algorithm 4.2.
- Solve L instances of independent SDPs given by equation (4.13) – line 7 in Algorithm 4.2.
- Solve L instances of equation (4.12), and for each instance find the M largest entries – line 7 in Algorithm 4.2.
- Test classifier on training data – line 11 in Algorithm 4.2.

We will analyze the computational complexity of each step independently in the following subsections. All our experiments were performed on an Intel(R) Xeon(R) CPU E5-2623 v3 @ 3.00GHz, with 16 CPU cores and 32GB of memory, equipped with 2 GeForce GTX TITAN X GPUs with 12GB of memory each.

4.4.1 Solve Algorithm 4.1

The first step in Algorithm 4.1 is sorting the base levels which takes $\mathcal{O}(L \log L)$ operations. The other steps have a linear complexity in L. The overall complexity is therefore linearithmic in the number of classes L: $\mathcal{O}(L \log L)$.

4.4.2 Solve L instances of independent SDPs

Most robust and widely-used algorithms to solve SDPs are known as interior-point methods and may require $\mathcal{O}(mn^3 + m^2n^2)$ operations in the worst case [92], where m is the number of optimization variables and n is the size of the matrices. In our case, $m = 3$ is the number of optimization variables in Problem (4.13) and $n = N + 1$, where N is the number of training samples per class. In our experiments, the largest N was around 1000. In practice, it is observed that the number of operations required to solve an SDP grows much slower than the theoretical worst-case bound and that it is not much harder to solve an SDP than it is to solve Linear Programs (LPs) [93]. Furthermore, our experience solving multiple SDPs for large-scale problems confirms these findings. In fact, the SDPA framework [78] we use in our implementation reports feasible running times for solving problems of much larger scale in [94, Table 5]. For example, benchmark problem qpG51, with $m = 1000, n = 2000$ is solved in 52.2 seconds for an accuracy (dual and primal relative gap and feasibility error) of 10^{-7}. We refer the reader to [94] for implementation details. All the SDP instances we encounter are considerably smaller than qpG51. This is especially true since our SDP formulation only involves 3 optimization variables (please refer to Remark C.2 in Appendix C for details on our choice of SDP relaxation). Furthermore, an accuracy of 10^{-7} is not needed in practice. We observe that decreasing this accuracy beyond 10^{-3} does not change the selected samples.

While our algorithm requires solving L SDPs, this process can be parallelized

to the extent of available cores, therefore substantially reducing the running time. In our experiments, we only utilize 16 CPU cores. A CPU cluster would significantly speedup the process, since the SDPs are completely independent and no data transfer is necessary while solving them. For reference, solving all 397 SDPs for the SUN397 dataset on our experiment setup takes less than 2 seconds.

Additionally, the SDP computation time does not depend on the budget M. Having a large batch size budget means SDPs are solved less frequently as each batch needs longer training time. The disadvantage of a large budget is less frequent feedback from the classifier. On the other hand, selecting a small budget introduces the SDP computational overhead more frequently, but has the benefit of providing more immediate feedback from the classifier. This trade-off is important in balancing system requirements and computational complexity. In our experiments, we investigated a variety of budget sizes ranging from an average of 5 per class (MNIST) to over 50 per class (ImageNet). One viable strategy might be to vary budget sizes as training progresses, which was not explored in this chapter and will be the focus of future work.

It is worth noting that other methods for solving SDPs exist and are more suitable for large-scale problems [92]. While we do not employ them in our implementation, this can be pursued if computation times become prohibitive.

4.4.3 Solve L instances of equation (4.12) and find M largest entries

Equation (4.12) involves computing a Moore-Penrose pseudo-inverse which may also require $\mathcal{O}(N^3)$ operations in the worst case, where N is the number of training samples per class. In our experiments, it was also observed that this computation was not prohibitive and the running time was dominated by solving the SDPs. As before, the L pseudo-inverse computations can be done completely in parallel. In the case that faster pseudo-inverse calculations are necessary, [95] shows that they can be solved efficiently using GPUs. Finding the M largest entries of the solution to (4.12) can be done efficiently in linear time.

4.4.4 Test classifier on training data

This step consists of running a forward pass of the CNN on the entire training dataset. This is done in batch mode using GPUs. For the SUN397 dataset, the running time is about 57 seconds, while for ImageNet it is about 12 minutes. This is an expensive operation and can be further optimized. First, as this step is performed at every loop, if SDPs are solved less frequently, the testing step will also be executed more sporadically. The same discussion regarding the feedback-complexity trade-off applies. Secondly, testing on the entire dataset may not be necessary assuming uncertainty levels only vary significantly for training samples in the vicinity of the previous training batch. We can, therefore, restrict the testing to a neighborhood around the previously selected batch and only update the uncertainty levels in those neighborhoods (neighborhoods can be found efficiently using the pre-computed

distance matrices).

4.4.5 One-time computation

In addition to the steps outlined above, which are performed at every loop of the algorithm, distance matrices for each of the L classes are computed only once at the beginning of the training process. This computation takes $\mathcal{O}(N^2)$ operations per class, where N is the number of training samples per class, and, as before, can be run in parallel. This computation is also amenable to a GPU implementation.

4.5 Concluding remarks

In this chapter, we proposed a novel approach which adaptively selects training data to be presented to a classifier. The approach is based on balancing four objectives: class balance, data representativeness, data diversity, and classifier uncertainty. We developed an efficient iterative and adaptive algorithm based on convex optimization. We demonstrated its effectiveness on several real-life classification datasets as well as its robustness to label noise and class imbalance. We showed that our algorithm is suitable for a wide range of applications, including face, scene, and object recognition. We were able to out-perform random selection in all of our experiments. This emphasizes the important role of the order in which data is presented to CNNs in its generalization ability.

Chapter 5: Task-aware compressed sensing with generative adversarial networks

5.1 Overview

In this chapter, we relax the last ideal condition considered in this dissertation. When real signals or images cannot be accessed, and instead lossy versions (such as downsampled or sensed inputs) are available, we develop a technique to recover the original signal as well as carry out usual supervised learning tasks.

The broad scope and generality of the field of compressed sensing has led to many impressive applications, such as rapid magnetic resonance imaging [8], single-pixel camera [9] and UAV systems. The core problem of compressed sensing is that of efficiently reconstructing a signal $\mathbf{x} \in \mathbb{R}^n$ from an under-determined linear system of noisy measurements given by

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\zeta} \tag{5.1}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the measurement sensing matrix, $m < n$, and $\boldsymbol{\zeta} \in \mathbb{R}^m$ is the measurement noise [96]. Since this is an under-determined system of equations, a unique solution does not exist, even in the absence of noise, unless some assumptions

are made on the structure of the unknown vector $\mathbf{x}$. Depending on applications, the structural assumptions may vary, the most common one being that $\mathbf{x}$ is sparse [96–98]. Under this specific assumption, the problem of recovering $\mathbf{x}$ has been widely studied, and different conditions on the matrix $\mathbf{A}$ have been established to guarantee reliable recovery [99]. These conditions include the Restricted Isometry Property (RIP) or the Restricted Eigenvalue Condition (REC) [42, 100].

Even though the sparsity assumption on $\mathbf{x}$ is the most common choice, it is not the only possible one. Indeed, other approaches, such as combining sparsity with additional model-based constraints [101] or graph structures [102], have been developed. Recently, in [10], a *generative model* was used and the unknown signal $\mathbf{x}$ was assumed to be the output of this model. Generative models have been successfully used to model data distributions, and include the variational auto-encoder (VAE) [103], generative adversarial networks (GANs) [15], and variations thereof [104–106]. In the GAN framework, two models are trained simultaneously in an adversarial setting: a generative model that emulates the data distribution, and a discriminative model that predicts whether a certain input came from real data or was artificially created. The generative model learns a mapping G from a low-dimensional vector $\mathbf{z} \in \mathbb{R}^k$ to the high dimensional space $\mathbb{R}^n$.

The authors in [10] use a pre-trained generative model G, and recover an estimate of $\mathbf{x}$ from the compressed measurements $\mathbf{y}$, assuming it is in the range of G. To this end, the following optimization problem is solved:

$$\begin{aligned} \min_{\hat{\mathbf{x}},\mathbf{z}} \quad & ||\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}||_2^2 \\ \text{s. t.} \quad & \hat{\mathbf{x}} = G(\mathbf{z}) \end{aligned} \tag{5.2}$$

In the setting of [10], the pre-trained G is unaware of the compressed sensing framework. Furthermore, it is assumed that an abundance of real (non-compressed) images is available to train G, which, depending on the application, may not be a realistic assumption [107]. After all, the aim of compressed sensing is to recover signals through the use of compressed measurements. In this chapter, we propose to train G specifically for the task of recovering compressed measurements. This makes our GAN task-aware, and improves the compressed sensing performance. Our approach will also address the case where no or very little non-compressed data is available for training, by complementing the training set with compressed training data. Finally, we empirically demonstrate that the low-dimensional latent vector $\mathbf{z}$ can be used, not only to perform reconstruction via G, but also for inference tasks such as classification.

Contributions.

1. We train the GAN in a task-aware fashion allowing it to be specifically optimized for the reconstruction task. We show that this consistently improves the reconstruction error obtained in [10] for various values of the number of measurements m.

2. We consider training using a combination of a small number of (or no) non-

compressed data and a larger set of compressed training data. This is achieved by introducing a second discriminator specifically for compressed data.

3. We show that we can regularize the latent space of $\mathbf{z}$ to make it discriminative, given a desired inference task.

5.2 Related work

In this work, we combine compressed sensing and generative models to perform reconstruction and classification tasks. To this end, we explain the related work in two parts. The first part addresses the use of generative models for reconstruction and classification tasks, and the second part reviews inference tasks in compressed sensing.

Using a generative model for reconstruction tasks is a fairly well-researched area. One line of work attempts to map an image to the range of the generator [108–110]. Unlike our setting, complete and non-compressed knowledge of the images is assumed. In [110], gradient descent (GD) is used to project the image samples onto the latent space of a pre-trained generative model. In [108,109], an inverse mapping between the input space of $\mathbf{x}$ and the latent space of $\mathbf{z}$ is jointly learned along with the generator in an adversarial setting. Generative models can also be used for classification tasks [106, 111–113]. This can be achieved by modifying the discriminator of the GAN to also output class probabilities [112, 113] or augmenting the loss function with discriminative features at training time [106, 111]. Such discriminative features include ground-truth class labels [111] and representations learned by a pre-trained

classifier [106].

Another related line of work considers compressed sensing frameworks for various machine learning and computer vision tasks [114–118]. In [117] theoretical results are provided showing that inference can be done directly in the compressed domain. Of particular relevance to our work are [114, 118, 119] which develop various techniques for the classification of compressed images. These methods operate directly on the compressed measurements, whereas we perform classification on the latent variable $\mathbf{z}$.

Finally, one last research area that is relevant to our application is super-resolution, the task of increasing the resolution of an image. This can be seen as a special case of compressed sensing where the sensing matrix $\mathbf{A}$ averages neighboring pixels. In [120], a sparse representation of image patches is sought and used to obtain a high-resolution output. Our framework adopts the generative model instead of the sparsity constraint. More recent work uses deep convolutional networks [121, 122].

5.3 Model description

5.3.1 Background information

Before describing our approach, we provide some necessary background information on compressed sensing and GANs.

In compressed sensing, the measurements are given as $\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\zeta}$. $\mathbf{A} \in \mathbb{R}^{m \times n}$ is the measurement matrix and is usually chosen to be a Gaussian random matrix because it satisfies desirable properties with high probability [96]. Unless otherwise

specified, we will assume that $\mathbf{A}$ is a zero-mean random Gaussian matrix with independent and identically distributed entries. $\mathbf{A}$ is kept constant in a given experiment.

GANs consist of two neural networks, G and D. $G : \mathbb{R}^k \to \mathbb{R}^n$ maps a low-dimensional latent space to the high dimensional sample space of $\mathbf{x}$. D is a binary neural network classifier. In the training phase, G and D are typically learned in an adversarial fashion using actual input data samples $\mathbf{x}$ and random vectors $\mathbf{z}$. An isotropic Gaussian prior is usually assumed on $\mathbf{z}$. While G learns to generate outputs $G(\mathbf{z})$ that have a distribution similar to that of $\mathbf{x}$, D learns to discriminate between "real" samples $\mathbf{x}$ and "fake" samples $G(\mathbf{z})$. D and G are trained in an alternating fashion to minimize the following min-max loss [15]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{5.3}$$

GAN training algorithm At every iteration, (5.3) is maximized over D for a fixed G, using GD, and then minimized over G, fixing D.

5.3.2 Motivation

In [10], a generative model is pre-trained on a set of uncompressed training images, using the algorithm described in [104]. In the testing phase, the generative model is used to reconstruct a compressed, previously unseen, test image using GD on the problem in (5.2). It is shown that, when $\mathbf{A}$ is a random Gaussian matrix, if $\hat{\mathbf{z}}$

minimizes $||\mathbf{A}G(\mathbf{z}) - \mathbf{y}||_2$ to within additive ϵ of the optimum, then for all $\mathbf{x}$, and with high probability

$$||G(\hat{\mathbf{z}}) - \mathbf{x}||_2 \leq 6 \min_{\mathbf{z}} ||G(\mathbf{z}) - \mathbf{x}||_2 + 3||\boldsymbol{\zeta}||_2 + 2\epsilon \tag{5.4}$$

In other words, the observed reconstruction error is bounded by the minimum possible error of any vector in the range of the generator with some additional terms due to noise and GD precision. We note that this upper bound depends on how well G can represent the unknown signal $\mathbf{x}$. Now, we show that, under certain conditions, the expected value of this error term converges to 0 as G is trained on $\mathbf{x}$.

Theorem 5.1 *Let G_t be the generator of a GAN after t steps of the GAN training algorithm described above. Additionally, as in [15], we assume:*

(i) G and D have enough capacity to represent the data.

(ii) At each update, D reaches its optimum given G.

(iii) At each update, G is updated to improve the min-max loss in (5.3).

Furthermore, we assume that the training samples $\mathbf{x}$ come from a continuous distribution with compact support. Then,

$$\lim_{t \to \infty} \mathbb{E}_{\mathbf{x}} \left[\min_{\mathbf{z}} ||G_t(\mathbf{z}) - \mathbf{x}||_2 \right] = 0 \tag{5.5}$$

This theorem shows that the right-hand side of (5.4) is actually small, which justifies the setup adopted in [10]. However, the conditions for Theorem 5.1 may be too strict in practice. For example, [15] assume that at every step of adversarial training, the discriminator D is allowed to reach its optimal value given G, which might be numerically infeasible. Therefore, the convergence of $||G(\hat{\mathbf{z}}) - \mathbf{x}||_2$ might not be computationally attainable. To this end, we consider a task-aware GAN training, which allows G to be optimized specifically for the task of reconstructing compressed measurements.

Algorithm 5.1 Task-aware GAN training algorithm.

for number of training iterations **do**
  Sample a batch of $\mathbf{s}$ training examples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\mathbf{s})}\}$.
  For all i, compute $\mathbf{y}^{(i)} = \mathbf{A}\mathbf{x}^{(i)} + \boldsymbol{\zeta}^{(i)}$.
  Initialize $\mathbf{s}$ random latent variables $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(\mathbf{s})}\}$ using a zero-mean Gaussian prior.
  Initialize D and G.
  for L steps **do**
    For all i, update $\mathbf{z}^{(i)}$ by GD on the loss:

$$||\mathbf{y}^{(i)} - \mathbf{A}G(\mathbf{z}^{(i)})||_2^2 + \lambda_{\text{prior}}||\mathbf{z}^{(i)}||_2^2 \tag{5.6}$$

  end for
  Update the discriminator by GD on the loss:

$$-\frac{1}{\mathbf{s}}\sum_{i=1}^{\mathbf{s}}[\log D(\mathbf{x}^{(i)}) + \log(1 - D(G(\mathbf{z}^{(i)})))] \tag{5.7}$$

  Update the generator by GD on the loss:

$$\frac{1}{\mathbf{s}}\sum_{i=1}^{\mathbf{s}}\log(1 - D(G(\mathbf{z}^{(i)}))) \tag{5.8}$$

end for
return $\{\hat{\mathbf{z}}^{(1)}, \hat{\mathbf{z}}^{(2)}, \ldots\}, \hat{G}, \hat{D}$

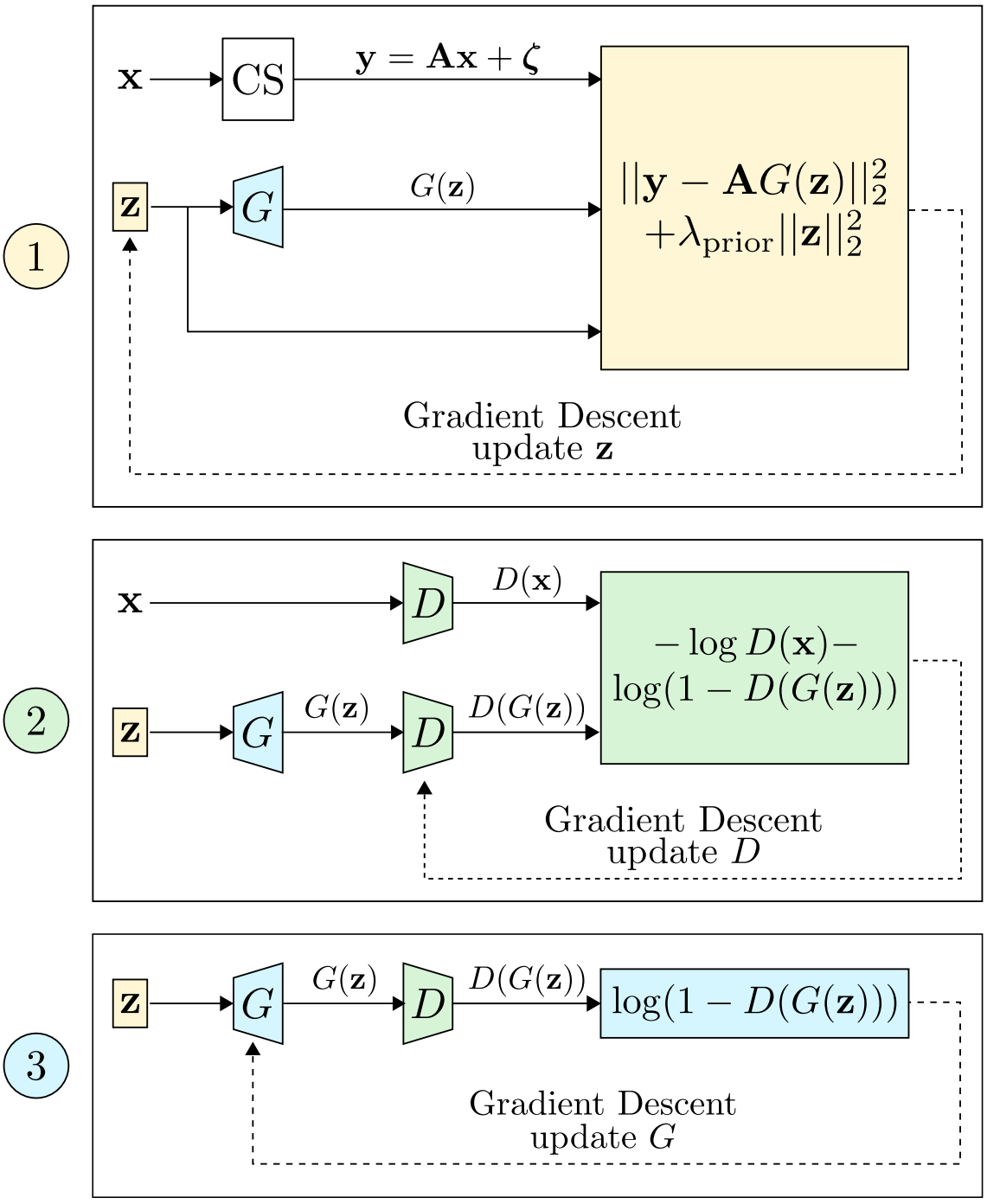

Figure 5.1: One iteration of the task-aware GAN training algorithm.

5.3.3 Task-aware GAN training

To make the GAN training task-aware, we propose to jointly optimize $\mathbf{z}$ and train the GAN using these $\mathbf{z}$'s. This is outlined in Algorithm 5.1, which alternates between three optimizations on $\mathbf{z}$, G, and D, respectively. In particular, we add the GD steps in lines 5-7 of the algorithm to the original GAN training framework. This enables the discriminator and generator to be optimized on values of $\mathbf{z}$ which resemble the ones seen at test time. As previously mentioned, the original GAN training algorithm uses randomly generated $\mathbf{z}$ values to train G and D. However, in our setting, the trained GAN will not be given random $\mathbf{z}$ values at test time, but rather specific $\mathbf{z}$'s selected to minimize a loss function. It is therefore beneficial to train the GAN on $\mathbf{z}$'s obtained through the same process. We note that the extra term $\lambda_{\text{prior}}||\mathbf{z}^{(i)}||_2^2$ in (5.6) comes from the negative log-likelihood of the Gaussian prior on $\mathbf{z}$ [10]. One iteration of this algorithm is also illustrated in Figure 5.1.

5.3.4 GAN training on compressed inputs

As mentioned earlier, a large set of non-compressed training data may not be available in practice. We, therefore, assume that a small (or empty) set of non-compressed training data and a larger set of compressed training measurements are available. We modify the training algorithm to reflect this change. In particular, we train two discriminators and a single generator using a combination of compressed and non-compressed training data. The first discriminator is used to distinguish between actual training data $\mathbf{x}$ and generated data $G(\mathbf{z})$. The second discriminator

Algorithm 5.2 GAN training algorithm using compressed training data.

for number of training iterations **do**
  Sample a batch of $\mathbf{s}_1$ non-compressed training examples $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\mathbf{s}_1)}\}$.
  For all i, compute $\mathbf{y}^{(i)} = \mathbf{A}\mathbf{x}^{(i)} + \boldsymbol{\zeta}^{(i)}$.
  Sample a batch of $\mathbf{s}_2$ compressed training examples $\{\tilde{\mathbf{y}}^{(1)}, \ldots, \tilde{\mathbf{y}}^{(\mathbf{s}_2)}\}$.
  Initialize $\mathbf{s}_1$ random variables $\{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(\mathbf{s}_1)}\}$ and $\mathbf{s}_2$ random variables $\{\tilde{\mathbf{z}}^{(1)}, \ldots, \tilde{\mathbf{z}}^{(\mathbf{s}_2)}\}$ using a zero-mean Gaussian prior.
  for L steps **do**
    For all i, update $\mathbf{z}^{(i)}$ by GD on the loss:

$$||\mathbf{y}^{(i)} - \mathbf{A}G(\mathbf{z}^{(i)})||_2^2 + \lambda_{\text{prior}}||\mathbf{z}^{(i)}||_2^2 \tag{5.9}$$

    For all i, update $\tilde{\mathbf{z}}^{(i)}$ by GD on the loss:

$$||\tilde{\mathbf{y}}^{(i)} - \mathbf{A}G(\tilde{\mathbf{z}}^{(i)})||_2^2 + \lambda_{\text{prior}}||\tilde{\mathbf{z}}^{(i)}||_2^2 \tag{5.10}$$

  end for
  Update the discriminators by GD on the losses:

$$\frac{-1}{\mathbf{s}_1}\sum_{i=1}^{\mathbf{s}_1} \log D_1(\mathbf{x}^{(i)}) + \log(1 - D_1(G(\mathbf{z}^{(i)}))) \tag{5.11}$$

$$\frac{-1}{\mathbf{s}_2}\sum_{i=1}^{\mathbf{s}_2} \log D_2(\tilde{\mathbf{y}}^{(i)}) + \log(1 - D_2(\mathbf{A}G(\tilde{\mathbf{z}}^{(i)}))) \tag{5.12}$$

  Update the generator by GD on the loss:

$$\begin{aligned} &\frac{1}{\mathbf{s}_1}\sum_{i=1}^{\mathbf{s}_1} \log(1 - D_1(G(\mathbf{z}^{(i)}))) \\ +&\frac{1}{\mathbf{s}_2}\sum_{i=1}^{\mathbf{s}_2} \log(1 - D_2(\mathbf{A}G(\tilde{\mathbf{z}}^{(i)}))) \end{aligned} \tag{5.13}$$

end for
return $\{\hat{\mathbf{z}}^{(1)}, \hat{\mathbf{z}}^{(2)}, \ldots\}, \hat{G}, \hat{D}_1, \hat{D}_2$

is used to distinguish between actual compressed training data $\mathbf{y}$ and generated data $\mathbf{A}G(\mathbf{z})$. The details of the training procedure can be found in Algorithm 5.2. One iteration of this algorithm is also illustrated in Figure 5.2. The addition of a second discriminator in Algorithm 5.2 does not affect the representative power of the generator. In fact, similar arguments as in [15, Proposition 2] can be made to show that, with the two discriminators, the distribution of the generator output being the same as that of the training data remains optimal.

5.3.5 Contrastive loss regularization for supervised learning tasks

The low-dimensional vector $\hat{\mathbf{z}}$, returned by Algorithm 5.1 or 5.2, can be used as an input to $\hat{G}$ to recover the original image $\mathbf{x}$. Since $\hat{G}$ learns to represent the overall data distribution of $\mathbf{x}$, $\hat{\mathbf{z}}$ must hold characteristic information specific to $\mathbf{x}$. This motivates us to use $\hat{\mathbf{z}}$ as an input feature for inference tasks such as classification. Since $\hat{\mathbf{z}}$ is of much lower dimension than $\mathbf{x}$ (and, usually, $\mathbf{y}$), using it as an input feature to a classifier reduces the curse of dimensionality. To drive $\hat{\mathbf{z}}$ to be more discriminative for the classification task, we add a contrastive loss [123] term to (5.6). We assume that labeled training data is available, and the ground-truth label of $\mathbf{x}^{(i)}$ is denoted by ℓ_i. The contrastive loss of a batch of $\mathbf{z}$'s is given by:

$$\mathcal{L}_{\text{contr}} \triangleq \frac{\lambda_{\text{contr}}}{\mathbf{s}(\mathbf{s}-1)} \sum_{i,j=1}^{\mathbf{s}} \big[\mathbb{1}(\ell_i = \ell_j)||\mathbf{z}^{(i)} - \mathbf{z}^{(j)}||_2^2 + \mathbb{1}(\ell_i \neq \ell_j) \max\{0, M - ||\mathbf{z}^{(i)} - \mathbf{z}^{(j)}||_2^2\}\big] \tag{5.14}$$

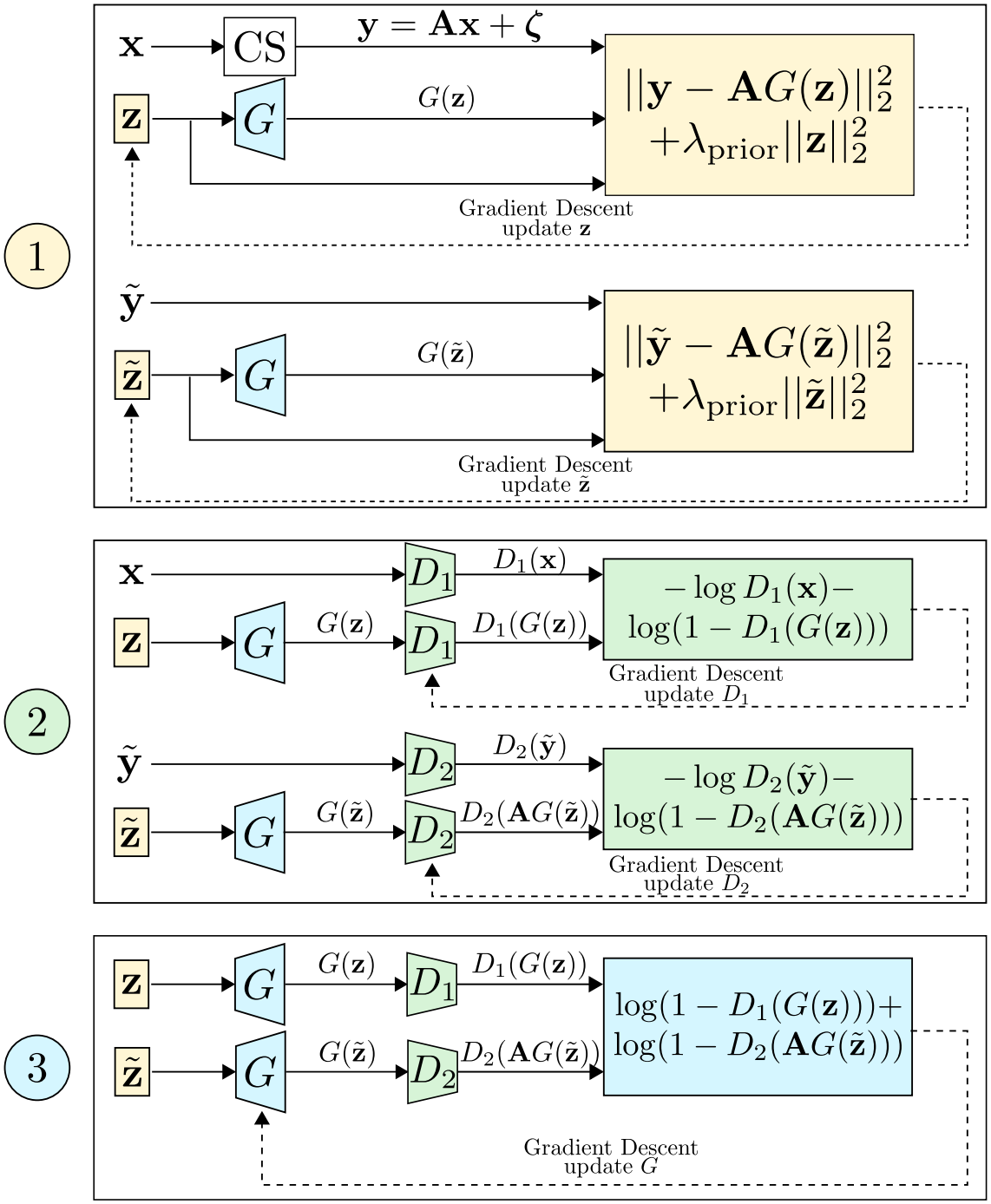

Figure 5.2: One iteration of the GAN training algorithm using compressed training data.

where λ_{contr} is a weight which dictates the relative importance of this loss, and M is a positive margin.

5.4 Experiments

In our experiments, we use three different image datasets: the MNIST handwritten digits dataset [82], the Fashion-MNIST (F-MNIST) clothing articles dataset [124], and the CelebFaces Attributes dataset (CelebA) [125].

The MNIST and F-MNIST datasets each consists of $60,000$ training images and $10,000$ testing images, each of size 28×28. We split the training images into a training set of $50,000$ images and hold-out a validation set containing $10,000$ images. The testing set is kept the same. The images contain a single channel, therefore the input dimension n is $28 \times 28 = 784$.

The CelebA dataset is a large-scale face dataset consisting of more than $200,000$ face images, split into training, validation, and testing sets. The RGB images were cropped to a size of 64×64, resulting in an input dimension of $n = 64 \times 64 \times 3 = 12,288$.

For all datasets, our generative and discriminative models follow the Deep Convolutional GAN (DCGAN) architecture in [104]. We use the Adam optimizer [126] for training the GAN. All hyper-parameters were either set to match the ones in [10] or chosen by testing on the holdout validation set. Our implementation is based on TensorFlow [127] and builds on open-source software [10, 128]. Details of the hyper-parameters used in our experiments can be found in the code repository.

5.4.1 Reconstruction

We first perform a compressed sensing reconstruction task. We train our model using Algorithm 5.1, assuming access to the original non-compressed training set. We refer to our trained model as Compressed Sensing GAN (CSGAN), since the GAN was trained in a task-aware fashion for compressed sensing. As a baseline, we compare our reconstruction results to those obtained by the method in [10], which trains a DCGAN using the usual GAN training framework. At test time, both methods optimize (5.6) to obtain $\hat{\mathbf{z}}$, with the same learning rate and number of GD iterations. For both cases, we perform the same number of random restarts on the initialization of $\mathbf{z}$. The reconstruction is then given by $G(\hat{\mathbf{z}})$.

Additionally, we compare the results to Lasso, performed directly on the pixel values for MNIST and F-MNIST, and in the Discrete Cosine Transform (DCT) and Wavelet Transform domains for CelebA as was done in [10]. We also compare our results to two iterative shrinkage-thresholding algorithms: the Two-step Iterative Shrinkage-Thresholding algorithm (TwIST) [129] and the Fast Iterative Shrinkage-Thresholding algorithm (FISTA) [130] and, in the case of MNIST and F-MNIST, to reconstructions based on the Split Bregman (SB) method with a total variation (TV) regularizer [131], and the SB method with a Besov norm regularizer [132]. The SB method was not performed on the CelebA dataset as the smoothness assumption is not applicable in the case of RGB images when different channels are not treated independently. We report per-pixel mean-squared reconstruction error results in Figure 5.3, as we vary the number of measurements m. It is shown that, and

especially for very low values of m, the task-aware training of CSGAN is able to more reliably reconstruct unseen samples $\mathbf{x}$.

Remark 5.1 *We note that the DCGAN results for MNIST in Figure 5.3 differ from those reported in [10], due to the use of a GAN instead of a VAE. As GANs and VAEs vary in their training methods, for clarity of presentation, we have opted to only use GANs. However, our method can be readily extended to VAE models.*

5.4.2 GAN training on compressed inputs

As previously mentioned, for some applications, it might be prohibitive to acquire a large training set consisting of non-compressed images. However, compressed training data can be readily available. To empirically validate the dual discriminator training method on compressed measurements and non-compressed inputs, we study the effect of varying the size of the non-compressed training set. Naturally, DCGAN can only be trained on the non-compressed training images, and suffers from over-fitting. The results are reported in Tables 5.1 and 5.2, and Figure 5.4, where NC = Number of non-compressed training samples. We can see that the addition of a compressed data discriminator has successfully allowed the training of a CSGAN using compressed measurements. Additionally, we note an interesting trend: when NC = 0, CSGAN performs better than when NC = 100 and $1,000$ (but not when NC = $8,000$). In fact, when the discriminator for non-compressed data D_1 overfits the small amount of training data, this negatively affects the performance of the generator (which is shared by both discriminators D_1 and D_2). In such cases, it

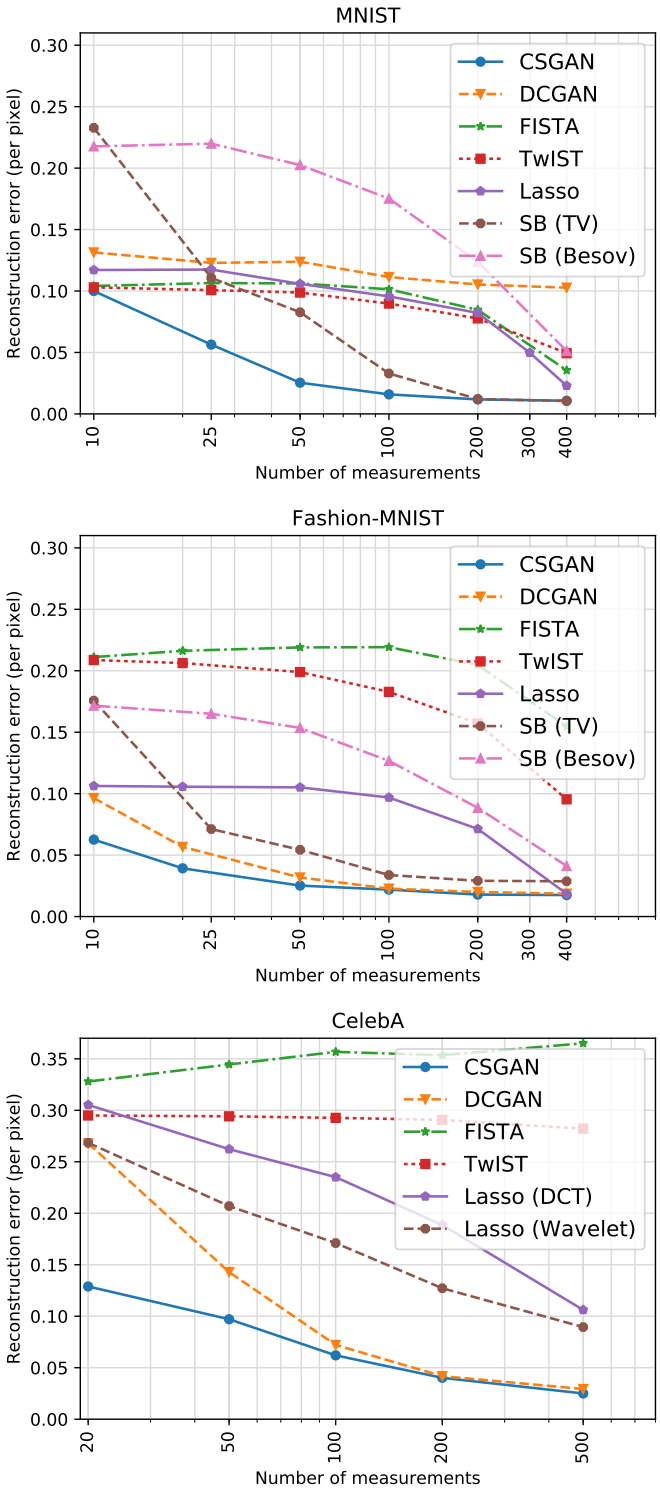

Figure 5.3: MNIST, F-MNIST, and CelebA reconstruction results for various measurements numbers m.

NC	DCGAN	CSGAN
0	-	**0.0299**
100	0.1138	**0.1053**
1,000	0.0859	**0.0322**
8,000	0.0894	**0.0124**

Table 5.1: MNIST: Reconstruction results for $m = 200$ when varying the number of non-compressed training data.

NC	**A** random Gaussian		Super-resolution	
	DCGAN	CSGAN	DCGAN	CSGAN
1,000	0.1278	**0.0514**	0.1006	**0.0510**
4,000	0.0837	**0.0394**	0.0582	**0.0436**
32,000	0.0800	**0.0308**	**0.0241**	0.0247

Table 5.2: CelebA: Reconstruction results for $m = 500$ when varying the number of non-compressed training data.

is beneficial to only use the compressed data discriminator D_2. The smallest number of non-compressed data needed to train D_1 can be determined using the validation set. Generally, we can see that the compressed data discriminator is extremely useful especially when the amount of available non-compressed training data is very low.

In the extreme case, where only compressed measurements are available for training, we show qualitative results of MNIST and F-MNIST reconstruction in Figures 5.4 and 5.6. We would like to emphasize that this CSGAN has never seen any real training image and has been solely trained on compressed measurements, yet can reconstruct reasonably good samples. Additionally, quantitative reconstruction results for various values of the number of measurements m can be found in Table 5.3.

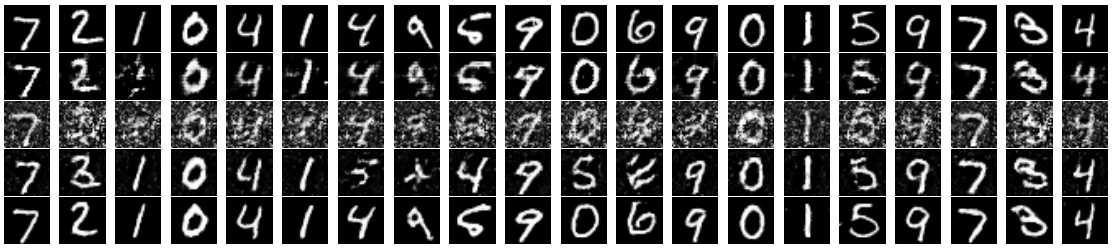

Figure 5.4: MNIST reconstruction results with $m = 200$. Top to bottom rows: original images, reconstructions with NC = 0, reconstructions with NC = 100, reconstructions with NC = 1,000, and reconstructions with NC = 8,000.

Figure 5.5: MNIST reconstruction results when only compressed training data is available. Top row: original image; middle row: reconstructed image from $m = 200$ measurements; bottom row: reconstructed image from $m = 400$ measurements.

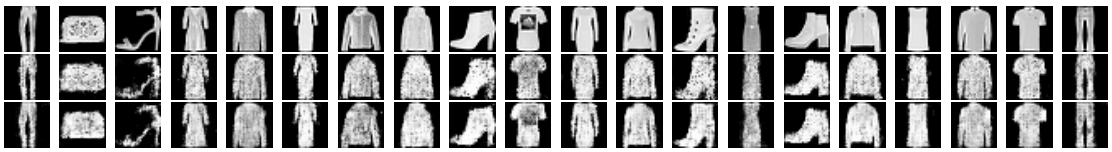

Figure 5.6: F-MNIST reconstruction results when only compressed training data is available. Top row: original image; middle row: reconstructed image from $m = 200$ measurements; bottom row: reconstructed image from $m = 400$ measurements.

m	**MNIST**	**F-MNIST**
20	0.2164	0.2829
50	0.0535	0.0988
100	0.0304	0.0534
200	0.0299	0.0579

Table 5.3: CSGAN reconstruction results when only compressed training data is available (NC = 0) for various measurements numbers m.

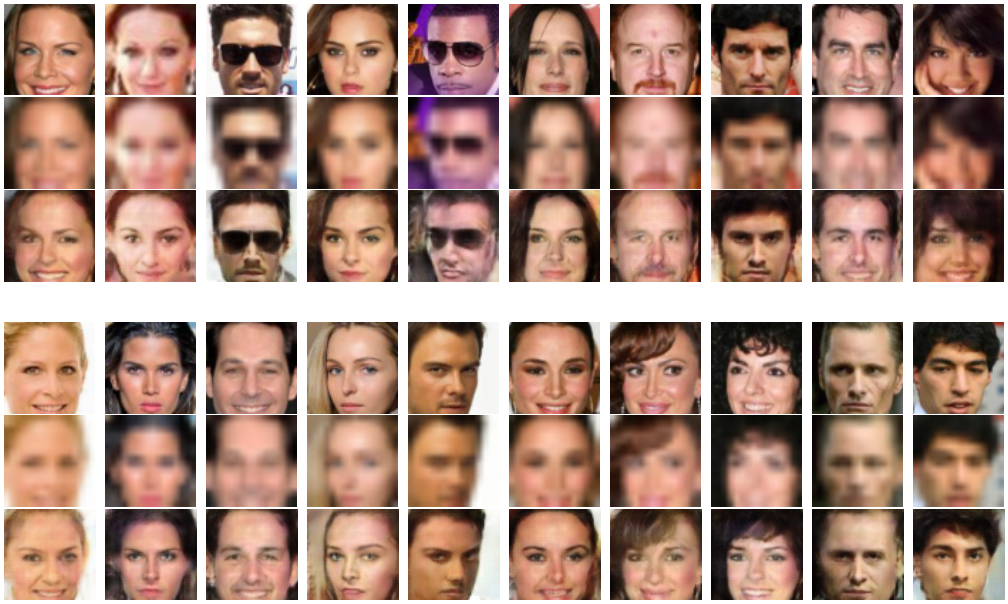

Figure 5.7: CelebA super-resolution results. Top row: original image; middle row: blurred image; bottom row: reconstructed image.

5.4.3 Super-resolution

Super-resolution is the task of increasing the resolution of an image. For this special case, where $\mathbf{A}$ is a matrix that averages neighboring pixels, no theoretical guarantees (such as (5.4)) are known. However, experiments using such averaging matrices $\mathbf{A}$'s still provide good results. Super-resolution is actually a relevant application where an abundance of non-compressed (i.e., high resolution) images may not be available. Results when varying the number of non-compressed training data are reported in Table 5.2 for compressed measurements four times smaller than the original images ($3,072$ measurements). Additionally, qualitative results can be seen in Figure 5.7 on the CelebA dataset. We can see that CSGAN produces realistic reconstructions that resemble the original image.

5.4.4 Classification

In this section, we use the discriminatively-regularized CSGAN with the additional contrastive loss defined in (5.14), with $\lambda_{\text{contr}} = 100$ and $M = 0.1$. We train a CNN classifier based on the LeNet [82] architecture, with a fully-connected layer to map the input latent variables $\hat{\mathbf{z}}$ to a 784-dimensional vector as expected by LeNet. We train the network over 30 epochs and pick the best model based on the holdout validation set. Our results are reported in Tables 5.4 and 5.5. In the case of MNIST, we compare our classification performance to that of SF [114]. We can see that inference can indeed be made even using an extremely small number of measurements. When training this CNN using the $\hat{\mathbf{z}}$'s obtained from DCGAN, we obtain much lower classification accuracies. This clearly demonstrates the effectiveness of the regularization of CSGAN using the contrastive loss.

In order to further investigate the structure of the GAN latent space, we report classification accuracies using a basic 50-nearest-neighbor (50-NN) classifier based on the Euclidean distance in Tables 5.5 and 5.4. This simple 50-NN classifier clearly does not give state-of-the-art classification performance. It, however, serves to show that the latent space has indeed been regularized so that samples belonging to the same class are represented by $\mathbf{z}$'s which are close to each other in the Euclidean distance sense. This is made even clearer when compared to the performance of the same 50-NN classifier on the DCGAN latent space.

Finally, we report per-pixel mean-squared reconstruction error results on the MNIST and F-MNIST datasets when using the contrastive loss regularizer in Table

m	SF	LeNet		50-NN	
		CSGAN + cont. $\hat{\mathbf{z}}$	DCGAN	CSGAN + cont. $\hat{\mathbf{z}}$	DCGAN
8	0.3697	**0.4560**	0.3814	0.3561	**0.3679**
39	0.4679	**0.7572**	0.4304	**0.5987**	0.3951
78	0.5645	**0.8740**	0.4296	**0.6991**	0.3957
196	0.7258	**0.9257**	0.4818	**0.7656**	0.4555

Table 5.4: Classification accuracy on MNIST using Smash Filters (SF), the LeNet CNN classifier, and a 50-NN classifier.

m	LeNet		50-NN	
	CSGAN + cont. $\hat{\mathbf{z}}$	DCGAN	CSGAN + cont. $\hat{\mathbf{z}}$	DCGAN
10	**0.4881**	0.4372	**0.3937**	0.3019
50	**0.7410**	0.6780	**0.6073**	0.4183
100	**0.7705**	0.7363	**0.6377**	0.4495
200	**0.7830**	0.7584	**0.6456**	0.4522

Table 5.5: Classification accuracy on F-MNIST using the LeNet CNN and 50-NN classifiers.

5.6. These results serve to show that the addition of the discriminative regularizer does not hurt reconstruction performance.

5.5 Concluding remarks

In this chapter, we present an effective method for training task-aware generative models, specifically for compressive sensing tasks. We show that this task awareness improves the performance, especially when a very low number of measurements is available. Additionally, we demonstrate that it is also possible to train CSGANs

m	MNIST		F-MNIST	
	CSGAN	CSGAN + cont. $\hat{\mathbf{z}}$	CSGAN	CSGAN + cont. $\hat{\mathbf{z}}$
10	0.1042	**0.0999**	**0.0627**	0.0732
50	0.0353	**0.0334**	**0.0253**	0.0254
100	0.0285	**0.0186**	0.0220	**0.0203**
200	0.0199	**0.0139**	0.0179	**0.0179**
400	0.0169	**0.0112**	0.0175	**0.0168**

Table 5.6: Per-pixel mean-squared reconstruction error results when using the contrastive loss regularizer (with $\mathbf{z}$ dimension $k = 20$).

with only compressed measurements as training data, or, if available, only a small number of non-compressed measurements.

Chapter 6: Conclusion

6.1 Discussion

In this dissertation, we analyzed the structure, performance, and limitations of deep networks. We analyzed the ideal conditions for deep networks to perform well and investigated methods to mitigate the effect of deviating from these ideal conditions.

As previously mentioned, the ideal scenario consists of training a DNN using a very large training set. Such a training set should consist of samples which are balanced across classes, not lossy, and void of label noise.

In Chapter 2, we investigated the relationship between the performance of a CNN, its depth, and the training set size. We showed that, under i.i.d. sampling of the training set, and if the training and testing sampling distributions are the same, good generalization performance is guaranteed with high probability whenever the training set size is some constant times d^4, where d is the convolutional depth of the CNN. We also showed how this result changes when the training and testing distributions are slightly different. Namely, we characterized this difference in terms of the variation divergence between the two distributions and showed that the training set size needed for a guaranteed generalization performance increases

with the variation divergence. The results in this chapter showed that, under i.i.d. sampling, we witness an exponential decrease in the incremental benefit that one new random training example adds to the CNN performance. We also empirically tested our results on the problem of gender classification on three different datasets.

In Chapter 3, we investigated other properties of CNNs by examining the structure of their layers. While the convolutional layers of CNNs were previously shown to be approximated by a series of CSC steps, we considered the addition of spatial pooling operations following the CSC steps to mirror modern CNN architectures. We showed that such an addition does not affect the uniqueness and stability properties of the deep CSC model. We also showed that these spatial pooling layers offer a variety of benefits including the decrease in the dimensionality of the involved vectors and dictionaries, noise suppression, and preventing codes from becoming too sparse. Our analysis served to explain why some of the most successful CNN architectures to date use convolutional filters, ReLU activations, and max-pooling.

In Chapter 4, we proposed a new technique to adaptively select training samples to be presented to a DNN. Such a sampling technique serves to overcome the exponential decrease in the incremental benefit of a data point discovered in Chapter 2. Our selection method exploits the knowledge and current state of the network to iteratively and actively find a new optimal subset of training examples to resume training on. This approach was based on balancing four objectives: class balance, data representativeness, data diversity, and classifier uncertainty. These four objectives were shown to improve the performance of a deep network by driving it to a better local minimum, as well as address the non-ideal cases of class imbalance,

noisy training labels, and lack of enough training samples. Our experiments were performed on a variety of computer vision classification problems spanning four benchmark datasets.

In Chapter 5, we examined another non-ideal scenario, in which samples given to the network at inference (and potentially training) time are lossy. We presented an effective method for training task-aware generative models specifically for the task of reconstructing such lossy samples. We empirically showed that this consistently improves the reconstruction performance compared to state-of-the-art compressed sensing recovery techniques. We also showed that it is possible to train the generative models using compressed samples (when training samples are also lossy) or a combination of compressed and complete examples. We finally showed that the latent space of the generator can be regularized and used as a feature for various supervised learning tasks. We carried out our experiments on three well-known image datasets.

6.2 Directions for future research

In Chapter 2, we studied the relationship between the depth of a CNN, its training set size, and its generalization performance for binary classification problems. It would be interesting to further develop this theory and experiments to study the effects of other CNN parameters, extend to multi-class classification problems, as well as investigate the impact of other important factors such as the underlying distribution and the training algorithm.

In Chapter 3, the role of spatial pooling layers in CNN and CSC models was investigated. In light of the parallelism between these two models, the structure and role of other common CNN layers and functionalities can be analyzed.

In Chapter 4, we introduced an adaptive training data selection algorithm. Our current implementation has not been optimized to make use of all available computational resources (including GPUs). There are known faster methods to solve SDPs and pseudo-inverses, e.g. [133–136], utilizing GPUs. Furthermore, as an exact solution to the SDP optimization problem is not required, approximate solution methods [137] can be used to reduce the complexity. In our current implementation, parameters such as the budget, and weights λ_1, λ_2, and λ_3, are chosen by cross-validation and a search over the parameter space. A more detailed analysis of the effect of these parameters can provide better insight into how to tune them. While our subset selection method was designed for classification tasks, it can also be extended for other supervised learning tasks such as verification.

In Chapter 5, we used GANs to impose structure in compressed sensing problems, replacing the usual sparsity constraint. Our current implementation only uses the Gaussian prior on the latent variable $\mathbf{z}$ as a regularizer in the reconstruction loss. It would be interesting to also add a discriminator loss which would drive the GAN to produce more realistic reconstructions. Additionally, training the task-aware GAN and classifier jointly in an end-to-end manner could yield better classification performance.

Appendix A: Proofs from Chapter 2

A.1 Proof of Lemma 2.1

A parametrized class of functions with parameters $\boldsymbol{\eta} \in \mathbb{R}^t$ that is computable in no more than p operations has a VC dimension which is $\mathcal{O}\left(t^2p^2\right)$ (see [34, Theorems 5, 8] for a list of allowable operations). We have: $t = \sum_{l=1}^{d}(m_1^l + n_1^l m_1^l f_1^l f_2^l) + |\mathbf{W}_f|$, where $|\mathbf{W}_f|$ is the number of trainable weights in the fully connected layers. By counting the number of operations required to compute (2.1)-(2.4), we see that the computational complexity of the l-th convolutional layer of a CNN is at most $\mathcal{O}(m_1^l \cdot (n_2^l - f_1^l + 1) \cdot (n_3^l - f_2^l + 1) \cdot (n_1^l n_2^l n_3^l + m_1^l (g^l)^2 + (p^l)^2)$. Using this result, together with the fact that we have assumed d' to be fixed, we prove the lemma. A more exact expression for the VC dimension bound which does not introduce a constant α can be derived from [138, Theorem 7]. It is omitted here for clarity of presentation. ■

A.2 Proof of Theorem 2.1

The proof can be derived using Lemma 2.1 and [139, Theorem A3.1]. Note that this result is not restricted to the exact architecture given in section 2.2 and any

activation function can be used as long as it can be computed using the operations listed in [34, Theorems 5, 8]. ■

A.3 Proof of Theorem 2.2

The proof is derived from [140, Theorem 3.1], [141, Theorem 1]. ■

A.4 Proof of Theorem 2.3

We define the following event:

$$A = \{ \text{ For every } c \in \mathcal{C}^d, \text{ one of (i) or (ii) holds } \}. \tag{A.1}$$

We will show that the probability that A does not occur is less than δ':

$$\begin{aligned}
\Pr\left[\bar{A}\right] &= \Pr\left[\exists c : e_T(c) > \epsilon', e_S(c) \leq \epsilon', \hat{e}_S(c) \leq (1-\gamma')\epsilon'\right] \\
&\quad + \Pr\left[\exists c : e_T(c) > \epsilon', e_S(c) > \epsilon', \hat{e}_S(c) \leq (1-\gamma')\epsilon'\right] && \text{(A.2)} \\
&\leq \Pr\left[\exists c : e_T(c) > \epsilon', e_S(c) \leq \epsilon', \hat{e}_S(c) \leq (1-\gamma')\epsilon'\right] \\
&\quad + \Pr\left[\exists c : e_S(c) > \epsilon', \hat{e}_S(c) \leq (1-\gamma')\epsilon'\right] && \text{(A.3)} \\
&\leq \Pr\left[\exists c : \epsilon' - \tau < e_S(c) \leq \epsilon', \hat{e}_S(c) \leq (1-\gamma')\epsilon'\right] \\
&\quad + \Pr\left[\exists c : e_S(c) > \epsilon', \hat{e}_S(c) \leq (1-\gamma')\epsilon'\right] && \text{(A.4)} \\
&\leq \Pr\left[\exists c : e_S(c) > \epsilon' - \tau, \hat{e}_S(c) \leq (1-\gamma')\epsilon'\right] \\
&\quad + \Pr\left[\exists c : e_S(c) > \epsilon', \hat{e}_S(c) \leq (1-\gamma')\epsilon'\right] && \text{(A.5)}
\end{aligned}$$

$$\leq \frac{\delta'}{2} + \frac{\delta'}{2} = \delta'. \tag{A.6}$$

where (A.4) follows from the fact that, from [28, Theorem 1], $e_T(c) \leq e_S(c) + \tau$, and (A.6) is an application of Theorem 2.1 with $\delta = \delta'/2$, $\epsilon = \epsilon' - \tau$, and $\gamma = \bar{\gamma}$. ■

Appendix B: Proofs from Chapter 3

B.1 Proof of Theorem 3.1

Assume we have two distinct sets of solutions $\{\mathbf{\Gamma}_i^*, \mathbf{P}_i^*\}_{i=1}^L$ and $\{\hat{\mathbf{\Gamma}}_i, \hat{\mathbf{P}}_i\}_{i=1}^L$. Note that if, for some i, $\mathbf{\Gamma}_i^* = \hat{\mathbf{\Gamma}}_i$, then, $\mathbf{P}_i^* = \hat{\mathbf{P}}_i$. Therefore, there must exist j such that $\mathbf{\Gamma}_j^* \neq \hat{\mathbf{\Gamma}}_j$. Consider the smallest such j. From (3.9), we must have $\mathbf{D}_j\mathbf{\Gamma}_j^* = \mathbf{D}_j\hat{\mathbf{\Gamma}}_j = \mathbf{\Gamma}_{j-1}^* = \hat{\mathbf{\Gamma}}_{j-1}$ (with $\mathbf{\Gamma}_0^* = \hat{\mathbf{\Gamma}}_0 = \mathbf{X}$). Define

$$\sigma_\infty(\mathbf{D}_j) = \arg\min_{\mathbf{\Delta}} \|\mathbf{\Delta}\|_{0,\infty}^{\mathbf{s}} \quad \text{s. t. } \mathbf{\Delta} \neq 0, \mathbf{D}_j\mathbf{\Delta} = 0. \tag{B.1}$$

$\mathbf{D}_j(\mathbf{\Gamma}_j^* - \hat{\mathbf{\Gamma}}_j) = 0$ implies $\|\mathbf{\Gamma}_j^* - \hat{\mathbf{\Gamma}}_j\|_{0,\infty}^{\mathbf{s}} \geq \sigma_\infty$. However,

$$\|\mathbf{\Gamma}_j^* - \hat{\mathbf{\Gamma}}_j\|_{0,\infty}^{\mathbf{s}} \leq \|\mathbf{\Gamma}_j^*\|_{0,\infty}^{\mathbf{s}} + \|\hat{\mathbf{\Gamma}}_j\|_{0,\infty}^{\mathbf{s}} \tag{B.2}$$

$$< \left(1 + \frac{1}{\mu(\mathbf{D}_j)}\right) \tag{B.3}$$

$$\leq \sigma_\infty, \tag{B.4}$$

which is a contradiction. In what precedes, (B.2) follows from the triangle inequality satisfied by the $\ell_{0,\infty}$ pseudo-norm [49, Theorem 15]. (B.3) is by the assumption in (3.10), and (B.4) follows from [49, Theorem 7]. ■

B.2 Proof of Lemma 3.1

Let $\mathcal{B}_k$ be the set of indices over which the max operates for the k-th element of $\mathbf{P}$ (and $\hat{\mathbf{P}}$), as defined in Definition 3.3, i.e., $(\mathbf{P})_k = \max_{j\in\mathcal{B}_k}(\mathbf{X})_j$ and $(\hat{\mathbf{P}})_k = \max_{j\in\mathcal{B}_k}(\hat{\mathbf{X}})_j$.

$$\|\mathbf{P}-\hat{\mathbf{P}}\|_2^2 = \sum_k \left(\max_{j\in\mathcal{B}_k}(\mathbf{X})_j - \max_{j\in\mathcal{B}_k}(\hat{\mathbf{X}})_j\right)^2 \tag{B.5}$$

$$= \sum_{k\in\mathcal{K}_1}\left(\max_{j\in\mathcal{B}_k}(\mathbf{X})_j - \max_{j\in\mathcal{B}_k}(\hat{\mathbf{X}})_j\right)^2 + \sum_{k\in\mathcal{K}_2}\left(\max_{j\in\mathcal{B}_k}(\hat{\mathbf{X}})_j - \max_{j\in\mathcal{B}_k}(\mathbf{X})_j\right)^2 \tag{B.6}$$

$$\leq \sum_{k\in\mathcal{K}_1}\left((\mathbf{X})_{j^\star(k)} - (\hat{\mathbf{X}})_{j^\star(k)}\right)^2 + \sum_{k\in\mathcal{K}_2}\left((\hat{\mathbf{X}})_{\underline{j}(k)} - (\mathbf{X})_{\underline{j}(k)}\right)^2 \tag{B.7}$$

$$\leq \sum_{j=1}^{N}\left((\mathbf{X})_j - (\hat{\mathbf{X}})_j\right)^2 = \|\mathbf{X}-\hat{\mathbf{X}}\|_2^2 \tag{B.8}$$

where in (B.6), $\mathcal{K}_1 \triangleq \{k \mid \max_{j\in\mathcal{B}_k}(\mathbf{X})_j \geq \max_{j\in\mathcal{B}_k}(\hat{\mathbf{X}})_j\}$, and $\mathcal{K}_2 \triangleq \{k \mid \max_{j\in\mathcal{B}_k}(\hat{\mathbf{X}})_j > \max_{j\in\mathcal{B}_k}(\mathbf{X})_j\}$. In addition, in (B.7), $j^\star(k) \triangleq \arg\max_{j\in\mathcal{B}_k}(\mathbf{X})_k$ and $\underline{j}(k) \triangleq \arg\max_{j\in\mathcal{B}_k}(\hat{\mathbf{X}})_j$. The inequality in (B.8) follows from the fact that the sets $\{\mathcal{B}_k\}$ are mutually exclusive since $s \geq b$. ■

B.3 Proof of Lemma 3.2

$$\|\mathbf{P}-\hat{\mathbf{P}}\|_2^2 = \sum_k \left(\frac{\sum_{j\in\mathcal{B}_k}(\mathbf{X})_j - \sum_{j\in\mathcal{B}_k}(\hat{\mathbf{X}})_j}{b}\right)^2 \tag{B.9}$$

$$\leq \frac{1}{b}\sum_{k}\sum_{j\in\mathcal{B}_k}((\mathbf{X})_j - (\hat{\mathbf{X}})_j)^2 \tag{B.10}$$

$$\leq \frac{1}{b}\sum_{i=1}^{N}((\mathbf{X})_i - (\hat{\mathbf{X}})_i)^2 \cdot \left\lceil \frac{b}{s} \right\rceil \tag{B.11}$$

$$= \left(\frac{1}{s} + \frac{\delta}{b}\right)\sum_{i=1}^{N}((\mathbf{X})_i - (\hat{\mathbf{X}})_i)^2 \tag{B.12}$$

$$\leq \sum_{i=1}^{N}((\mathbf{X})_i - (\hat{\mathbf{X}})_i)^2 = \|\mathbf{X} - \hat{\mathbf{X}}\|_2^2 \tag{B.13}$$

where (B.10) follows from the convexity of the function $f(x) = x^2$. (B.11) follows by observing that each index j is included in at most $\lceil b/s \rceil$ of the sets $\{\mathcal{B}_k\}$. In (B.12), $\lceil b/s \rceil = \delta + b/s$, with $\delta < 1$ and $\delta = 0$ if b/s is an integer. (B.13) follows by noting that, if $s = 1$, then $\delta = 0$ and $1/s + \delta/b = 1$ (similarly if $b = 1$, since $\lceil b/s \rceil = 1$). Now assume $b, s \geq 2$. Then, $1/s \leq 1/2$ and $\delta/b < 1/b \leq 1/2$. Then, we always have $1/s + \delta/b \leq 1$. ■

B.4 Proof of Theorem 3.2

We start with $i = 1$. By the assumptions in (3.12) and (3.13), and the feasibility of $\mathbf{\Gamma}_1^*$, $\hat{\mathbf{\Gamma}}_1$, we have

$$\|\mathbf{\Gamma}_1^*\|_{0,\infty}^{\mathbf{s}} < \frac{1}{2}\left(1 + \frac{1}{\mu(\mathbf{D}_1)}\right), \quad \|\mathbf{E}\|_2 = \|\mathbf{Y} - \mathbf{D}_1\mathbf{\Gamma}_1^*\|_2 \leq \epsilon_1,$$
$$\|\hat{\mathbf{\Gamma}}_1\|_{0,\infty}^{\mathbf{s}} < \frac{1}{2}\left(1 + \frac{1}{\mu(\mathbf{D}_1)}\right), \quad \|\mathbf{Y} - \mathbf{D}_1\hat{\mathbf{\Gamma}}_1\|_2 \leq \epsilon_1. \tag{B.14}$$

Then, by [3, Theorem 5]

$$\|\mathbf{\Gamma}_1^* - \hat{\mathbf{\Gamma}}_1\|_2^2 \leq \frac{4\epsilon_1^2}{1-(2\|\mathbf{\Gamma}_1^*\|_{0,\infty}^{\mathbf{s}}-1)\mu(\mathbf{D}_1)} = \epsilon_2^2. \tag{B.15}$$

Let $\mathbf{\Delta} \triangleq \hat{\mathbf{P}}_1 - \mathbf{P}_1^*$. Then, $\|\mathbf{\Delta}\|_2^2 = \|\mathbf{P}_1^* - \hat{\mathbf{P}}_1\|_2^2 \leq \|\mathbf{\Gamma}_1^* - \hat{\mathbf{\Gamma}}_1\|_2^2 \leq \epsilon_2^2$ by Lemmas 3.1 and 3.2. Then, we again have

$$\begin{aligned}
&\|\mathbf{\Gamma}_2^*\|_{0,\infty}^{\mathbf{s}} < \frac{1}{2}\left(1+\frac{1}{\mu(\mathbf{D}_2)}\right), \|\mathbf{\Delta}\|_2 = \|\hat{\mathbf{P}}_1 - \mathbf{D}_2\mathbf{\Gamma}_2^*\|_2 \leq \epsilon_2, \\
&\|\hat{\mathbf{\Gamma}}_2\|_{0,\infty}^{\mathbf{s}} < \frac{1}{2}\left(1+\frac{1}{\mu(\mathbf{D}_2)}\right), \|\hat{\mathbf{P}}_1 - \mathbf{D}_2\hat{\mathbf{\Gamma}}_2\|_2 \leq \epsilon_2.
\end{aligned} \tag{B.16}$$

Using the same theorem and lemmas, we conclude that $\|\mathbf{P}_2^* - \hat{\mathbf{P}}_2\|_2 \leq \|\mathbf{\Gamma}_2^* - \hat{\mathbf{\Gamma}}_2\| \leq \epsilon_3$. We repeat this until $i = L$. ■

B.5 Proof of Theorem 3.3

As shown in [3, Theorem 10], and for all i, $\mathbf{\Gamma}_i$ has the same support as $\hat{\mathbf{\Gamma}}_i$, given by $\hat{\mathbf{\Gamma}}_i = \mathcal{S}_{\beta_i}(\mathbf{D}_i^T\hat{\mathbf{\Gamma}}_{i-1})$ (with the convention $\hat{\mathbf{\Gamma}}_0 = \mathbf{X}$). Then,

$$\mathcal{F}(\mathbf{\Gamma}_i) = \mathcal{F}(\hat{\mathbf{\Gamma}}_i) = \mathcal{F}(\mathcal{S}_{\beta_i}(\mathbf{D}_i^T\hat{\mathbf{\Gamma}}_{i-1})) \tag{B.17}$$

$$= \frac{\|\mathcal{S}_{\beta_i}(\mathbf{D}_i^T\hat{\mathbf{\Gamma}}_{i-1})\|_0}{Nm_i} \tag{B.18}$$

$$\leq \frac{\|\mathbf{D}_i^T\hat{\mathbf{\Gamma}}_{i-1}\|_0}{Nm_i} \tag{B.19}$$

$$\leq \frac{\|\mathbf{D}_i^T\|_0\|\hat{\mathbf{\Gamma}}_{i-1}\|_0}{Nm_i} \tag{B.20}$$

$$= Nm_{i-1}\mathcal{F}(\mathbf{D}_i^T)\mathcal{F}(\hat{\mathbf{\Gamma}}_{i-1}) \tag{B.21}$$

$$\leq \mathcal{F}(\hat{\mathbf{\Gamma}}_{i-1}) = \mathcal{F}(\mathbf{\Gamma}_{i-1}) \tag{B.22}$$

where (B.19) follows from the fact that thresholding does not increase the ℓ_0 pseudo-norm. In (B.20), we define the ℓ_0 pseudo-norm of a matrix as the maximum number of non-zero elements in a column. Then, the multiplication $\mathbf{D}_i^T \hat{\mathbf{\Gamma}}_{i-1}$ is a linear combination of $\|\hat{\mathbf{\Gamma}}_{i-1}\|_0$ columns of $\mathbf{D}_i^T$, each having no more than $\|\mathbf{D}_i^T\|_0$ non-zero elements, which leads to (B.20). (B.22) follows form the assumption of the theorem.

■

Appendix C: Proofs from Chapter 4

C.1 Proof of Theorem 4.1

First, we ignore the $M_k^t \leq |\mathcal{X}_k|$ constraints, in (4.3), and let $h_k = \alpha c_k^t / M^t$. Then the problem in (4.3) becomes

$$\begin{aligned} \max_{M_k^t \in \mathbb{Z}^+} \quad & \sum_{k=1}^{L} \log\left(1 + h_k M_k^t\right) \\ \text{s. t.} \quad & \sum_{k=1}^{L} M_k^t \leq M^t, \end{aligned} \tag{C.1}$$

We start by showing a necessary optimality condition.

Lemma C.1 *An optimal profile* $\{M_1^{t\star}, M_2^{t\star}, \ldots, M_L^{t\star}\}$*, must satisfy*

$$\begin{aligned} &\left| \left(\frac{1}{h_i} + M_i^{t\star}\right) - \left(\frac{1}{h_j} + M_j^{t\star}\right) \right| \leq 1, \\ &\textit{for all } i, j \textit{ such that } M_i^t > 0, M_j^t > 0. \end{aligned} \tag{C.2}$$

Proof: We prove the lemma by contradiction. Assume there exist i, j such that

$$\left|\left(\frac{1}{h_i} + M_i^{t\star}\right) - \left(\frac{1}{h_j} + M_j^{t\star}\right)\right| = \Delta > 1. \tag{C.3}$$

Without loss of generality, assume that $\frac{1}{h_i} + M_i^{t\star} > \frac{1}{h_j} + M_j^{t\star}$. Now consider a new profile where $\overline{M_i^t} = M_i^{t\star} - 1$, $\overline{M_j^t} = M_j^{t\star} + 1$, and $\overline{M_k^t} = M_k^{t\star}$ for $k \neq i, j$. This new policy is clearly feasible. We consider the difference between the objective values achieved by $\{\overline{M_k^t}\}$ and $\{M_k^{t\star}\}$:

$$\begin{aligned}
&\sum_{k=1}^{L} \log\left(1 + h_k \overline{M_k^t}\right) - \sum_{k=1}^{L} \log\left(1 + h_k M_k^{t\star}\right) \\
&= \log\left(1 + h_i \overline{M_i^t}\right) + \log\left(1 + h_j \overline{M_j^t}\right) - \log\left(1 + h_i M_i^{t\star}\right) - \log\left(1 + h_j M_j^{t\star}\right) && \text{(C.4)} \\
&= \log \frac{\left(1 + h_i(M_i^{t\star} - 1)\right)\left(1 + h_j(M_j^{t\star} + 1)\right)}{\left(1 + h_i M_i^{t\star}\right)\left(1 + h_j M_j^{t\star}\right)} && \text{(C.5)} \\
&= \log\left(1 + \frac{h_j(1 + h_i M_i^{t\star}) - h_i(1 + h_j M_j^{t\star}) - h_i h_j}{\left(1 + h_i M_i^{t\star}\right)\left(1 + h_j M_j^{t\star}\right)}\right) && \text{(C.6)} \\
&= \log\left(1 + \frac{h_i h_j(M_i^{t\star} - M_j^{t\star} - 1) + h_j - h_i}{\left(1 + h_i M_i^{t\star}\right)\left(1 + h_j M_j^{t\star}\right)}\right) && \text{(C.7)} \\
&= \log\left(1 + \frac{h_i h_j(\Delta - 1)}{\left(1 + h_i M_i^{t\star}\right)\left(1 + h_j M_j^{t\star}\right)}\right) && \text{(C.8)} \\
&> 0, && \text{(C.9)}
\end{aligned}$$

where (C.8) follows from the assumption in (C.3) and (C.9) from $\Delta > 1$. This shows that the profile $\{\overline{M_k^t}\}$ achieves a higher objective value than the profile $\{M_k^{t\star}\}$ which contradicts its optimality. ■

Now, we show that Algorithm 4.1 solves the problem in (C.1). We proceed by

induction on the budget M^t:

- Base case, $M^t = 1$: It is trivial to show that in the optimal profile, one water unit will be assigned to the class with the lowest base level $\frac{M^t}{\alpha c_k^t}$, with ties broken arbitrarily.

- Induction step: Given an optimal profile $\{M_k^t(m)\}$ that solves (C.1) for $M^t = m$, we find the optimal profile $\{M_k^t(m+1)\}$ for $M^t = m+1$. It can be seen that one water unit should be added to $\{M_k^t(m)\}$ because any other deviation from this profile will violate Lemma C.1. Furthermore, using similar arguments as in Lemma C.1, we show that this water unit should be placed at the class with the smallest $\frac{M^t}{\alpha c_k^t} + M_k^t(m)$, with ties broken by the ordering of $\frac{M^t}{\alpha c_k^t}$.

This corresponds exactly to the operation of Algorithm 4.1 in the case of no caps. When caps are added, i.e., with the constraints $M_k^t \leq |\mathcal{X}_k|$, it is easy to show the following:

- If the optimal profile for Problem (C.1) is feasible for Problem (4.3), then it is also optimal for Problem (4.3).

- If the optimal profile for Problem (C.1) contains a class i such that $M_i^t > |\mathcal{X}_i|$ (infeasible for Problem (4.3)), then the solution of Problem (4.3) must have $M_i^t = |\mathcal{X}_i|$. Furthermore, since no more water units can be allocated to this class, we can solve Problem (4.3) with class i removed and total budget decreased by $|\mathcal{X}_i|$. This is equivalent to skipping class i when its water level reaches its cap.

As this describes the operation of Algorithm 4.1 with caps, this proves the theorem.

■

Remark C.1 *The use of the ceiling operation $\lceil \frac{M^t}{\alpha c_k^t} \rceil$ in Algorithm 4.1 is not necessary for the optimality of the solution. However, it makes the procedure easier to visualize and results in the same optimal profile.*

C.2 Proof of Lemma 4.1

We start with the objective function in (4.8).

$$
\begin{aligned}
& -\frac{\lambda_d}{M}\mathbf{s}^\intercal\tilde{\mathbf{D}}\mathbf{s} + \frac{\lambda_r}{N-M}(\mathbf{1}-\mathbf{s})^\intercal\tilde{\mathbf{D}}\mathbf{s} - \lambda_u\tilde{\mathbf{c}}^\intercal\mathbf{s} && \text{(C.10)}\\
=& -\frac{\lambda_d}{4M}(\mathbf{x}+\mathbf{1})^\intercal\tilde{\mathbf{D}}(\mathbf{x}+\mathbf{1}) + \frac{\lambda_r}{4(N-M)}(\mathbf{1}-\mathbf{x})^\intercal\tilde{\mathbf{D}}(\mathbf{x}+\mathbf{1}) - \frac{\lambda_u}{2}\tilde{\mathbf{c}}^\intercal(\mathbf{x}+\mathbf{1}) && \text{(C.11)}\\
=& -\frac{\lambda_d}{4M}\mathbf{x}^\intercal\tilde{\mathbf{D}}\mathbf{x} - \frac{\lambda_d}{2M}\mathbf{1}^\intercal\tilde{\mathbf{D}}\mathbf{x} - \frac{\lambda_d}{4M}\mathbf{1}^\intercal\tilde{\mathbf{D}}\mathbf{1} - \frac{\lambda_r}{4(N-M)}\mathbf{x}^\intercal\tilde{\mathbf{D}}\mathbf{x} + \frac{\lambda_r}{4(N-M)}\mathbf{1}^\intercal\tilde{\mathbf{D}}\mathbf{1}\\
& -\frac{\lambda_u}{2}\tilde{\mathbf{c}}^\intercal\mathbf{x} - \frac{\lambda_u}{2}\tilde{\mathbf{c}}^\intercal\mathbf{1} && \text{(C.12)}\\
=& \mathbf{x}^\intercal\underbrace{\left(-\frac{\lambda_d}{4M} - \frac{\lambda_r}{4(N-M)}\right)\tilde{\mathbf{D}}}_{\triangleq\mathbf{A}}\mathbf{x} + \underbrace{\left(-\frac{\lambda_d}{2M}\mathbf{1}^\intercal\tilde{\mathbf{D}} - \frac{\lambda_u}{2}\tilde{\mathbf{c}}^\intercal\right)}_{\triangleq\mathbf{b}^\intercal}\mathbf{x}\\
& \underbrace{-\frac{\lambda_d}{4M}\mathbf{1}^\intercal\tilde{\mathbf{D}}\mathbf{1} + \frac{\lambda_r}{4(N-M)}\mathbf{1}^\intercal\tilde{\mathbf{D}}\mathbf{1} - \frac{\lambda_u}{2}\tilde{\mathbf{c}}^\intercal\mathbf{1}}_{\text{constant w.r.t. }\mathbf{x}} && \text{(C.13)}
\end{aligned}
$$

where (C.11) follows from the change of variable $\mathbf{x} = 2\mathbf{s} - \mathbf{1}$. Furthermore, we can drop the constant term in (C.13) since we are only interested in the minimizer $\mathbf{x}$, and not the objective value.

Similarly, the constraint in (4.8) can be written as:

$$\left\{\mathbf{1}^\intercal \mathbf{s} = M\right\} \equiv \left\{(\mathbf{1}^\intercal \mathbf{s} - M)^2 = 0\right\} \tag{C.14}$$

$$\equiv \left\{(\mathbf{1}^\intercal \frac{\mathbf{x}+\mathbf{1}}{2} - M)^2 = 0\right\} \tag{C.15}$$

$$\equiv \left\{(\mathbf{1}^\intercal \mathbf{x} + \mathbf{1}^\intercal \mathbf{1} - 2M)^2 = 0\right\} \tag{C.16}$$

$$\equiv \left\{(\mathbf{1}^\intercal \mathbf{x} + N - 2M)^2 = 0\right\} \tag{C.17}$$

This concludes the proof. ■

C.3 Proof of Theorem 4.2

We denote the optimal objective value of (4.11) by $p^\star$. We start by writing the Lagrangian dual. This problem has zero duality gap [73, Theorem 9], therefore:

$$p^\star = \max_{\mu\in\mathbb{R}} \min_{\mathbf{x}\in\{-1,1\}^N} \left\{\mathbf{x}^\intercal \mathbf{A}\mathbf{x} + \mathbf{b}^\intercal \mathbf{x} + \mu(\mathbf{1}^\intercal \mathbf{x} - 2M + N)^\intercal(\mathbf{1}^\intercal \mathbf{x} - 2M + N)\right\} \tag{C.18}$$

$$= \max_{\mu\in\mathbb{R}} \min_{\mathbf{x}\in\{-1,1\}^N} \Big\{\mathbf{x}^\intercal \mathbf{A}\mathbf{x} + \mathbf{b}^\intercal \mathbf{x} + \mu\mathbf{x}^\intercal \mathbf{1}\mathbf{1}^\intercal \mathbf{x} - 2\mu(2M - N)\mathbf{1}^\intercal \mathbf{x} + \mu(2M - N)^2\Big\} \tag{C.19}$$

$$= \max_{\mu\in\mathbb{R}} \Bigg[(2M - N)^2\mu + \min_{\mathbf{x}\in\{-1,1\}^N} \Big\{\mathbf{x}^\intercal \left(\mathbf{A} + \mu\mathbf{1}\mathbf{1}^\intercal\right)\mathbf{x} + (\mathbf{b} - 2\mu(2M - N)\mathbf{1})^\intercal \mathbf{x}\Big\}\Bigg]. \tag{C.20}$$

Next, we relax the constraint $\mathbf{x} \in \{-1, 1\}^N$ to the non-binary constraint $\mathbf{x}^\intercal \mathbf{x} = N$. Commonly used relaxations for $\mathbf{x} \in \{-1, 1\}^N$ include $\mathbf{x}^\intercal \mathbf{x} = N$ and $x_i \in [-1, 1]$, resulting in the **same** duality gap [73, Theorem 2]. We use the former as it results

another well-known problem with a zero-duality gap: the trust-region problem. Thus, we have:

$$p^\star \geq \max_{\mu\in\mathbb{R}} \Bigg[(2M-N)^2\mu + \min_{\mathbf{x}^\intercal\mathbf{x}=N} \Big\{ \mathbf{x}^\intercal \left(\mathbf{A}+\mu\mathbf{1}\mathbf{1}^\intercal\right)\mathbf{x} + (\mathbf{b}-2\mu(2M-N)\mathbf{1})^\intercal\mathbf{x} \Big\} \Bigg] \tag{C.21}$$

The inner minimization over $\mathbf{x}$ is known as the trust-region problem and has no duality gap [77, p.229], therefore:

$$\begin{aligned}
(\text{C.21}) &= \max_{\mu\in\mathbb{R}} \Bigg[(2M-N)^2\mu \\
&\qquad + \max_{\gamma\in\mathbb{R}} \Bigg\{ \min_{\mathbf{x}} \Big\{ \mathbf{x}^\intercal \left(\mathbf{A}+\mu\mathbf{1}\mathbf{1}^\intercal\right)\mathbf{x} + (\mathbf{b}-2\mu(2M-N)\mathbf{1})^\intercal\mathbf{x} \\
&\qquad\qquad + \gamma(\mathbf{x}^\intercal\mathbf{x}-N) \Big\} \Bigg\} \Bigg] \qquad (\text{C.22}) \\
&= \max_{\mu\in\mathbb{R}} \Bigg[(2M-N)^2\mu \\
&\qquad + \max_{\gamma\in\mathbb{R}} \Bigg\{ -\gamma N + \min_{\mathbf{x}} \Big\{ \mathbf{x}^\intercal \left(\mathbf{A}+\mu\mathbf{1}\mathbf{1}^\intercal+\gamma\mathbf{I}\right)\mathbf{x} \\
&\qquad\qquad + (\mathbf{b}-2\mu(2M-N)\mathbf{1})^\intercal\mathbf{x} \Big\} \Bigg\} \Bigg] \qquad (\text{C.23}) \\
&= \max_{\mu\in\mathbb{R},\gamma\in\mathbb{R}} \Bigg[(2M-N)^2\mu - \gamma N \\
&\qquad + \min_{\mathbf{x}} \Big\{ \mathbf{x}^\intercal \left(\mathbf{A}+\mu\mathbf{1}\mathbf{1}^\intercal+\gamma\mathbf{I}\right)\mathbf{x} \\
&\qquad\qquad + (\mathbf{b}-2\mu(2M-N)\mathbf{1})^\intercal\mathbf{x} \Big\} \Bigg] \qquad (\text{C.24})
\end{aligned}$$

We change the inner minimization over $\mathbf{x}$ to an equivalent form and rewrite (C.24) as:

$$\max_{\mu\in\mathbb{R},\gamma\in\mathbb{R}} (2M-N)^2\mu - \gamma N + \max_{\tau} -\tau \tag{C.25}$$

$$\text{s.t.} \quad \text{for all } \mathbf{x}:$$

$$\mathbf{x}^\intercal \left(\mathbf{A} + \mu\mathbf{1}\mathbf{1}^\intercal + \gamma\mathbf{I}\right)\mathbf{x} + (\mathbf{b} - 2\mu(2M-N)\mathbf{1})^\intercal\mathbf{x} \geq -\tau \tag{C.26}$$

The constraint in (C.26) is equivalent to (from [142] and [143, Section 3.4]):

$$\begin{pmatrix} \tau & \frac{1}{2}(\mathbf{b} - 2\mu(2M-N)\mathbf{1})^\intercal \\ \frac{1}{2}(\mathbf{b} - 2\mu(2M-N)\mathbf{1}) & \mathbf{A} + \mu\mathbf{1}\mathbf{1}^\intercal + \gamma\mathbf{I} \end{pmatrix} \succeq 0. \tag{C.27}$$

This concludes the proof. ■

Remark C.2 *We note that in the proof of Theorem 4.2, only one relaxation, (C.21), was made. Other relaxations are possible such as introducing $x_i^2 = 1\ \forall i$ constraints and taking the Lagrangian dual as done in [74]. However, this results in $(N-1)$ additional optimization variables and makes the resulting SDP more complex. We do not adopt such an approach as we would like the subset selection problem to run as efficiently as possible.*

Remark C.3 *The inner minimization over $\mathbf{x}$ in (C.24) is unconstrained and can*

be solved exactly, with a minimum value of

$$-\frac{1}{4}(\mathbf{b}-2\mu(2M-N)\mathbf{1})^{\intercal}\cdot(\mathbf{A}+\mu\mathbf{1}\mathbf{1}^{\intercal}+\gamma\mathbf{I})^{\dagger}\cdot(\mathbf{b}-2\mu(2M-N)\mathbf{1}), \qquad \text{(C.28)}$$

if $(\mathbf{A}+\mu\mathbf{1}\mathbf{1}^{\intercal}+\gamma\mathbf{I})\succeq 0$. *However, plugging the expression in (C.28) into (C.24) and maximizing over* μ,γ *does not result in a convex formulation and cannot be solved efficiently.*

Appendix D: Proofs from Chapter 5

D.1 Proof of Theorem 5.1

Let $g_t(\mathbf{x})$ be the probability distribution function (p.d.f.) of $G_t(\mathbf{z})$ and $f(\mathbf{x})$ be the p.d.f. of $\mathbf{x}$. Then, from [15, Proposition 2], $g_t(\mathbf{x})$ converges to $f(\mathbf{x})$ pointwise in $\mathbf{x}$. By assumption, $f(\mathbf{x})$ has bounded support $\mathcal{X}$, i.e., $\mu(\mathcal{X})$ is finite, where $\mu(\cdot)$ is the Lebesgue measure. We note that the assumption of $\mathcal{X}$ having bounded support is reasonable, especially for computer vision tasks where pixel values are usually bounded (for instance, in $[0, 255]$).

Then, by Egorov's theorem, for all $\epsilon > 0$, there exists a set $B \subseteq \mathcal{X}$ such that $\mu(B) < \epsilon$ and $g_t(\mathbf{x})$ converges to $f(\mathbf{x})$ uniformly on $\mathcal{X} \setminus B$. This implies that, for all $\mathbf{x} \in \mathcal{X} \setminus B$ and for all ν, there exists t_0 such that $|g_t(\mathbf{x}) - f(\mathbf{x})| < \nu$, for all $t \geq t_0$. This means that, for $\mathbf{x} \in \mathcal{X} \setminus B$, $g_t(\mathbf{x}) = 0$ implies $f(\mathbf{x}) < \nu$. Additionally, $g_t(\mathbf{x}) > 0$ implies that there exists $\mathbf{z}$ such that $G_t(\mathbf{z}) = \mathbf{x}$, i.e., $\min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 = 0$.

Let $\mathcal{X}_\nu = \{\mathbf{x} \in \mathcal{X} \mid f(\mathbf{x}) \leq \nu\}$. Note that $\{\mathbf{x} \in \mathcal{X} \setminus B \mid g_t(\mathbf{x}) = 0\} \subseteq \mathcal{X}_\nu$ for all $t \geq t_0$. Then, for all $\epsilon, \nu > 0$ and $t \geq t_0$,

$$\mathbb{E}_{\mathbf{x}}\left[\min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2\right] \tag{D.1}$$

$$\leq \int_B \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \; f(\mathbf{x}) \; d\mathbf{x} + \int_{\mathcal{X}_\nu} \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \; f(\mathbf{x}) \; d\mathbf{x}$$

$$+ \int_{\mathcal{X}\setminus(B\cup\mathcal{X}_\nu)} \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \; f(\mathbf{x}) \; d\mathbf{x} \tag{D.2}$$

$$\leq \int_B \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \; d\mathbf{x} + \nu \int_{\mathcal{X}_\nu} \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \; d\mathbf{x} \tag{D.3}$$

$$\leq \mu(B) \sup_{\mathbf{x}\in B} \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 + \nu\mu(\mathcal{X}_\nu) \sup_{\mathbf{x}\in\mathcal{X}_\nu} \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \tag{D.4}$$

$$\leq (\mu(B) + \nu\mu(\mathcal{X}_\nu)) \sup_{\mathbf{x}\in\mathcal{X}} \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \tag{D.5}$$

$$= (\mu(B) + \nu\mu(\mathcal{X}_\nu)) \max_{\mathbf{x}\in\mathcal{X}} \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \tag{D.6}$$

$$\leq C(\epsilon + \nu\mu(\mathcal{X})) \tag{D.7}$$

where $C > 0$ is a positive constant.

Equation (D.3) follows from the fact that $\min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 = 0$ for $\mathbf{x} \in \mathcal{X} \setminus (B \cup \mathcal{X}_\nu)$, $f(\mathbf{x}) \leq 1$ for $\mathbf{x} \in \mathcal{X}$, and $f(\mathbf{x}) \leq \nu$ for $\mathbf{x} \in \mathcal{X}_\nu$. Equation (D.6) is obtained using the extreme value theorem since $\mathcal{X}$ is compact. To prove (D.7) we proceed as follows:

$$\max_{\mathbf{x}\in\mathcal{X}} \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \leq \min_{\mathbf{z}} \max_{\mathbf{x}\in\mathcal{X}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \tag{D.8}$$

$$\leq \max_{\mathbf{x}\in\mathcal{X}} ||\mathbf{x} - G_t(\bar{\mathbf{z}})||_2 \tag{D.9}$$

$$= C \tag{D.10}$$

Equation (D.8) follows from the max-min inequality. In (D.9), $\bar{\mathbf{z}}$ is such that $G_t(\bar{\mathbf{z}}) \in \mathcal{X}$. Such a $\bar{\mathbf{z}}$ always exists for $t \geq t_0$. Equation (D.10) follows from the fact

that $\mathcal{X}$ is compact.

Therefore, we obtain (D.7) by noting that $\sup_{t \geq t_0} \max_{\mathbf{x} \in \mathcal{X}} \min_{\mathbf{z}} ||\mathbf{x} - G_t(\mathbf{z})||_2 \leq C$. Since $\mu(\mathcal{X})$ is a finite positive constant and (D.7) is satisfied for any $\epsilon, \nu > 0$, this proves the theorem. ■

Bibliography

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Conference on Neural Information Processing Systems (NIPS)*, 2012.

[3] V. Papyan, Y. Romano, and M. Elad. Convolutional neural networks analyzed via convolutional sparse coding. *arXiv:1607.08194*, 2016.

[4] Y. L. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *International Conference on Machine Learning (ICML)*, 2010.

[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[6] Y. T. Zhou and R. Chellappa. Computation of optical flow using a neural network. In *IEEE International Conference on Neural Networks*, 1988.

[7] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *IEEE International Conference on Computer Vision (ICCV)*, 2009.

[8] M. Lustig, D. Donoho, and J. M. Pauly. Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magnetic Resonance in Medicine*, 58(6):1182–1195, 2007.

[9] M. F. Duarte, M. A. Davenport, D. Takbar, J. N. Laska, T. Sun, K. F. Kelly, and R. G. Baraniuk. Single-pixel imaging via compressive sampling. *IEEE Signal Processing Magazine*, 25(2):83–91, 2008.

[10] A. Bora, A. Jalal, E. Price, and A. G. Dimakis. Compressed sensing using generative models. In *International Conference on Machine Learning (ICML)*, 2017.

[11] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *International Conference on Learning Representations (ICLR), Workshop Track*, 2014.

[12] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations (ICLR)*, 2015.

[13] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *IEEE Symposium on Security and Privacy*, 2016.

[14] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks. *International Conference on Learning Representations (ICLR)*, 2017.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Conference on Neural Information Processing Systems (NIPS)*, 2014.

[16] P. Samangouei, M. Kabkab, and R. Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations (ICLR)*, 2018.

[17] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *International Conference on Learning Representations (ICLR), Workshop Track*, 2014.

[18] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*. 2014.

[19] D. Eigen, J. Rolfe, R. Fergus, and Y. LeCun. Understanding deep architectures using a recursive convolutional network. In *International Conference on Learning Representations (ICLR), Workshop Track*, 2014.

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural computation*, 1(1):151–160, 1989.

[22] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *Conference on Neural Information Processing Systems (NIPS)*, 2014.

[23] P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, 1998.

[24] M. Bianchini and F. Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE Transactions on Neural Networks and Learning Systems*, 25(8):1553–1565, 2014.

[25] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14(3):326–334, June 1965.

[26] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

[27] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer Science & Business Media, 2000.

[28] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175, 2010.

[29] A. C. Gallagher and T. Chen. Understanding images of groups of people. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[30] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. *Technical Report 07-49, University of Massachusetts, Amherst*, 1(2), 2007.

[31] N. Kumar, P. Belhumeur, and S. Nayar. Facetracer: A search engine for large collections of images with faces. In *European Conference on Computer Vision (ECCV)*. Springer, 2008.

[32] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM International Conference on Multimedia*, 2014.

[33] P. Dago-Casas, D. González-Jiménez, L. L. Yu, and J. L. Alba-Castro. Single- and cross-database benchmarks for gender classification under unconstrained settings. In *IEEE International Conference on Computer Vision (ICCV), Workshop Track*, 2011.

[34] P. L. Bartlett and W. Maass. Vapnik-Chervonenkis dimension of neural nets. *The handbook of brain theory and neural networks*, pages 1188–1192, 2003.

[35] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[36] Q. Wang, S. R. Kulkarni, and S. Verdú. Divergence estimation for multidimensional densities via-nearest-neighbor distances. *IEEE Transactions on Information Theory*, 55(5):2392–2405, 2009.

[37] D. L. Donoho and M. Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.

[38] J. A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.

[39] M. Elad. *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. Springer, 2010.

[40] J. Wright, A. Y. Yang, and A. Ganesh. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):210–227, 2009.

[41] D. L. Donoho. Stable recovery of sparse overcomplete representations in the presence of noise. *IEEE Transactions on Information Theory*, 52(1):6–18, 2006.

[42] Y. Chandra Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Asilomar Conference on Signals, Systems and Computers*, 1993.

[43] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Review*, 43(1):129–159, 2001.

[44] M. Elad, M. A. T. Figueiredo, and Y. Ma. On the role of sparse and redundant representations in image processing. *Proceedings of the IEEE*, 98(6):972–982, 2010.

[45] W. Dong, X. Li, L. Zhang, and G. Shi. Sparsity-based image denoising via dictionary learning and structural clustering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[46] R. Grosse, R. Raina, H. Kwong, and A. Y. Ng. Shift-invariant sparse coding for audio classification. In *Uncertainty in Artificial Intelligence*, 2007.

[47] H. Bristow, A. Eriksson, and S. Lucey. Fast convolutional sparse coding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[48] F. Heide, W. Heidrich, and G. Wetzstein. Fast and flexible convolutional sparse coding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[49] V. Papyan, J. Sulam, and M. Elad. Working locally thinking globally-part I: Theoretical guarantees for convolutional sparse coding. *arXiv:1607.02005*, 2016.

[50] V. Papyan, J. Sulam, and M. Elad. Working locally thinking globally-part II: Stability and algorithms for convolutional sparse coding. *arXiv:1607.02009*, 2016.

[51] R. Giryes, G. Sapiro, and A. M. Bronstein. Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 64(13):3444–3457, 2015.

[52] S. Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A*, 374(2065), 2016.

[53] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[54] K. Kavukcuoglu, P. Sermanet, Y. L. Boureau, K. Gregor, M. Mathieu, and Y. LeCun. Learning convolutional feature hierarchies for visual recognition. In *Conference on Neural Information Processing Systems (NIPS)*, 2010.

[55] Y. Gwon, M. Cha, and H. T. Kung. Deep sparse-coded network (DSN). In *International Conference on Pattern Recognition (ICPR)*, 2016.

[56] J. Dean, G. Corrado, and R. Monga. Large scale distributed deep networks. In *Conference on Neural Information Processing Systems (NIPS)*, 2012.

[57] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[58] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

[59] P. W. Munro. Repeat until bored: A pattern selection strategy. In *Conference on Neural Information Processing Systems (NIPS)*, 1992.

[60] M. Plutowski and H. White. Selecting concise training sets from clean data. *IEEE Transactions on Neural Networks*, 4(2):305–318, 1993.

[61] A. P. Engelbrecht. Sensitivity analysis for selective learning by feedforward neural networks. *Fundamenta Informaticae*, 46(3):219–252, 2001.

[62] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

[63] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *International Conference on Machine Learning (ICML)*, 2000.

[64] N. Roy and A. McCallum. Toward optimal active learning through Monte Carlo estimation of error reduction. In *International Conference on Machine Learning (ICML)*, 2001.

[65] K. Yu, J. Bi, and V. Tresp. Active learning via transductive experimental design. In *International Conference on Machine Learning (ICML)*, 2006.

[66] Y. Guo and D. Schuurmans. Discriminative batch mode active learning. In *Conference on Neural Information Processing Systems (NIPS)*, 2008.

[67] L. Zhang, C. Chen, J. Bu, D. Cai, X. He, and T. S. Huang. Active learning based on locally linear reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(10):2026–2038, 2011.

[68] S. Chakraborty, V. Balasubramanian, Q. Sun, S. Panchanathan, and J. Ye. Active batch selection via convex relaxations with guaranteed solution bounds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(10):1945–1958, 2015.

[69] E. Elhamifar, G. Sapiro, A. Yang, and S. S. Sasrty. A convex optimization framework for active learning. In *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[70] J. A. Tropp. Just relax: Convex programming methods for subset selection and sparse approximation. *ICES Report*, 404, 2004.

[71] E. Elhamifar, G. Sapiro, and R. Vidal. See all by looking at a few: Sparse modeling for finding representative objects. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[72] L. Ott, L. Pang, F. T. Ramos, and S. Chawla. On integrated clustering and outlier detection. In *Conference on Neural Information Processing Systems (NIPS)*, 2014.

[73] S. Poljak, F. Rendl, and H. Wolkowicz. A recipe for semidefinite relaxation for (0, 1)-quadratic programming. *Journal of Global Optimization*, 7(1):51–73, 1995.

[74] X. Zheng, X. Sun, D. Li, and Y. Xia. Duality gap estimation of linear equality constrained binary quadratic programming. *Mathematics of Operations Research*, 35(4):864–880, 2010.

[75] S. Reed, H. Lee, D. Anguelov, C. Szegedy, D. Erhan, and A. Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014.

[76] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 2012.

[77] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[78] M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0). *Optimization Methods and Software*, 18(4):491–505, 2003.

[79] T. Ojala, M. Pietikainen, and D. Harwood. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In *International Conference on Pattern Recognition (ICPR)*, 1994.

[80] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[81] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *ACM International Conference on Multimedia*, 2014.

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

[83] J. Xiao, J. Hays, K. A. Ehinger, and A. Oliva. SUN database: Large-scale scene recognition from abbey to zoo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[84] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using Places database. In *Conference on Neural Information Processing Systems (NIPS)*, 2014.

[85] M. Lapin, M. Hein, and B. Schiele. Loss functions for top-k error: Analysis and insights. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[86] L. Wang, S. Guo, W. Huang, and Y. Qiao. Places205-VGGNet models for scene recognition. *arXiv preprint arXiv:1508.01667*, 2015.

[87] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[88] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[89] J. C. Chen, V. M. Patel, and R. Chellappa. Unconstrained face verification using deep CNN features. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016.

[90] D. Yi, Z. Lei, S. Liao, and S. Z. Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014.

[91] O. M. Parkhi, A. Vedaldi, and A. Zisserman. Deep face recognition. In *British Machine Vision Conference (BMVC)*, 2015.

[92] S. Tu and J. Wang. Practical first order methods for large scale semidefinite programming. Technical report, University of California, Berkeley, 2014.

[93] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[94] M. Yamashita, K. Fujisawa, K. Nakata, M. Nakata, M. Fukuda, K. Kobayashi, and K. Goto. A high-performance software package for semidefinite programs: SDPA 7. Technical report, Tokyo Institute of Technology, 2010.

[95] S. Lahabar and P. J. Narayanan. Singular value decomposition on GPU using CUDA. In *IEEE International Symposium on Parallel & Distributed Processing*, 2009.

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

[97] E. J. Candes, J. K. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006.

[98] E. J. Candes and M. B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008.

[99] P. J. Bickel, Y. Ritov, and A. B. Tsybakov. Simultaneous analysis of Lasso and Dantzig selector. *The Annals of Statistics*, pages 1705–1732, 2009.

[100] E. J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

[101] R. G. Baraniuk, V. Cevher, M. F. Duarte, and C. Hegde. Model-based compressive sensing. *IEEE Transactions on Information Theory*, 56(4):1982–2001, 2010.

[102] C. Hegde, P. Indyk, and L. Schmidt. A nearly-linear time framework for graph-structured sparsity. In *International Conference on Machine Learning (ICML)*, 2015.

[103] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

[104] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[105] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: interpretable representation learning by information maximizing generative adversarial nets. In *Conference on Neural Information Processing Systems (NIPS)*, 2016.

[106] A. Lamb, V. Dumoulin, and A. Courville. Discriminative regularization for generative models. *arXiv preprint arXiv:1602.03220*, 2016.

[107] M. Lustig, D. L. Donoho, J. M. Santos, and J. M. Pauly. Compressed sensing MRI. *IEEE Signal Processing Magazine*, 25(2):72–82, 2008.

[108] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

[109] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

[110] Z. C. Lipton and S. Tripathi. Precise recovery of latent vectors from generative adversarial networks. *arXiv preprint arXiv:1702.04782*, 2017.

[111] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[112] J. T. Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2015.

[113] A. Odena. Semi-supervised learning with generative adversarial networks. *arXiv preprint arXiv:1606.01583*, 2016.

[114] M. A. Davenport, M. F. Duarte, M. B. Wakin, J. N. Laska, D. Takhar, K. F. Kelly, and R. G. Baraniuk. The smashed filter for compressive classification and target recognition. In *Electronic Imaging*, 2007.

[115] V. Cevher, A. Sankaranarayanan, M. Duarte, D. Reddy, R. Baraniuk, and R. Chellappa. Compressive sensing for background subtraction. In *European Conference on Computer Vision (ECCV)*, 2008.

[116] O. Maillard and R. Munos. Compressed least-squares regression. In *Conference on Neural Information Processing Systems (NIPS)*, 2009.

[117] R. Calderbank, S. Jafarpour, and R. Schapire. Compressed learning: Universal sparse dimensionality reduction and learning in the measurement domain. 2009.

[118] S. Lohit, K. Kulkarni, P. Turaga, J. Wang, and A. C. Sankaranarayanan. Reconstruction-free inference on compressive measurements. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Workshop Track*, 2015.

[119] S. Lohit, K. Kulkarni, and P. Turaga. Direct inference on compressive measurements using convolutional neural networks. In *IEEE International Conference on Image Processing (ICIP)*, 2016.

[120] J. Yang, J. Wright, T. S. Huang, and Y. Ma. Image super-resolution via sparse representation. *IEEE Transactions on Image Processing*, 19(11):2861–2873, 2010.

[121] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

[122] J. Kim, J. Kwon Lee, and K. Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[123] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[124] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[125] Ziwei L., Ping L., Xiaogang W., and Xiaoou T. Deep learning face attributes in the wild. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

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

[127] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[128] T. Kim. A TensorFlow implementation of: Deep convolutional generative adversarial networks, 2017. Software available at https://github.com/carpedm20/DCGAN-tensorflow.

[129] J. M. Bioucas-Dias and M. A. T. Figueiredo. A new TwIST: Two-step iterative shrinkage/thresholding algorithms for image restoration. *IEEE Transactions on Image Processing*, 16(12):2992–3004, 2007.

[130] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[131] T. Goldstein and S. Osher. The split Bregman method for ℓ_1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.

[132] W. Yin, S. Osher, D. Goldfarb, and J. Darbon. Bregman iterative algorithms for ℓ_1-minimization with applications to compressed sensing. *SIAM Journal on Imaging Sciences*, 1(1):143–168, 2008.

[133] V. N. Katsikis, D. Pappas, and A. Petralias. An improved method for the computation of the Moore-Penrose inverse matrix. *Applied Mathematics and Computation*, 2011.

[134] P. Courrieu. Fast computation of Moore-Penrose inverse matrices. *arXiv preprint arXiv:0804.4809*, 2008.

[135] S Lahabar and PJ Narayanan. Singular value decomposition on GPU using CUDA. In *IEEE International Symposium on Parallel & Distributed Processing*, 2009.

[136] V Volkov and J Demmel. LU, QR and Cholesky factorizations using vector capabilities of GPUs. Technical report, 2008.

[137] C. Musco and C. Musco. Randomized block Krylov methods for stronger and faster approximate singular value decomposition. In *Conference on Neural Information Processing Systems (NIPS)*, 2015.

[138] M. Karpinski and A. Macintyre. Polynomial bounds for VC dimension of sigmoidal and general Pfaffian neural networks. *Journal of Computer and System Sciences*, 54(1):169–176, 1997.

[139] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[140] D. Haussler, N. Littlestone, and M. K. Warmuth. Predicting $\{0, 1\}$-functions on randomly drawn points. *Information and Computation*, 115(2):248–292, 1994.

[141] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82(3):247–261, 1989.

[142] N. Z. Shor. Quadratic optimization problems. *Soviet Journal of Computer and and Systems Sciences*, 25(6):1–11, 1987.

[143] A. Ben-Tal. Conic and robust optimization. 2002.