Title

Nonconvex Models and Algorithms for Sparse Regularization in Deep Learning and Image Segmentation

Author

Bui, Kevin

Publication Date

2022

UNIVERSITY OF CALIFORNIA,
IRVINE

Nonconvex Models and Algorithms for Sparse Regularization in Deep Learning and Image Segmentation

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mathematics

by

Kevin Bui

Dissertation Committee:
Professor Jack Xin, Chair
Professor Long Chen
Professor Knut Sølna

2023

DEDICATION

To all my colleagues, mentors, friends, and family who have supported me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

There are many people whom I am deeply grateful for because they have helped me lead a very fulfilling life that allows me to earn my Ph.D. in Mathematics at UC Irvine. Hence, the acknowledgement will be very long to read.

First, I would like to thank Professor Jack Xin for being my adviser. Back in beginning of 2018 when I was deciding between Ph.D. programs to attend, Professor Xin personally reached out to me and did his best to convince me to come to UC Irvine. He helped me obtain the UC Irvine Diversity Recruitment Fellowship to supplement my first-year Ph.D. stipend. Moreover, he persuaded me to do deep learning research with him when I was initially dubious of the success and practicality of deep learning at the time. His persistence broke through my doubts and stubbornness that I ultimately decided to attend UC Irvine, where I never would have thought to be a comfortable, conducive environment for my academic and professional growth. As my Ph.D. adviser, Professor Xin provided me great starting ideas that I explored and worked on. As a result, I became heavily invested in researching novel optimization methods to better compress deep learning architectures. Moreover, I am grateful that he has allowed me to pursue other research endeavors, such as in image processing. Together with him, we wrote more than five papers, and I will continue to collaborate with him on many more projects after my Ph.D. Recently, I am most thankful for his career advice in Summer 2022 when I was unsure about accepting the full-time job offer as a data scientist at Lacework. Professor Xin was very grounded, practical, and reasonable that he helped influence my decision to accept the offer for a possibly better future for myself. Without him as my adviser, my achievements throughout my Ph.D. would not have existed. Moreover, because of him, I met some of my most valuable research collaborators.

One of the most important collaborators whom Professor Xin introduced me to was Professor Fredrick Park from Whittier College. We collaborated on various deep learning and image segmentation projects. In our deep learning projects, Professor Park was helpful in implementing some of the deep learning codes that needed to be run and having more GPUs available when I ran out of Professor Xin's GPUs. Without his help, I would not have been able to publish some of the deep learning papers on time. I truly hope that his electricity bill was not too expensive because of me. Besides the deep learning research, he inspired the image segmentation half of my thesis. He reignited my interest in image processing with his research presentation at UC Irvine Computer Science AI/ML Seminar in November 2018. In July 2019, he gave me access to his previous image segmentation work and code that I extended them to the multiphase and color cases while developing algorithms for them. This work eventually led to my first publication in SIAM Journal on Imaging Sciences, which is one of my proudest achievements as a computational mathematician. Professor Park also introduced me to Professor Yifei Lou from UT Dallas, who became another valuable collaborator.

Professor Lou was also important in regards to the segmentation half of my thesis. She is one of the most hardworking people I know because sometimes she worked on weekends revising some of the papers we wrote together in order to provide me edits and feedback on time.

In her revisions, she was very thorough and careful with the clarity and style of our papers. Working with Professor Lou immensely improved the cogency of the image processing papers we wrote together, especially after several email exchanges, that journal reviewers praised how well written they were. I truly respect her writing and editing abilities as a researcher. Moreover, Professor Lou was influential in my research taste in computational mathematics. Reading many of her research papers shaped my interest in nonconvex regularization and optimization that I picked up many ideas and techniques, inspiring new deep learning and image processing projects to pursue. Working with her, I am definitely an expert in weighted anisotropic–isotropic total variation that she and Professor Jack Xin devised several years ago.

Thanks to Professor Xin's connections, I had the opportunity to collaborate with Dr. Shuai Zhang and Dr. Yingyong Qi from Qualcomm. They were helpful in their feedback and editing in my deep learning papers. Most importantly, they were critical in helping me win the Best Paper Award at ISVC'20. Moreover, I am grateful for the email exchanges I had with Professor Qi, especially in introducing me to some interesting deep learning papers.

At UC Irvine Mathematics Department, I would like to thank my following classmates for enriching my time at UC Irvine in one way or another: Thomas Beardsley, William Black, Eric Carmody, Jinghao Chen, Matthew Cheung, David Clausen, Kathryn Dover, Liam Hardiman, Matthew Hirning, Omar Hurtado, Vignesh Iyer, Joshua Jordan, Hannah Knight, Jesse Kreger, Long-sin "Joe" Li, David Lin, Samuel Lopez, Daniel Morrison, Bao Nguyen, Jennifer Pi, Matthew Powell, Fernando Quintino, Jessica Schirle, Ashley Shade, Jiayi Shen, Daniel Shvartsman, Yonathan Stone, Alex Sutherland, Tin Yau "Whisky" Tsang, Dong Yan, and Maureen Zhang. I want to personally thank Jorge Silva Guzman for being my study partner for the Real Analysis qualifying exam when I somehow had no one else to study with and for bringing together our algebra classmates to overcome the horribly taught Math 230 sequence in the 2019-2020 school year. I also want to personally thank Michael Hehmann for introducing me to the food scene in Irvine and outside of it and inviting me along with our other classmates to many foodie adventures. Also his sense of humor is truly unforgettable. Lee Fisher is another person I want to thank personally because he helped me with some very difficult homework problems and projects in Math 270AB, Math 225C, and Math 226A. Despite being one of the most intelligent people I met, he is also one of the most down-to-earth and easygoing people to work with. Next I want to thank Jingrong Wei for the fruitful discussion I had about her research in optimization. There aren't many students in the department who research optimization, so I was happy to find someone to talk about optimization with. Elisha Dayag is a recent collaborator I want to thank, especially for helping me start the transformed ℓ_1 segmentation research project that I did not have time to work on myself. Despite being a second-year, he has the drive and enthusiasm to be a budding computational mathematician. I want to thank Xiaowen Zhu for being an amazing TA for Math 210 that because of her, I was able to pass the Real Analysis qualifying exam in my first try. Moreover, she is one of the most amazing Splatoon players I played with and I enjoyed learning about her research in mathematical physics. Lastly, I want to thank my Ph.D. brothers: Carson Hu, Thu Dinh, Ziang Long, Wesley Whiting, Biao Yang, and Yunling Zheng. I want to especially thank my Ph.D. brothers Zhijian Li, Nhat Thanh Van

Tran, and Fanghui Xue. I enjoyed my times hanging out with them and talking about our research together. Zhijian Li is also a great Splatoon player to play with, who is a vastly better than I am. Nhat Thanh Van Tran helped me start a new deep learning project on image processing and he was fun to talk with about random Vietnamese food and culture. Fanghui Xue is one of the most considerate people I met and he is an important collaborator for one of my deep learning projects.

In addition to my classmates, I want to thank some of the professors at UC Irvine Math Department. First, I want to thank Professor Peyam Ryan Tabrizian. He was one of the most outgoing postdocs I met in the department and we went on several foodie adventures together with Michael Hehmann. I also enjoyed the teaching lunches he hosted, where we talked about our students. The department felt like it was missing something after he left in 2020. Next I want to thank Professor Anna Ma. I first met her when we were undergraduates at UCLA and I never would have thought to see her again at UC Irvine. Her personality definitely lightens up the department culture and it was fun discussing with her about her research. I was happy that the mathematics department hired her as an assistant professor because she is a competent applied mathematician and the department needs more people like her to be a more conducive environment for applied and computational mathematics. I also want to thank Professor Svetlana (Lana) Jitomirskaya and Professor Roman Vershynin. Despite her unconventional teaching method, Lana taught measure theory very well that I fully understood the subject much better than any other math classes I took at UC Irvine. Moreover, her stories about famous mathematicians were a delight to listen to during lectures. Professor Vershynin was one of the best lecturers I had in the Math 270 sequence and Math 271A. I was utterly amazed by how lucidly he teaches probability theory with a perfect balance of both rigor and intuition. Even during COVID, his online lectures were as good as his in-person lectures. It was also an honor to have him serve as a committee member for my candidacy exam. Both professors solidified my love for real analysis and its related branches. Lastly, I want to thank Professor Long Chen and Professor Knut Sølna for serving on both my candidacy and Ph.D. defense committees. I want to thank Professor Long Chen even more for being a lecturer in my Math 225BC and Math 226A classes. He supplemented my knowledge in convex optimization and deep learning and he introduced me to numerical PDEs. I also picked up some neat MATLAB tricks from him.

There are some more people I met at UC Irvine whom I want to thank. I want to thank my students Abhishek Devarajan and Xinhan Huang for attending my convex optimization classes under the Directed Reading Program in the Mathematics Department. Within the SIAM Journal Club on Machine Learning, I want to thank the following members for their attendance, participation, and discussion: Ahyeon Hwang, Axel Almet, Eric Bourgain-Chang, Katrina Lee Bartas, Matthew Duong, Muhammad Hasan Celik, and Yueqi Ren. I also want to give a shout out to Pieter Derdeyn for advertising the club when he was SIAM president and Vincent Zaballa for presenting interesting, advanced machine learning papers and succeeding me as organizer.

During my Ph.D., I was fortunate to have three internships, where I met new friends and mentors. First, I want to thank my mentor Dr. Zichao Wendy Di from Argonne National Laboratory. During my internship as a Givens Associate in Summer 2021, she was a great research mentor who introduced me to image ptychography and encouraged me to learn stochastic optimization algorithms. Together we were able to devise a novel stochastic ADMM algorithm for image ptychography, but unfortunately I did not include our work into this thesis or else this already long thesis will have fifty more pages. However, I will present our work at SIAM CSE 2023 that Dr. Di invited me to. Next, I want to thank my internship mentor Dr. Alessandro Panella from Lacework. During my data science internship in Spring 2022, he fostered my growth as a budding data scientist by having me work on impactful projects and collaborate with various employees outside our team. Moreover, he was pivotal in helping me obtain the full-time offer at Lacework. Finally, I want to thank my colleagues at my Summer 2022 internship at InterDigital, Inc. I thank my fellow intern colleagues Eric Lei and Pranav Kadam for the enlightening conversations about deep learning, point cloud processing, and their respective research areas. Moreover, I thank my senior colleagues Dr. Junghyun Ahn for his feedback and questions on my project and Dr. Muhammad Asad Lodhi for helping me become familiar with the computing environment. Lastly, I thank my mentors Dr. Dong Tian for teaching me about point cloud compression and providing me motivation for my project and Dr. Jiahao Pang for closely supervising my point cloud processing project. Especially because of Jiahao Pang, I became a lot more familiar with programming deep learning architectures in Pytorch than before I started my internship.

One of the most important people I want to thank is Professor Sanjay Mehrotra from Northwestern University. I am greatly indebted to him because he is one of the most compassionate people I met in my life. After I was dismissed from the Ph.D. program at Northwestern University, he took a huge risk by hiring me as his research assistant because he still saw potential in me, thereby saving me from my darkest moments. Burdened by my failures in the Ph.D. program at Northwestern University, I was provided various research opportunities by Professor Mehrotra to redeem myself. As a result, I became hardworking and determined and I learned so much about machine learning methodologies and techniques and applied them in healthcare engineering. Together we wrote five papers, which boosted my confidence and solidified my work ethics necessary for the Ph.D. program at UC Irvine. Furthermore, we continue to collaborate in writing a healthcare engineering textbook together with Professor Hari Balasubramanian from University of Massachusetts, Amherst, whom I thank for writing the chapters I did not have time to write during my Ph.D. at UC Irvine and for providing me constructive feedback for the machine learning chapters I wrote. After my first year at UC Irvine, Professor Mehrotra invited me to work for his startup Medecipher, Inc., as a once-in-a-lifetime opportunity. I accepted the offer to not only to supplement my Ph.D. stipend but also to earn valuable industry experience. At Medecipher, Inc., I worked with some amazing people, such as the CEO Stephanie Gravenor, CTO Steve Tinsley, COO Rada Yovovich, software engineers Parker Eischen and Connor Graflund, and data science intern James Vongphasouk. I am extremely grateful to Professor Mehrotra for fostering my growth as a researcher, healthcare engineer, and data scientist.

I would like to thank my following friends for enriching my Ph.D. life in one way or another: Joshua Chan, Rodrick Dass, Nikhat Elias, Mo Emish, Andrea Treveño Gavito, Kyung Ha, Absaar Javed, Nicolas Jeung, Zeyad Kelani, David Kes, Brianna Bohyun Kim, Vikram Kilambi, Cheolmin Kim, Wonjun Lee, Christina Luu, Binh K. Nguyen, Peter Nguyen, Nhi Minh Nguyen, Xuan Nguyen, David Renteria, Mark Semelhago, Alto Senda, Hao-Jun Michael Shi, Amy Tang, Courtney Tran, Kimberly Tran, Navin Velazco, Victor Wong, John Wu, and Yiben Yang. I want to especially thank one of my best friends Nicolas Bravo for not only supporting me throughout my Ph.D. journey but also leading me there by inspiring me to become a mathematics major and to learn programming in my first year of college. Lastly, I want to thank my girlfriend Tham Minh Thi Bui for her everlasting support and love and being there for me at my best and worst of times. She was the only person willing to listen to my research presentations when I needed to practice, even if it is a hundred times and she does not understand what my research is about. Moreover, she showed me that there is more to life than research and academics and helped me reconnect with my Vietnamese heritage.

I want to thank my relatives Khoa Dinh, Duc Ho, Hoa Ho, Thu Thao Hoang Ho, Hieu Hoang, Huy Hoang, Kim Hoang, Phuc Hoang, Phuong Hoang, Toan Hoang, Trung Hoang, Tung Hoang, Dinh Tran Huyen, Vi Le, and Cynthia Rubi.

Last and not least, I want to thank my parents Tan Bui and Huong Hoang and my sister Cathleen Bui for their unconditional love and support. Striving to be the best mother possible, Huong Hoang has always loved and cared for me by doing many things to ensure that nothing impedes my Ph.D. research. However, as her son, I feel guilty that she might end up forgetting to take care of herself. That is why I thank my father Tan Bui for keeping my mother in check and reminding her that I can fully take care of myself. Also he is the reason why I am a computational mathematician today because he fosters my interest in math at an early age by teaching me how to add when I was four years old. Despite being younger than me, Cathleen Bui is someone whom I viewed as an equal and my best friend. She gives valuable personal advice when I need it and she calls me out for my stupid moments. She helps me become an empathetic and caring person.

Funding Acknowledgement: This dissertation was partially supported by NSF grants IIS-1632935, DMS-1854434, DMS-1924548, DMS-1952644, and DMS-2151235 and the Data Science Fellowship at the UCI School of Physical Sciences.

VITA

Kevin Bui

EDUCATION

Doctor of Philosophy in Mathematics **2023**
University of California, Irvine *Irvine, CA*

Master of Science in Mathematics **2020**
University of California, Irvine *Irvine, CA*

Master of Science in Industrial Engineering & Management Sciences **2016**
Northwestern University *Evanston, IL*

Bachelor of Science in Mathematics of Computation **2011**
University of California, Los Angeles *Los Angeles, CA*

WORK EXPERIENCE

Research & Innovation Intern **Jun. 2022 - Sept. 2022**
InterDigital

Data Science Intern **Mar. 2022 - Jun. 2022**
Lacework

Givens Associate **Jun. 2021 - Sept. 2021**
Argonne National Laboratory

Operations Researcher **Jul. 2019 - Jun. 2022**
Medecipher, Inc.

Research Technologist **Aug. 2016 - Jun. 2019**
Northwestern University

ABSTRACT OF THE DISSERTATION

Nonconvex Models and Algorithms for Sparse Regularization in Deep Learning and Image Segmentation

By

Kevin Bui

Doctor of Philosophy in Mathematics

University of California, Irvine, 2023

Professor Jack Xin, Chair

In this thesis, we propose sparse, nonconvex optimization models in both areas of deep learning and image segmentation and develop algorithms to solve them. To be more specific, we design optimization algorithms that perform channel pruning of convolutional neural networks (CNNs) in order to compress them without deteriorating their accuracy, while we investigate a nonconvex alternative of total variation (TV) to obtain sharper segmentation results. Various numerical experiments are conducted to showcase the effectiveness and performance of the proposed nonconvex models.

In Part I, we follow the direction of sparse optimization to perform channel pruning of over-parametrized CNNs. In one class of models, we propose a family of nonconvex sparse group lasso that blends nonconvex regularization (e.g., transformed ℓ_1, $\ell_1 - \ell_2$, and ℓ_0) that induces sparsity onto the individual weights and $\ell_{2,1}$ regularization onto the output channels of a layer. Next, we provide two directions to improve network slimming, a channel pruning method that applies ℓ_1 regularization on the scaling factors in the batch normalization layers. In one direction, we replace ℓ_1 regularization with nonconvex alternatives, such as transformed ℓ_1 and $\ell_p, 0 < p < 1$, and derive their subgradient formulas to perform subgradient descent during training. In another direction, because subgradient descent is used

for network slimming as the default optimization algorithm with theoretical and numerical flaws, we propose a theoretically convergent algorithm called proximal network slimming that trains CNNs to be more robust against channel pruning. As a result, fine tuning CNNs after channel pruning becomes optional by training with our proposed proximal network slimming.

In Part II, we examine the applications of the weighted anisotropic–isotropic total variation (AITV), the $\ell_1 - \alpha\ell_2$ variant of TV, in image segmentation. In one direction, we replace the TV regularization in the Chan–Vese segmentation model and a fuzzy region competition model by AITV. To deal with the nonconvex nature of AITV, we apply the difference-of-convex algorithm (DCA), in which the subproblems can be minimized by the primal-dual hybrid gradient method with linesearch. In another direction, we design an efficient, multi-stage image segmentation framework that incorporates AITV. The segmentation framework generally consists of two stages: smoothing and thresholding. In the first stage, a smoothed image is obtained by an AITV-regularized Mumford-Shah model, which can be solved efficiently by the alternating direction method of multipliers with a closed-form solution of a proximal operator of the $\ell_1 - \alpha\ell_2$ regularizer. In the second stage, we threshold the smoothed image by k-means clustering to obtain the final segmentation result.

Part I

Compression of Deep Learning Models

Chapter 1

Introduction

1.1 Motivation

In the past years, convolutional neural networks (CNNs) have evolved into superior models for various computer vision tasks, such as image classification [89, 111, 192], image segmentation [51, 135, 185], and object detection [75, 96, 183]. These models often contain millions of weight parameters that often exceed the number of training data. This is a double-edged sword since on one hand, large models allow for high accuracy, while on the other, they contain many redundant parameters that lead to overparametrization. Overparametrization is a well-known phenomenon in CNN models [59, 11] that results in overfitting, learning useless random patterns in data [243], and having inferior generalization. Additionally, training a highly accurate CNN is computationally demanding. State-of-the-art CNNs such as ResNet [89] can have up to at least a hundred layers and thus require millions of parameters to train and billions of floating-point-operations to execute. Consequently, deploying CNNs in low-memory devices, such as mobile smartphones, is difficult, making their real-world applications limited.

To make CNNs more practical, many works suggest several different directions to compress large CNNs or to learn smaller, more efficient models from scratch. Low-rank approximation [59, 98, 215, 220, 221] minimizes network redundancy by approximating the network's weight matrices with low-rank matrices. Weight quantization [52, 57, 121, 255, 232] replaces the floating-point weights with quantized weights, such as binary weights $\{-1, +1\}$ and ternary weights $\{-1, 0, +1\}$. Pruning [2, 84, 122, 92] determines which weights, filters, and/or channels are unnecessary and removes them from the network. Lastly, another popular direction is to sparsify the CNN while training it [6, 47, 189, 214]. Sparsity can be imposed on various types of structures existing in CNNs, such as filters and channels [214]. In the following section, we review the recent literature for the aforementioned directions.

1.2 Compression Techniques for CNN

Low-rank decomposition. Low-rank decomposition aims to reduce weight matrices to their low-rank structures for faster computation and more efficient storage. One set of methods focuses on decomposing pre-trained weight tensors. Denton *et al.* [59] compressed the weight tensors of convolutional layers using singular value decomposition to approximate them. Jaderberg *et al.* [98] exploited the redundancy between different feature channels and filters to approximate a full-rank filter bank in CNNs by combinations of a rank-one filter basis. On the other hand, there are methods that train CNNs with low-rank weight matrices from scratch. Tai *et al.*[198] incorporated low-rank tensor decomposition into their CNN training algorithm. Wen *et al.* [215] proposed *force regularization* to train a CNN towards having a low-rank representation. Xu *et al.* [220, 221] developed trained rank pruning, an optimization scheme that incorporates low-rank decomposition into the training process. Trained rank pruning was further strengthened by nuclear norm regularization.

Weight Quantization. Quantization aims to represent weights with low-precision values

($\leq$8 bits arithmetic). The simplest form of quantization is binarization, constraining weights to only two values. Courbariaux *et al.* [57] proposed BinaryConnect, a method that trains deep neural networks (DNNs) with strictly binary weights. Neural networks with ternary weights have also been developed and investigated. Li *et al.* [121] created ternary weight networks, where the weights are only $-1, 0$, or $+1$. Zhu *et al.* [255] proposed Trained Ternary Quantization that constrains the weights to more general values $-W^n, 0$, and W^p, where W^n and W^p are parameters learned through the training process. For more general quantization, Yin *et al.* [232] developed BinaryRelax, which relaxes the quantization constraint into a continuous regularizer for the optimization problem needed to be solved in CNNs. Later, Bai *et al.*[15] proposed Proxquant, a stochastic proximal gradient method for quantizing networks while training them.

Pruning. Pruning methods identify which weights, filters, and/or channels in CNNs are redundant and remove them from the networks. Early works focus on pruning weights. Han *et al.* [84] proposed a three-step framework to first train a CNN, prune weights if their norms are below a fixed threshold, and retrain the compressed CNN. Aghasi *et al.* [2, 3] proposed using convex optimization to determine which weights to prune while preserving model accuracy. For CNNs, channel or filter pruning is preferred over individual weight pruning since the former significantly eliminates more unnecessary weights. Many works [6, 27, 122, 160, 189, 214, 234] have imposed group regularization onto various CNN structures, such as filters and channels. Li *et al.*[125] incorporated a sparsity-inducing matrix corresponding to each feature map and imposed group regularization row-wise and column-wise onto this matrix to determine which filters to remove. Lin *et al.*[132] pruned filters that generate low-rank feature maps. Hu *et al.*[92] devised network trimming that iteratively removes zero-activation neurons from the CNN and retrains the compressed CNN. Luo *et al.*[145, 146] developed the Thinet framework, which formulates channel pruning as an NP-hard optimization problem that is solved by a greedy approach combined with fine tuning the optimization problem itself. Rather than regularizing the weight parameters, Liu *et al.*[134]

developed network slimming, where they applied ℓ_1 regularization on the scaling factors in the batch normalization layers in a CNN to determine which of their corresponding channels are redundant to remove and then they retrained the pruned CNN to restore its accuracy. Zhao *et al.*[252] applied probabilistic learning onto the scaling factors to identify which redundant channels to prune with minimal accuracy loss, making retraining unnecessary. Instead of regularizing the inherent scaling factors of the batch normalization layers, Lin *et al.*[133] introduced an external soft mask as a set of parameters corresponding to the CNN structures (e.g., filters and channels) and regularized the mask by adversarial learning.

Sparse optimization. Sparse optimization methods introduce a sparse regularizer term to the loss function of the CNN so that the CNN is trained to have a compressed structure from scratch. BinaryRelax [232] and network slimming [134] are examples of sparse optimization methods for CNNs. Alvarez and Salzmann [6] and Scardapane *et al.* [189] applied group lasso [238] and sparse group lasso [189] to CNNs to obtain group-sparse networks. Nonconvex regularizers have also been examined recently. Xue and Xin [225] used ℓ_0 and Tℓ_1 regularization in three-layer CNNs that classify shaky vs. normal handwriting. Both Ma *et al.* [153] and Pandit *et al.*[170] proposed a regularizer that combines group sparsity and Tℓ_1 and applied it to CNNs for image classification. Li *et al.*[125] introduced sparsity-inducing matrices into CNNs and imposed group sparsity on the rows or columns via ℓ_1 or other nonconvex regularizers to prune filters and/or channels.

1.3 Organization of Part I

In Part I, we propose three methods to compress CNNs. These methods are in line with channel pruning and sparse optimization. In Chapter 2, we propose a family of sparse regularizers called nonconvex sparse group lasso to jointly prune channels and individual weights in CNNs. This chapter is derived from [27]. In Chapter 3, we improve network slimming by replaing ℓ_1 regularization on the scaling factors in the batch normalization layers with nonconvex, sparse regularization to potentially prune more channels. This chapter is adapted from [25, 26]. Lastly, in Chapter 4, we propose an alternative algorithm to perform network slimming so that fine tuning becomes optional. We conclude Part I in Chapter 5.

Chapter 2

Structured Sparsity of Convolutional Neural Networks via Nonconvex Sparse Group Regularization

In this chapter, we propose a family of group regularization methods that balances both group lasso for group-wise sparsity and nonconvex regularization for element-wise sparsity. The family extends sparse group lasso by replacing the ℓ_1 penalty term with a nonconvex penalty term. The nonconvex penalty terms considered are ℓ_0, $\ell_1 - \alpha\ell_2$, transformed ℓ_1, and SCAD. The proposed family is supposed to yield a more accurate and/or more compressed network than sparse group lasso since ℓ_1 suffers various weaknesses due to being a convex relaxation of ℓ_0. We develop an algorithm to optimize loss functions equipped with the proposed nonconvex, group regularization terms for DNNs.

2.1 Model and Algorithm

2.1.1 Preliminaries

Given a training dataset consisting of N input-output pairs $\{(x_i, y_i)\}_{i=1}^N$, the weight parameters of a DNN are learned by optimizing the following objective function:

$$\min_W \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(h(x_i, W), y_i) + \lambda \mathcal{R}(W), \tag{2.1}$$

where

- W is the set of weight parameters of the DNN.
- $h(\cdot, \cdot)$ is the output of the DNN used for prediction.
- $\mathcal{L}(\cdot, \cdot) \geq 0$ is the loss function that compares the prediction $h(x_i, W)$ with the ground-truth output y_i. Examples include cross-entropy loss function for classification and mean-squared error for regression.
- $\mathcal{R}(\cdot)$ is the regularizer on the set of weight parameters W.
- $\lambda > 0$ is a regularization parameter for $\mathcal{R}(\cdot)$.

The most common regularizer used for DNNs is ℓ_2 regularization $\|\cdot\|_2^2$, also known as weight decay. It prevents overfitting and improves generalization because it enforces the weights to decrease proportionally to their magnitudes [112]. Sparsity can be imposed by pruning weights whose magnitudes are below a certain threshold at each iteration during training. However, an alternative regularizer is the ℓ_1 norm $\|\cdot\|_1$, also known as the lasso penalty [199]. The ℓ_1 norm is the tightest convex relaxation of the ℓ_0 penalty [64, 71, 203] and it yields a sparse solution that is found on the corners of the 1-norm ball [87, 139]. Theoretical results justify the ℓ_1 norm's ability to reconstruct sparse solution in compressed sensing. When a sensing matrix satisfies the restricted isometry property, the ℓ_1 norm recovers the sparse solution exactly with high probability [32, 71, 203]. On the other hand, the null space property is a necessary and sufficient condition for ℓ_1 minimization to guarantee exact recovery of sparse solutions [55, 71]. Being able to yield sparse solutions, the ℓ_1 norm has gained popularity in other types of inverse problems such as compressed imaging [103, 148] and image segmentation [105, 104, 120] and in various fields of applications such as geoscience [188], medical imaging [103, 148], machine learning [30, 199, 107, 166, 228], and traffic flow network [231]. Unfortunately, element-wise sparsity by ℓ_1 or ℓ_2 regularization in CNNs may not yield meaningful speedup as the number of filters and channels required for computation and inference may remain the same [214].

To determine which filters or channels are relevant in each layer, group sparsity using the group lasso penalty [238] is considered. The group lasso penalty has been utilized in various applications, such as microarray data analysis [154], machine learning [13, 159], and EEG data [128]. Suppose a DNN has L layers, so the set of weight parameters W is divided into L sets of weights: $W = \{W_l\}_{l=1}^{L}$. The weight set of each layer W_l is divided into N_l groups (e.g., channels or filters): $W_l = \{w_{l,g}\}_{g=1}^{N_l}$. The group lasso penalty applied to W_l is formulated as

$$\mathcal{R}_{GL}(W_l) = \sum_{g=1}^{N_l} \sqrt{\#w_{l,g}} \|w_{l,g}\|_2 = \sum_{g=1}^{N_l} \sqrt{\#w_{l,g}} \sqrt{\sum_{i=1}^{\#w_{l,g}} w_{l,g,i}^2}, \tag{2.2}$$

where $w_{l,g,i}$ corresponds to the weight parameter with index i in group g in layer l and the term $\#w_{l,g}$ denotes the number of weight parameters in group g in layer l. Because group sizes vary, the constant $\sqrt{\#w_{l,g}}$ is multiplied in order to rescale the ℓ_2 norm of each group with respect to the group size, ensuring that each group is weighed uniformly [238, 191, 159]. The group lasso regularizer imposes the ℓ_2 norm on each group, forcing weights of the same groups to decrease altogether at every iteration during training. As a result, the groups of weights are pruned when their ℓ_2 norms are negligible, resulting in a highly compact network compared to element-sparse networks.

As an alternative to group lasso that encourages feature sharing, exclusive sparsity [254] enforces the model weight parameters to compete for features, making the features discriminative for each class in the context of classification. The regularization for exclusive sparsity is

$$\frac{1}{2}\sum_{g=1}^{N_l}\|w_{l,g}\|_1^2 = \frac{1}{2}\sum_{g=1}^{N_l}\left(\sum_{i=1}^{\#w_{l,g}}|w_{l,g,i}|\right)^2. \tag{2.3}$$

Now, within each group, sparsity is enforced. Because exclusivity cannot guarantee the optimal features since some features do need to be shared, exclusive sparsity can be combined with group sparsity to form combined group and exclusive sparsity (CGES) [234]. CGES is formulated as

$$\mathcal{R}_{CGES} = \sum_{g=1}^{N_l}\left[(1-\mu_l)\sqrt{\sum_{i=1}^{\#w_{l,g}} w_{l,g,i}^2} + \frac{\mu_l}{2}\left(\sum_{i=1}^{\#w_{l,g}}|w_{l,g,i}|\right)^2\right], \tag{2.4}$$

where $\mu_l \in (0,1)$ is a parameter for balancing exclusivity and sharing among features.

To obtain an even sparser network, element-wise sparsity and group sparsity can be combined and applied together to the training of DNNs. One regularizer that combines these two types of sparsity is the sparse group lasso penalty [191], which is formulated as

$$\mathcal{R}_{SGL_1}(W_l) = \mathcal{R}_{GL}(W_l) + \|W_l\|_1 \tag{2.5}$$

where

$$\|W_l\|_1 = \sum_{g=1}^{N_l}\sum_{i=1}^{\#w_{l,g}}|w_{l,g,i}|.$$

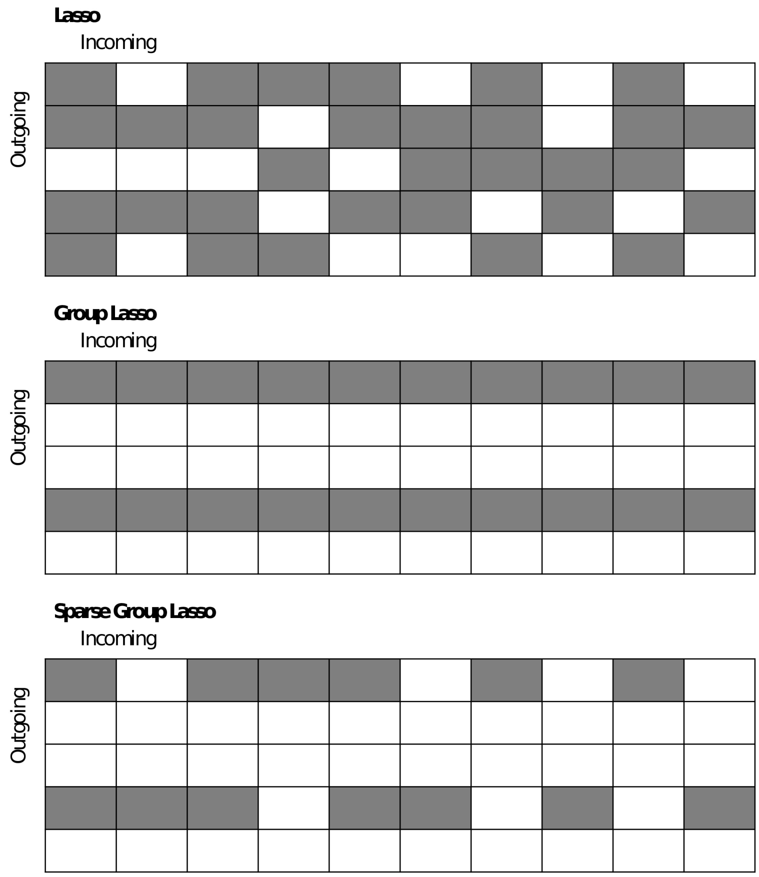


Figure 2.1: Comparison between lasso, group lasso, and sparse group lasso applied to a weight matrix. Entries in white are zero'ed out or removed; entries in gray remain.

Sparse group lasso simultaneously enforces group sparsity by having the regularizer $\mathcal{R}_{GL}(\cdot)$ and element-wise sparsity by having the ℓ_1 norm. This regularizer has been used in machine learning [205], bioinformatics [130, 253], and medical imaging [129].

Figure 2.1 demonstrates the differences between lasso, group lasso, and sparse group lasso applied to a weight matrix connecting a 5-dimensional input layer to a 10-dimensional output layer. In white, the entries are zero'ed out; in gray; the entries are not. Unlike lasso, group lasso results in a more structured method of pruning since three of the five neurons can be zero'ed out. Combined with ℓ_1 regularization on the individual weights, sparse group lasso allows for more weights in the remaining two neurons to be pruned.

2.1.2 Nonconvex Sparse Group Lasso

We recall that the ℓ_1 norm is the tightest convex relaxation of the ℓ_0 penalty, given by

$$\|W_l\|_0 = \sum_{g=1}^{N_l} \sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}|_0 \tag{2.6}$$

where

$$|w|_0 = \begin{cases} 1 \text{ if } w \neq 0 \\ 0 \text{ if } w = 0 \end{cases}$$

when applied to the weight set W_l of layer l. The ℓ_0 penalty is non-convex and discontinuous. In addition, any ℓ_0-regularized problem is NP-hard [71]. These properties make developing convergent and tractable algorithms for ℓ_0-regularized problems difficult, thereby making ℓ_1-regularized problems better alternatives to solve. However, the ℓ_0-regularized problems have been shown to recover better solutions in terms of sparsity and/or accuracy than do ℓ_1-regularized problems in various applications, such as compressed sensing [144], image restoration [16, 40, 63, 251, 143], MRI reconstruction [202], and machine learning [144, 239]. In particular, ℓ_0-regularized inverse problems were demonstrated to be more robust against Poisson noise than are ℓ_1-regualarized inverse problems [249].

A continuous alternative to the ℓ_0 penalty is the SCAD penalty term [69, 149], given by

$$\lambda\|W_l\|_{\text{SCAD}(a)} = \sum_{g=1}^{N_l} \sum_{i=1}^{\#w_{l,g}} \lambda|w_{l,g,i}|_{\text{SCAD}(a)} \tag{2.7}$$

where

$$\lambda|w|_{\mathrm{SCAD}(a)} := \begin{cases} \lambda|w| & \text{if } |w| < \lambda \\ \frac{2a\lambda|w|-w^2-\lambda^2}{2(a-1)} & \text{if } \lambda \leq |w| < a\lambda \\ (a+1)\lambda^2/2 & \text{if } |w| \geq a\lambda \end{cases}$$

for $\lambda > 0$ and $a > 2$. This penalty term enjoys three properties – unbiasedness, sparsity, and continuity – while the ℓ_1 norm, on the other hand, has only sparsity and continuity [69]. In linear and logistic regression, SCAD was shown to outperform ℓ_1 in variable selection [69]. SCAD has been applied to wavelet approximation [8], bioinformatics [22, 209], and compressed sensing [158].

The transformed ℓ_1 penalty term [167] also enjoys the properties of unbiasedness, sparsity, and continuity [149]. In fact, the regularizer is not just continuous but Lipschitz continuous [247]. The term is given by

$$\|W_l\|_{\mathrm{TL1}(a)} = \sum_{g=1}^{N_l} \sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}|_{\mathrm{TL1}(a)} \tag{2.8}$$

where

$$|w|_{\mathrm{TL1}(a)} = \frac{(a+1)|w|}{a+|w|}.$$

In addition, it interpolates the ℓ_0 and ℓ_1 penalties through the parameter a [247] because

$$\lim_{a\to 0^+} |w|_{\mathrm{TL1}(a)} = |w|_0 \quad \text{and} \quad \lim_{a\to\infty} |w|_{\mathrm{TL1}(a)} = |w|.$$

The transformed ℓ_1 penalty term was investigated and was shown to outperform ℓ_1 in compressed sensing [246, 247, 200], deep learning [153, 225, 127], matrix completion [248], and epidemic forecasting [127].

Another Lipschitz continous, nonconvex regularizer is the $\ell_1 - \alpha\ell_2$ penalty given by

$$\|W_l\|_{\ell_1 - \alpha\ell_2} = \|W_l\|_1 - \alpha\|W_l\|_2 = \sum_{g=1}^{N_l} \sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}| - \alpha \sqrt{\sum_{g=1}^{N_l} \sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}|^2}, \tag{2.9}$$

where $\alpha \in (0, 1]$. In a series of works [139, 230, 137, 138], the penalty term $\ell_1 - \ell_2$ with $\alpha = 1$ yields better solutions than does ℓ_1 in various compressed sensing applications especially when the sensing matrix is highly coherent or it violates the restricted isometry property condition. To guarantee exact recovery of sparse solution, $\ell_1 - \ell_2$ only requires a relaxed variant of the null space property [200]. Furthermore, $\ell_1 - \alpha\ell_2$ is more robust against impulsive noise in yielding sparse, accurate solutions for inverse problems than is ℓ_1 [123]. Besides compressed sensing, it has been utilized in image denoising and deblurring [141], image segmentation [172], image inpainting [155], and hyperspectral demixing [67]. In deep learning application, the $\ell_1 - \ell_2$ regularization was used to learn permutation matrices [150] for ShuffleNet [250, 152].

Due to the advantages and recent successes of the aforementioned nonconvex regularizers, we propose to replace the ℓ_1 norm in (2.5) with nonconvex penalty terms. Hence, we propose a family of group regularizers called nonconvex sparse group lasso. The family includes the following:

$$\mathcal{R}_{SGL_0}(W_l) = \mathcal{R}_{GL}(W_l) + \|W_l\|_0 \tag{2.10}$$

$$\mathcal{R}_{SGSCAD(a)}(W_l) = \mathcal{R}_{GL}(W_l) + \|W_l\|_{\mathrm{SCAD}(a)} \tag{2.11}$$

$$\mathcal{R}_{SGTL_1(a)}(W_l) = \mathcal{R}_{GL}(W_l) + \|W_l\|_{\mathrm{TL1}(a)} \tag{2.12}$$

$$\mathcal{R}_{SGL_1 - \alpha L_2}(W_l) = \mathcal{R}_{GL}(W_l) + \|W_l\|_{\ell_1 - \alpha\ell_2}. \tag{2.13}$$

Using these regularizers, we expect to obtain a sparser and/or more accurate network than from using the original sparse group lasso. The ℓ_1 norm can also be replaced with other nonconvex penalties not mentioned in this paper. Refer to [4, 213] to see other nonconvex penalties. However, we focus on the aforementioned nonconvex regularizers because they have closed-form proximal operators required by our proposed algorithm described in the next section.

2.1.3 Notations and Definitions

Before discussing the algorithm, we summarize notations that we will use to save space. They are the following:

- If $V = \{V_l\}_{l=1}^L$ and $W = \{W_l\}_{l=1}^L$, then
  $(V, W) \coloneqq (\{V_l\}_{l=1}^L, \{W_l\}_{l=1}^L) = (V_1, \ldots, V_L, W_1, \ldots, W_L)$.
- $V^+ \coloneqq V^{k+1}$.
- $\tilde{\mathcal{L}}(W) \coloneqq \frac{1}{N}\sum_{i=1}^N \mathcal{L}(h(x_i, W), y_i)$.

In addition, we define the proximal operator for the regularization function $r(\cdot)$ as follows:

$$\operatorname{prox}_{\lambda r}(y) = \arg\min_x \lambda\, r(x) + \frac{1}{2}\|x - y\|_2^2$$

for $\lambda > 0$.

2.1.4 Numerical Optimization

We develop a general algorithm framework to solve

$$\min_W \tilde{\mathcal{L}}(W) + \lambda \sum_{l=1}^L \mathcal{R}(W_l) = \tilde{\mathcal{L}}(W) + \sum_{l=1}^L (\lambda \mathcal{R}_{GL}(W_l) + \lambda r(W_l)) \tag{2.14}$$

where $W = \{W_l\}_{l=1}^L$, $\mathcal{R}$ is either $\mathcal{R}_{SGL_1}$ or one of the nonconvex regularizers (2.10)-(2.13), and $r(\cdot)$ is the corresponding sparsity-inducing regularizer. Throughout the paper, our assumption on (2.14) is the following:

Assumption 2.1. *The function $\tilde{\mathcal{L}}$ is continuously differentiable with respect to W_l for each $l = 1, \ldots, L$.*

By introducing an auxiliary variable $V = \{V_l\}_{l=1}^L$ for (2.14), we have a constrained optimization problem:

$$\begin{aligned}
&\min_{V,W} \quad \tilde{\mathcal{L}}(W) + \sum_{l=1}^{L} \left(\lambda \mathcal{R}_{GL}(W_l) + \lambda r(V_l)\right) \\
&\text{s.t.} \quad V_l = W_l \qquad l = 1, \ldots, L.
\end{aligned} \tag{2.15}$$

The constraints can be relaxed by adding the quadratic penalty terms with $\beta > 0$ so that we have

$$\min_{V,W} F_\beta(V, W) := \tilde{\mathcal{L}}(W) + \sum_{l=1}^{L} \left[\lambda \mathcal{R}_{GL}(W_l) + \lambda r(V_l) + \frac{\beta}{2}\|V_l - W_l\|_2^2\right]. \tag{2.16}$$

With β fixed, (2.16) can be solved by alternating minimization:

$$W^{k+1} = \arg\min_{W} F_\beta(V^k, W) \tag{2.17a}$$

$$V^{k+1} = \arg\min_{V} F_\beta(V, W^{k+1}). \tag{2.17b}$$

To solve (2.17a), we simultaneously update W_l for $l = 1, \ldots L$ by gradient descent

$$W_l^{k+1} = W_l^k - \gamma\Big(\nabla_{W_l}\tilde{\mathcal{L}}(W^k) + \lambda \partial_{W_l}\mathcal{R}_{GL}(W_l^k) - \beta(V_l^k - W_l^k)\Big) \tag{2.18}$$

where $\gamma > 0$ is the learning rate and $\partial_{W_l}\mathcal{R}_{GL}$ is the subdifferential of $\mathcal{R}_{GL}$ with respect to W_l. In practice, (2.18) is performed using stochastic gradient descent (or one of its variants)

Algorithm 1: Algorithm for Nonconvex Sparse Group Lasso Regularization

Initialize V^1 and W^1 with random entries; learning rate γ; regularization parameters λ and β; and multiplier $\sigma > 1$.
Set $j := 1$.
while *stopping criterion for outer loop not satisfied* **do**
  Set $k := 1$.
  Set $W^{j,1} = W^j$ and $V^{j,1} = V^j$.
  while *stopping criterion for inner loop not satisfied* **do**
    Update $W^{j,k+1}$ by Eq. (2.18).
    Update $V^{j,k+1}$ by Eq. (2.19).
    $k := k + 1$
  end
  Set $W^{j+1} = W^{j,k}$ and $V^{j+1} = V^{j,k}$.
  Set $\beta := \sigma\beta$.
  Set $j := j + 1$.
end
Output: W^j and V^j.

with mini-batches due to the large-size computation dealing with the amount of data and weight parameters that a typical DNN has.

To update V, we see that (2.17b) can be rewritten as

$$V^{k+1} = \arg\min_{V} \sum_{l=1}^{L} \left(\frac{\lambda}{\beta} r(V_l) + \frac{1}{2} \|V_l - W_l\|_2^2 \right) = \left(\mathrm{prox}_{\frac{\lambda}{\beta} r}(W_1), \ldots, \mathrm{prox}_{\frac{\lambda}{\beta} r}(W_L) \right). \tag{2.19}$$

The proximal operators for the considered regularizers are thresholding functions as their closed-form solutions, and as a result, the V update simplifies to thresholding W. The regularization functions and their corresponding proximal operators are summarized in Table 2.1.

Incorporating the algorithm that solves the quadratic penalty problem (2.16), we now develop a general algorithm to solve (2.14). We solve a sequence of quadratic penalty problems (2.16) with $\beta \in \{\beta_j\}_{j=1}^{\infty}$ where $\beta_j \uparrow \infty$. This will yield a sequence $\{(V^j, W^j)\}_{j=1}^{\infty}$ so that $W^j \uparrow W^*$, a solution to (2.14). This algorithm is based on the quadratic penalty method [168] and the penalty decomposition method [144]. The algorithm is summarized in Algorithm 1.

Table 2.1: Regularization penalties and their corresponding proximal operators with $\lambda > 0$.

Regularizer Name	Penalty Formulation	Proximal Operator
ℓ_1	$\lambda\\|x\\|_1 = \lambda \sum_{i=1}^{n} \|x_i\|$	$\mathrm{prox}_{\lambda\\|\cdot\\|_1}(x) = (\mathcal{S}_\lambda(x_1), \ldots, \mathcal{S}_\lambda(x_n))$, with $\mathcal{S}_\lambda(t) = \mathrm{sign}(t)\max\{\|t\| - \lambda, 0\}$
ℓ_0	$\lambda\\|x\\|_0 = \lambda \sum_{i=1}^{n} \|x_i\|_0$	$\mathrm{prox}_{\lambda\\|\cdot\\|_0}(x) = (\mathcal{H}_\lambda(x_1), \ldots, \mathcal{H}_\lambda(x_n))$, with $\mathcal{H}_\lambda(t) = \begin{cases} 0 & \text{if } \|t\| \leq \sqrt{2\lambda} \\ t & \text{if } \|t\| > \sqrt{2\lambda} \end{cases}$
SCAD(a)	$\lambda\\|x\\|_{\mathrm{SCAD}(a)} = \sum_{i=1}^{n} \lambda\|x_i\|_{\mathrm{SCAD}(a)}$ with $\lambda\|t\|_{\mathrm{SCAD}(a)} = \begin{cases} \lambda\|t\| & \text{if } \|t\| < \lambda \\ \frac{2a\lambda\|t\| - t^2 - \lambda^2}{2(a-1)} & \text{if } \lambda \leq \|t\| < a\lambda \\ (a+1)\lambda^2/2 & \text{if } \|t\| \geq a\lambda \end{cases}$	$\mathrm{prox}_{\lambda\\|\cdot\\|_{\mathrm{SCAD}(a)}}(x) = (\mathscr{S}_{a,\lambda}(x_1), \ldots, \mathscr{S}_{a,\lambda}(x_n))$, with $\mathscr{S}_{a,\lambda}(t) = \begin{cases} \mathcal{S}_\lambda(t) & \text{if } \|t\| \leq 2\lambda \\ \frac{(a-1)t - \mathrm{sign}(t)a\lambda}{a-2} & \text{if } 2\lambda < \|t\| \leq a\lambda \\ t & \text{if } \|t\| > a\lambda. \end{cases}$
TL1(a)	$\lambda\\|x\\|_{\mathrm{TL1}(a)} = \lambda \sum_{i=1}^{n} \frac{(a+1)\|x_i\|}{a + \|x_i\|}$	$\mathrm{prox}_{\lambda\\|\cdot\\|_{\mathrm{TL1}(a)}}(x) = (\mathcal{T}_{a,\lambda}(x_1), \ldots, \mathcal{T}_{a,\lambda}(x_n))$, with $\mathcal{T}_{a,\lambda}(t) = \begin{cases} 0 & \text{if } \|t\| \leq \tau(a,\lambda) \\ g_{a,\lambda}(t) & \text{if } \|t\| > \tau(a,\lambda) \end{cases}$ where $g_{a,\lambda}(t) = \mathrm{sign}(t)\left(\frac{2}{3}(a + \|t\|)\cos\left(\frac{\phi_{a,\lambda}(t)}{3}\right) - \frac{2a}{3} + \frac{\|t\|}{3}\right)$, $\phi_{a,\lambda}(t) = \arccos\left(1 - \frac{27\lambda a(a+1)}{2(a+\|t\|)^3}\right)$, and $\tau(a,\lambda) = \begin{cases} \sqrt{2\lambda(a+1)} - \frac{a}{2} & \text{if } \lambda > \frac{a^2}{2(a+1)} \\ \lambda\frac{a+1}{a} & \text{if } \lambda \leq \frac{a^2}{2(a+1)} \end{cases}$
$\ell_1 - \ell_2$	$\lambda\\|x\\|_{\ell_1-\ell_2} = \lambda\left(\sum_{i=1}^{n} \|x_i\| - \sqrt{\sum_{i=1}^{n} x_i^2}\right)$	$\mathrm{prox}_{\lambda\\|\cdot\\|_{\ell_1-\ell_2}}(x) = \begin{cases} \frac{\\|z_1\\|_2 + \lambda}{\\|z_1\\|_2} z_1 & \text{if } \\|x\\|_\infty > \lambda \\ z_2 & \text{if } 0 \leq \\|x\\|_\infty \leq \lambda \end{cases}$ with $z_1 = \mathcal{S}_\lambda(x)$ and $(z_2)_i = \begin{cases} 0 & \text{if } i \neq k \\ \mathrm{sign}(x_i)\\|x\\|_\infty & \text{if } i = k, \end{cases}$ where $k = \arg\min_{1 \leq k \leq n}\{\|x_i\| = \\|x\\|_\infty\}$.

An alternative algorithm to solve (2.14) is proximal gradient descent [171]. By this method, the update for $W_l, l = 1, \ldots, L$, is

$$W_l^{k+1} = \mathrm{prox}_{\gamma\lambda r}\left(W_l^k - \gamma\left(\nabla_{W_l}\tilde{\mathcal{L}}(W^k) + \lambda\partial_{W_l}\mathcal{R}_{GL}(W_l^k)\right)\right). \tag{2.20}$$

Using this algorithm results in weight parameters with some already zero'ed out.

However, the advantage of our proposed algorithm lies in (2.17a), written more specifically as

$$\begin{aligned} W_l^{k+1} &= \arg\min_{W_l} \tilde{\mathcal{L}}(W) + \mathcal{R}_{GL}(W_l) + \frac{\beta}{2}\|V_l - W_l\|_2^2 \qquad (2.21)\\ &= \arg\min_{W_l} \tilde{\mathcal{L}}(W) + \mathcal{R}_{GL}(W_l) + \frac{\beta}{2}\sum_{i=1}^{\#W_l}(v_{l,i} - w_{l,i})^2. \end{aligned}$$

We see that this step performs exact weight decay or ℓ_2 regularization on weights $w_{l,i}$ whenever $v_{l,i} = 0$. On the other hand, when $v_{l,i} \neq 0$, the effect of ℓ_2 regularization is mitigated on the corresponding weight $w_{l,i}$ based on the absolute difference $|v_{l,i} - w_{l,i}|$. Using ℓ_2 regularization was shown to give superior pruning results in terms of accuracy by Han et al. [84]. Our proposed algorithm can be perceived as an adaptive ℓ_2 regularization method, where (2.17b) identifies which weights to perform exact ℓ_2 regularization on and (2.17a) updates and regularizes the weights accordingly.

2.1.5 Convergence Analysis

To establish convergence for the proposed algorithm, the results below state that the accumulation point of the sequence generated by (2.17a)-(2.17b) is a block-coordinate minimizer, and an accumulation point generated by Algorithm 1 is a sparse feasible solution to (2.15).

Proofs are provided in Section 2.3. Unfortunately, the feasible solution generated may not be a local minimizer of (2.15) because the loss function $\mathcal{L}(\cdot,\cdot)$ is nonconvex. However, it was shown in [62] that a similar algorithm to Algorithm 1, but for fixed β in a bounded interval, generates an approximate global solution with high probability for a one-layer CNN with ReLu activation function.

Theorem 2.1. *Let* $\{(V^k, W^k)\}_{k=1}^{\infty}$ *be a sequence generated by the alternating minimization algorithm* (2.17a)-(2.17b)*, where* $r(\cdot)$ *is* ℓ_0, ℓ_1, *transformed* ℓ_1, $\ell_1-\alpha\ell_2$, *or SCAD. If* (V^*, W^*) *is an accumulation point of* $\{(V^k, W^k)\}_{k=1}^{\infty}$, *then* (V^*, W^*) *is a block-coordinate minimizer of* (2.16)*. that is*

$$\begin{aligned} V^* &\in \arg\min_V F_\beta(V, W^*) \\ W^* &\in \arg\min_W F_\beta(V^*, W). \end{aligned}$$

Theorem 2.2. *Let* $\{(V^k, W^k, \beta_k)\}_{k=1}^{\infty}$ *be a sequence generated by Algorithm 1. Suppose that* $\{F_{\beta_k}(V^k, W^k)\}_{k=1}^{\infty}$ *is uniformly bounded. If* (V^*, W^*) *is an accumulation point of* $\{V^k, W^k)\}_{k=1}^{\infty}$, *then* (V^*, W^*) *is a feasible solution to* (2.15)*, that is* $V^* = W^*$.

Remark: To safely ensure that $\{F_{\beta_k}(V^k, W^k)\}_{k=1}^{\infty}$ is uniformly bounded in practice, we can find a feasible solution $(V^{\text{feas}}, W^{\text{feas}})$ to (2.15) and impose a bound M such that

$$M \geq \max\left\{\tilde{L}(W^{\text{feas}}) + \lambda \sum_{l=1}^{L} \mathcal{R}(W_l^{\text{feas}}), \min_W F_{\beta_0}(V^1, W)\right\}.$$

If $\min_W F_{\beta_{k+1}}(V^k, W) > M$, then we set $V^{k+1} = W^{\text{feas}}$. This strategy is based on [144]. However, in our numerical experiments, we have not yet encountered $F_{\beta_k}(V^k, W^k)$ to diverge.

2.2 Numerical Experiments

2.2.1 Application to Deep Neural Networks

We compare the proposed nonconvex sparse group lasso against four other methods as baselines: group lasso, sparse group lasso (SGL_1), CGES proposed in [234], and the group variant of ℓ_0 regularization (denoted as ℓ_0 for simplicity) proposed in [142]. SGL_1 is optimized using the same algorithm proposed for nonconvex sparse group lasso. For the group terms, the weights are grouped together based on the filters or output channels, which we will refer to as neurons. We trained various CNN architectures on MNIST [116] and CIFAR 10/100 [110]. The MNIST dataset consists of 60k training images and 10k test images. MNIST is trained on two simple CNN architectures: LeNet-5-Caffe [101, 116] and a 4-layer CNN with two convolutional layers (32 and 64 channels, respectively) and an intermediate layer of 1000 fully connected neurons. CIFAR 10/100 is a dataset that has 10/100 classes split into 50k training images and 10k test images. It is trained on Resnets [89] and wide Resnets [241]. Throughout all of our experiments, for $SGSCAD(a)$, we set $a = 3.7$ as suggested in [69]; for $SGTL_1(a)$, we set $a = 1.0$ as suggested in [248]; and for $SGL_1 - L_2$, we set $\alpha = 1.0$ as suggested by the literatures [139, 230, 137, 138]. For CGES, we have $\mu_l = l/L$. Because the optimization algorithms do not drive most, if not all, the weights and neurons to zeroes, we have to set them to zeroes when their values are below a certain threshold. In our experiments, if the absolute weights are below 10^{-5}, we set them to zeroes. Then, **weight sparsity** is defined to be *the percentage of zero weights with respect to the total number of weights trained in the network*. If the normalized sum of the absolute values of the weights of the neuron is less than 10^{-5}, then the weights of the neuron are set to zeroes. **Neuron sparsity** is defined to be *the percentage of neurons whose weights are zeroes with respect to the total number of neurons in the network*.

MNIST Classification

MNIST is trained on Lenet-5-Caffe, which has four layers with 1,370 total neurons and 431,080 total weight parameters. All layers of the network are applied with strictly the same type of regularization. No other regularization methods (e.g., dropout and batch normalization) are used. The network is optimized using Adam [108] with initial learning rate 0.001. For every 40 epochs, the learning rate decays by a factor of 0.1. We set the regularization parameter to the following values: $\lambda = \alpha/60000$ for $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. For SGL_1 and nonconvex sparse group lasso, we set $\beta = 25\alpha/60000$, and for every 40 epochs, β increases by a factor of $\sigma = 1.25$. The network is trained for 200 epochs across 5 runs.

Table 2.2 reports the mean results for test error, weight sparsity, and neuron sparsity across five runs of Lenet-5-Caffe trained after 200 epochs. We see that although CGES has the lowest test errors at $\alpha \in \{0.1, 0.3, 0.4\}$ and the largest weight sparsity for all $\alpha \in \{0.1, 0.2, \ldots, 0.5\}$, nonconvex sparse group lasso's test errors and weight sparsity are comparable. Additionally, nonconvex sparse group lasso's neuron sparsity is nearly two times larger than the neuron sparsity attained by CGES. Across all parameters and methods, SGL_0 with $\alpha = 0.5$ attains the best average test error of 0.630 with average weight sparsity 95.7% and neuron sparsity 80.7%. Furthermore, its test error is lower than the test errors of other nonconvex sparse group lasso regularization methods for all α's tested. Generally, SGL_1 and nonconvex sparse group lasso outperform ℓ_0 regularization proposed by Louizos et al. [142] and group lasso by average weight and neuron sparsity.

Table 2.3 reports the mean results for test error, weight sparsity, and neuron sparsity of the Lenet-5-Caffe models with the lowest test errors from the five runs. According to the results, the best test errors are attained by SGL_0 at $\alpha = 0.3, 0.5$; $SGL_1 - L_2$ at $\alpha = 0.2$; and CGES at $\alpha = 0.1, 0.4$. For average weight sparsity, SGL_0 attains the largest weight sparsity at $\alpha \in \{0.2, 0.3, 0.4, 0.5\}$. For average neuron sparsity, the largest values are attained by

Table 2.2: Average test error, weight sparsity, and neuron sparsity of Lenet-5 models trained on MNIST after 200 epochs across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha=0.1$	0.816 (0.024)	**0.644** (0.039)	0.742 (0.030)	0.722 (0.028)	0.682 (0.044)	0.734 (0.039)	0.716 (0.048)	0.688 (0.034)
$\alpha=0.2$	0.914 (0.029)	0.718 (0.044)	0.772 (0.031)	**0.704** (0.031)	0.712 (0.042)	0.788 (0.045)	0.718 (0.025)	0.746 (0.031)
$\alpha=0.3$	1.032 (0.045)	**0.678** (0.007)	0.782 (0.035)	0.732 (0.045)	0.686 (0.048)	0.760 (0.037)	0.728 (0.034)	0.712 (0.061)
$\alpha=0.4$	1.062 (0.030)	**0.662** (0.024)	0.820 (0.054)	0.792 (0.034)	0.704 (0.033)	0.786 (0.045)	0.766 (0.045)	0.756 (0.014)
$\alpha=0.5$	1.098 (0.035)	0.696 (0.016)	0.834 (0.033)	0.720 (0.039)	**0.630** (0.024)	0.728 (0.044)	0.684 (0.024)	0.750 (0.017)
Avg. Weight Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha=0.1$	2.12×10^{-4} (1.54×10^{-5})	**0.940** (1.51×10^{-3})	0.885 (2.25×10^{-3})	0.889 (4.30×10^{-3})	0.894 (3.81×10^{-3})	0.894 (3.61×10^{-3})	0.901 $(1.57\times10^{-3}$	0.893 (2.77×10^{-3})
$\alpha=0.2$	2.16×10^{-4} (3.76×10^{-6})	**0.952** (1.51×10^{-3})	0.922 (2.07×10^{-3})	0.926 (1.19×10^{-3})	0.926 (1.75×10^{-3})	0.926 (3.31×10^{-3})	0.930 (2.37×10^{-3})	0.923 (2.86×10^{-3})
$\alpha=0.3$	2.24×10^{-4} (5.35×10^{-6})	**0.956** (1.41×10^{-3})	0.933 (1.03×10^{-3})	0.945 (1.43×10^{-3})	0.941 (1.73×10^{-3})	0.941 (2.52×10^{-3})	0.941 (1.28×10^{-3})	0.943 (1.04×10^{-3})
$\alpha=0.4$	2.06×10^{-4} (6.27×10^{-6})	**0.960** (1.05×10^{-3})	0.943 $1.63\times10^{-3})$	0.952 (1.21×10^{-3})	0.951 (1.82×10^{-3})	0.950 (1.64×10^{-3})	0.952 (1.91×10^{-3})	0.952 (1.14×10^{-3})
$\alpha=0.5$	2.27×10^{-4} (1.53×10^{-5})	**0.963** (1.85×10^{-3})	0.946 (1.43×10^{-3})	0.954 (1.63×10^{-3})	0.957 (9.21×10^{-4})	0.956 (1.37×10^{-3})	0.956 (2.00×10^{-3})	0.956 (2.43×10^{-3})
Avg. Neuron Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha=0.1$	0.531 (3.79×10^{-4})	0.387 (9.13×10^{-3})	0.696 (2.42×10^{-3})	0.691 (7.38×10^{-3})	0.682 (6.27×10^{-3})	**0.704** (3.94×10^{-3})	0.703 (5.09×10^{-3})	0.697 (3.93×10^{-3})
$\alpha=0.2$	0.578 (1.19×10^{-3})	0.449 (1.26×10^{-2})	0.756 (3.39×10^{-3})	0.754 (2.72×10^{-3})	0.740 (4.01×10^{-3})	**0.758** (5.78×10^{-3})	0.757 (3.93×10^{-3})	0.749 (6.50×10^{-3})
$\alpha=0.3$	0.602 (4.42×10^{-4})	0.476 (1.17×10^{-2})	0.776 (3.18×10^{-3})	**0.787** (2.55×10^{-3})	0.769 (4.44×10^{-3})	0.785 (4.97×10^{-3})	0.774 (4.11×10^{-3})	0.783 (3.78×10^{-3})
$\alpha=0.4$	0.616 (7.58×10^{-4})	0.518 (9.72×10^{-3})	0.795 (3.44×10^{-3})	**0.805** (3.89×10^{-3})	0.791 (5.40×10^{-3})	0.803 (3.35×10^{-3})	0.799 (3.56×10^{-3})	0.804 (2.69×10^{-3})
$\alpha=0.5$	0.626 (1.07×10^{-3})	0.539 (1.27×10^{-2})	0.799 (2.59×10^{-3})	0.811 (4.07×10^{-3})	0.807 (3.15×10^{-3})	**0.819** (2.79×10^{-3})	0.811 (6.29×10^{-3})	0.815 (6.10×10^{-3})

Table 2.3: Average test error, weight sparsity, and neuron sparsity of Lenet-5 models trained on MNIST with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.1$	0.682 (0.023)	**0.532** (0.031)	0.568 (0.026)	0.568 (0.021)	0.576 (0.027)	0.602 (0.027)	0.582 (0.028)	0.554 (0.056)
$\alpha = 0.2$	0.846 (0.033)	0.584 (0.038)	0.630 (0.017)	0.582 (0.035)	0.584 (0.049)	0.616 (0.021)	0.592 (0.026)	**0.578** (0.032)
$\alpha = 0.3$	0.980 (0.033)	0.590 (0.028)	0.642 (0.013)	0.600 (0.030)	**0.588** (0.019)	0.618 (0.037)	0.594 (0.022)	0.596 (0.039)
$\alpha = 0.4$	1.014 (0.019)	**0.562** (0.015)	0.680 (0.038)	0.652 (0.025)	0.604 (0.033)	0.630 (0.035)	0.630 (0.048)	0.628 (0.020)
$\alpha = 0.5$	1.066 (0.024)	0.598 (0.027)	0.682 (0.043)	0.616 (0.052)	**0.572** (0.012)	0.654 (0.015)	0.586 (0.034)	0.670 (0.026)
Avg. Weight Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.1$	2.38×10^{-4} (1.97×10^{-5})	0.541 (0.024)	0.661 (0.073)	0.757 (0.015)	0.768 (0.019)	0.680 (0.167)	**0.773** (7.48×10^{-3})	0.719 (0.066)
$\alpha = 0.2$	2.26×10^{-4} (9.43×10^{-6})	0.583 (0.017)	0.728 (0.170)	0.845 (4.79×10^{-3})	**0.857** (6.15×10^{-3})	0.821 (0.041)	0.854 (5.60×10^{-3})	0.836 (6.76×10^{-3})
$\alpha = 0.3$	2.19×10^{-4} (1.36×10^{-5})	0.603 (0.020)	0.810 (0.078)	0.886 (3.69×10^{-3})	**0.889** (3.62×10^{-3})	0.878 (9.43×10^{-4})	0.827 (0.115)	0.879 (3.97×10^{-3})
$\alpha = 0.4$	2.22×10^{-4} (1.47×10^{-5})	0.627 (0.019)	0.845 (0.040)	0.896 (3.57×10^{-3})	**0.905** (3.66×10^{-3})	0.846 (0.097)	0.899 (4.23×10^{-3})	0.852 (0.097)
$\alpha = 0.5$	2.24×10^{-4} (1.02×10^{-5})	0.633 (0.013)	0.886 (6.40×10^{-3})	0.905 (2.87×10^{-3})	**0.922** (0.015)	0.902 (2.64×10^{-3})	0.871 (0.084)	0.848 (0.080)
Avg. Neuron Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.1$	0.363 (0.047)	0.315 (0.030)	0.389 (0.120)	0.497 (0.014)	0.496 (0.030)	0.426 (0.172)	**0.513** (9.57×10^{-3})	0.440 (0.107)
$\alpha = 0.2$	0.574 (2.22×10^{-3})	0.392 (0.016)	0.498 (0.185)	0.627 (0.011)	0.631 (0.012)	0.549 (0.169)	**0.634** (9.30×10^{-3})	0.608 (0.015)
$\alpha = 0.3$	0.599 (2.61×10^{-3})	0.418 (0.021)	0.570 (0.154)	**0.697** (9.73×10^{-3})	0.692 (8.19×10^{-3})	0.684 (5.69×10^{-3})	0.613 (0.154)	0.686 (8.60×10^{-3})
$\alpha = 0.4$	0.614 (1.71×10^{-3})	0.482 (0.020)	0.586 (0.184)	0.721 (8.16×10^{-3})	**0.725** (9.97×10^{-3})	0.642 (0.151)	0.724 (0.015)	0.655 (0.150)
$\alpha = 0.5$	0.625 (1.55×10^{-3})	0.492 (0.024)	0.708 (8.94×10^{-3})	0.735 (3.73×10^{-3})	**0.759** (0.020)	0.733 (8.59×10^{-3})	0.683 (0.143)	0.570 (0.216)

Table 2.4: Average test error, weight sparsity, and neuron sparsity of 4-layer CNN models trained on MNIST after 200 epochs across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.2$	0.962 (0.041)	0.470 (0.036)	0.486 (0.030)	0.418 (0.010)	0.432 (0.023)	**0.408** (0.013)	0.418 (0.026)	0.436 (0.012)
$\alpha = 0.4$	1.454 (0.070)	0.486 (0.030)	0.502 (0.035)	**0.436** (0.026)	0.49 (0.017)	0.456 (0.016)	0.47 (0.035)	0.446 (0.031)
$\alpha = 0.6$	2.396 (0.066)	0.512 (0.035)	0.510 (0.028)	0.494 (0.031)	0.500 (0.023)	**0.488** (0.019)	0.498 (0.025)	0.522 (0.019)
$\alpha = 0.8$	3.396 (0.096)	**0.502** (0.020)	0.544 (0.026)	0.542 (0.025)	0.536 (0.037)	0.524 (0.015)	0.536 (0.014)	0.524 (0.015)
$\alpha = 1.0$	4.74 (0.148)	**0.524** (0.26)	0.568 (0.004)	0.566 (0.041)	0.576 (0.014)	0.544 (0.024)	0.552 (0.017)	0.556 (0.022)
Avg. Weight Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.2$	5.99×10^{-5} (9.28×10^{-6})	**0.655** (4.10×10^{-3})	0.284 (6.47×10^{-3})	0.302 (6.68×10^{-3})	0.306 (0.014)	0.297 (5.42×10^{-3})	0.298 (8.63×10^{-3}	0.299 (7.74×10^{-3})
$\alpha = 0.4$	5.84×10^{-5} (7.95×10^{-6})	**0.710** (2.45×10^{-3})	0.489 (7.38×10^{-3})	0.510 (1.85×10^{-3})	0.502 (8.01×10^{-3})	0.507 (8.80×10^{-3})	0.510 (0.011)	0.505 (7.25×10^{-3})
$\alpha = 0.6$	6.06×10^{-5} (1.22×10^{-5})	**0.737** (2.13×10^{-3})	0.593 (5.67×10^{-3})	0.606 (5.41×10^{-3})	0.603 (7.61×10^{-3})	0.605 (5.46×10^{-3})	0.599 (0.012)	0.609 (6.96×10^{-3})
$\alpha = 0.8$	7.18×10^{-5} (6.24×10^{-6})	**0.755** (5.67×10^{-3})	0.661 (6.11×10^{-3})	0.660 (6.42×10^{-3})	0.663 (7.30×10^{-3})	0.661 (8.74×10^{-3})	0.665 (3.95×10^{-3})	0.661 (5.72×10^{-3})
$\alpha = 1.0$	6.90×10^{-5} (7.33×10^{-6})	**0.767** (2.92×10^{-3})	0.695 (5.08×10^{-3})	0.696 (4.68×10^{-3})	0.697 (2.38×10^{-4})	0.698 (6.51×10^{-3})	0.699 (4.27×10^{-3})	0.689 (9.47×10^{-3})
Avg. Neuron Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.2$	**0.472** (7.10×10^{-4})	0.299 (2.40×10^{-3})	0.153 (4.06×10^{-3})	0.160 (4.54×10^{-3})	0.164 (8.58×10^{-3})	0.158 (3.68×10^{-3})	0.158 (5.20×10^{-3})	0.159 (5.87×10^{-3})
$\alpha = 0.4$	**0.494** (1.01×10^{-3})	0.329 (2.10×10^{-3})	0.280 (5.64×10^{-3})	0.287 (7.55×10^{-4})	0.280 (6.57×10^{-3})	0.281 (5.05×10^{-3})	0.285 (8.48×10^{-3})	0.284 (7.22×10^{-3})
$\alpha = 0.6$	**0.506** (7.23×10^{-4})	0.343 (1.78×10^{-3})	0.351 (4.72×10^{-3})	0.354 (2.47×10^{-3})	0.35 (7.17×10^{-3})	0.352 (3.99×10^{-3})	0.347 (9.65×10^{-3})	0.353 (5.88×10^{-3})
$\alpha = 0.8$	**0.516** (6.72×10^{-4})	0.355 (8.23×10^{-3})	0.404 (6.20×10^{-3})	0.391 (4.66×10^{-3})	0.396 (7.60×10^{-3})	0.395 (9.59×10^{-3})	0.399 (3.89×10^{-3})	0.398 (6.39×10^{-3})
$\alpha = 1.0$	**0.526** (9.45×10^{-4})	0.361 (5.36×10^{-3})	0.432 (5.02×10^{-3})	0.424 (5.62×10^{-3})	0.427 (2.64×10^{-3})	0.427 (7.36×10^{-3})	0.430 (6.37×10^{-3})	0.417 (0.011)

Table 2.5: Average test error, weight sparsity, and neuron sparsity of 4-layer CNN models trained on MNIST with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.2$	0.916 (0.010)	0.452 (0.033)	0.440 (0.021)	**0.384** (0.015)	0.404 (0.019)	**0.384** (0.020)	0.392 (0.023)	0.398 (0.015)
$\alpha = 0.4$	1.414 (0.073)	0.448 (0.012)	0.456 (0.024)	0.414 (0.021)	0.426 (0.016)	0.426 (0.017)	0.428 (0.034)	**0.412** (0.012)
$\alpha = 0.6$	1.890 (0.033)	0.464 (0.022)	0.472 (0.013)	**0.434** (0.010)	0.460 (0.026)	0.440 (0.017)	0.452 (0.016)	0.454 (0.024)
$\alpha = 0.8$	1.966 (0.010)	**0.478** (0.007)	0.506 (0.014)	0.484 (0.019)	0.504 (0.015)	0.482 (0.019)	0.488 (0.016)	0.492 (0.007)
$\alpha = 1.0$	2.046 (0.019)	**0.492** (0.024)	0.530 (0.014)	0.514 (0.026)	0.520 (0.035)	0.506 (0.019)	0.514 (0.014)	**0.492** (0.016)
Avg. Weight Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.2$	5.86×10^{-5} (4.32×10^{-6})	**0.384** (0.112)	0.201 (0.005)	0.248 (0.012)	0.249 (0.017)	0.254 (0.013)	0.250 (0.013)	0.244 (0.006)
$\alpha = 0.4$	6.45×10^{-5} (9.15×10^{-6})	**0.541** (0.155)	0.424 (0.006)	0.467 (0.007)	0.449 (0.012)	0.466 (0.011)	0.460 0.020)	0.468 (0.015)
$\alpha = 0.6$	1.41×10^{-4} (1.74×10^{-5})	0.502 (0.157)	0.541 (0.010)	0.563 (0.016)	0.563 (0.016)	**0.568** (0.011)	0.559 (0.015)	0.565 (0.008)
$\alpha = 0.8$	1.39×10^{-4} (1.06×10^{-6})	0.576 (0.166)	0.619 (0.012)	0.620 (0.012)	0.625 (0.014)	0.624 (0.014)	**0.628** (0.007)	0.626 (0.012)
$\alpha = 1.0$	1.47×10^{-4} (7.84×10^{-6})	0.518 (0.169)	0.658 (0.010)	0.661 (0.007)	0.658 (0.007)	**0.664** (0.006)	0.659 (0.007)	0.653 (0.008)
Avg. Neuron Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.2$	**0.470** (5.97×10^{-4})	0.293 (2.61×10^{-3})	0.099 (3.77×10^{-3})	0.122 (7.25×10^{-3})	0.123 (9.71×10^{-3})	0.126 (8.39×10^{-3})	0.123 (7.86×10^{-3})	0.120 (4.93×10^{-3})
$\alpha = 0.4$	**0.494** (6.51×10^{-4})	0.328 (1.43×10^{-3})	0.224 (4.23×10^{-3})	0.243 (6.85×10^{-3})	0.231 (0.011)	0.241 (3.74×10^{-3})	0.238 (0.015)	0.249 (0.014)
$\alpha = 0.6$	0.198 (6.25×10^{-5})	**0.343** (4.82×10^{-3})	0.296 (9.94×10^{-3})	0.305 (0.013)	0.307 (0.014)	0.311 (6.32×10^{-3})	0.303 (0.010)	0.306 (9.24×10^{-3})
$\alpha = 0.8$	0.217 (2.03×10^{-5})	0.353 (3.37×10^{-3})	0.357 (0.012)	0.343 (0.015)	0.350 (0.011)	0.348 (0.013)	0.356 (4.78×10^{-3})	**0.358** (0.016)
$\alpha = 1.0$	0.229 (3.98×10^{-5})	0.359 (2.78×10^{-3})	**0.387** (0.010)	0.379 (3.75×10^{-3})	0.382 (5.85×10^{-3})	0.385 (6.37×10^{-3})	0.383 (4.66×10^{-3})	0.373 (9.97×10^{-3})

$SGTL_1$ at $\alpha = 0.1, 0.2$; by SGL_1 at $\alpha = 0.3$; and by SGL_0 at $\alpha = 0.4, 0.5$. Although SGL_0 does not outperform all the other methods across the board, its results are still comparable to the best results. Overall, we see that nonconvex sparse group lasso outperforms ℓ_0 in test error, weight sparsity, and neuron sparsity and group lasso in weight and neuron sparsity.

MNIST is also trained on a 4-layer CNN with two convolutional layers with 32 and 64 channels, respectively, and an intermediate layer with 1000 neurons. Each convolutional layer has a 5×5 convolutional filters. The 4-layer CNN has 2,120 total neurons and 1,087,010 total weight parameters. All layers of the network are applied with strictly the same type of regularization. The network is optimized with the same settings as Lenet-5-Caffe. However, the regularization parameter is different: we have $\lambda = \alpha/60000$ for $\alpha \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. For SGL_1 and nonconvex sparse group lasso, we set $\beta = 5\alpha/60000$ and for every 40 epochs, β increases by a factor of $\sigma = 1.25$. The network is trained for 200 epochs across 5 runs.

Table 2.4 reports the mean results for test error, weight sparsity, and neuron sparsity across five runs of the 4-layer CNN models trained after 200 epochs. Although CGES consistently has the highest weight sparsity, it does not yield the most accurate models until when $\alpha \geq 0.8$. Moreover, its neuron sparsity is smaller than the neuron sparsity by group lasso, SGL_1, and nonconvex group lasso when $\alpha \geq 0.6$. ℓ_0 has the highest neuron sparsity for all α's given, but its test errors are much greater. When $\alpha \leq 0.6$, $SGSCAD$ yields the most accurate models at $\alpha = 0.2, 0.6$ while SGL_1 yields one at $\alpha = 0.4$. Overall, we see that nonconvex group lasso has comparable weight sparsity and neuron sparsity as group lasso and SGL_1.

Table 2.5 reports the mean results for test error, weight sparsity, and neuron sparsity of the 4-layer CNN models with the lowest test errors from the five runs. At $\alpha = 0.2$, SGL_1 and $SGSCAD$ have the lowest test errors, but their weight sparsity are exceeded by CGES and their neuron sparsity are exceed by ℓ_0. At $\alpha = 0.4$, $SGL_1 - L_2$ has the lowest test error, but its weight sparsity and neuron sparsity are exceeded by CGES and ℓ_0, respectively. At $\alpha = 0.6$, SGL_1 has the lowest test error, but $SGSCAD$ has the largest weight sparsity with comparable test error. At $\alpha \geq 0.8$, CGES has the lowest test error, but its weight sparsity is exceeded by group lasso, SGL_1, and the nonconvex group lasso regularizers, which all have slightly higher test error. At $\alpha = 0.8$, the neuron sparsity of CGES is comparable to the neuron sparsity of group lasso, SGL_1, and the nonconvex group lasso regularizers. At $\alpha = 1.0$, group lasso has the highest neuron sparsity, but nonconvex group lasso has slightly lower neuron sparsity. In general, weight sparsity of nonconvex group lasso is comparable to or larger than the weight sparsity of group lasso and SGL_1.

CIFAR Classification

CIFAR 10/100 is trained on Resnet-40 and wide Resnet with depth 28 and width 10 (WRN-28-10). Resnet-40 has approximately 570,000 weight parameters and 1520 neurons while WRN-28-10 has approximately 36,500,000 weight parameters and 10,736 neurons. The networks are optimized using stochastic gradient descent with initial learning rate 0.1. After every 60 epochs, learning rate decays by a factor of 0.2. Strictly the same type of regularization is applied to the weights of the hidden layer where dropout is utilized in the residual block. We vary the regularization parameter $\lambda = \alpha/50000$. For Resnet-40, we have $\alpha \in \{1.0, 1.5, 2.0, 2.5, 3.0\}$ for CIFAR 10 and $\alpha \in \{2.0, 2.5, 3.0, 3.5, 4.0\}$ for CIFAR 100. For SGL_1 and nonconvex sparse group lasso, we set $\beta = 15\alpha/50000$ for Resnet-40 and $\beta = 25\alpha/50000$ for WRN-28-10. For every 20 epochs, β increases by a factor of $\sigma = 1.25$. The networks are trained for 200 epochs across 5 runs. We excluded ℓ_0 regularization by Louizos *et al.* [142] because it was unstable for the provided α's. Furthermore, we only analyze the models with the lowest test errors since the test errors did not stabilize by the end of the 200 epochs in our experiments.

Table 2.6 reports mean test error, weight sparsity, and neuron sparsity across the Resnet-40 models trained on CIFAR 10 with the lowest test errors from the five runs. Group lasso has the lowest test errors for all α's provided while CGES, SGL_1, and nonconvex sparse group

Table 2.6: Average test error, weight sparsity, and neuron sparsity of Resnet-40 models trained on CIFAR 10 with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 1.0$	6.932 (0.154)	**6.154** (0.199)	6.442 (0.065)	6.456 (0.176)	6.618 (0.128)	6.500 (0.158)	6.512 (0.126)
$\alpha = 1.5$	7.248 (0.145)	**6.504** (0.122)	6.850 (0.078)	7.108 (0.084)	6.948 (0.124)	6.958 (0.158)	6.820 (0.177)
$\alpha = 2.0$	7.306 (0.206)	**6.860** (0.174)	7.494 (0.092)	7.642 (0.176)	7.450 (0.192)	7.388 (0.140)	7.384 (0.122)
$\alpha = 2.5$	7.590 (0.148)	**7.298** (0.105)	7.760 (0.079)	8.146 (0.178)	8.026 (0.196)	8.096 (0.137)	7.968 (0.190)
$\alpha = 3.0$	7.672 (0.082)	**7.542** (0.135)	8.424 (0.081)	8.740 (0.166)	8.426 (0.192)	8.624 (0.083)	8.598 (0.144)
Avg. Weight Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 1.0$	**0.350** (0.009)	0.201 (0.018)	0.189 (0.007)	0.191 (0.008)	0.213 (0.015)	0.205 (0.015)	0.224 (0.016)
$\alpha = 1.5$	**0.371** (0.012)	0.322 (0.008)	0.345 (0.013)	0.313 (0.008)	0.354 (0.029)	0.330 (0.020)	0.343 (0.008)
$\alpha = 2.0$	0.385 (0.009)	0.431 (0.013)	0.457 (0.012)	0.422 (0.014)	**0.466** (0.015)	0.428 (0.013)	0.451 (0.012)
$\alpha = 2.5$	0.386 (0.010)	0.509 (0.017)	0.525 (0.010)	0.507 (0.011)	**0.534** (0.012)	0.522 (0.026)	0.537 (0.013)
$\alpha = 3.0$	0.401 (0.008)	0.551 (0.015)	**0.594** (0.009)	0.568 (0.009)	0.598 (0.012)	0.569 (0.014)	0.585 (0.006)
Avg. Neuron Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 1.0$	0.035 (0.003)	0.096 (0.011)	0.087 (0.004)	0.082 (0.005)	0.102 (0.008)	0.093 (0.010)	**0.105** (0.012)
$\alpha = 1.5$	0.040 (0.006)	0.154 (0.006)	0.159 (0.008)	0.144 (0.009)	**0.168** (0.013)	0.151 (0.009)	0.155 (0.004)
$\alpha = 2.0$	0.048 (0.004)	0.207 (0.005)	0.203 (0.008)	0.188 (0.006)	**0.217** (0.015)	0.195 (0.009)	0.209 (0.009)
$\alpha = 2.5$	0.045 (0.005)	**0.247** (0.010)	0.232 (0.010)	0.225 (0.017)	0.245 (0.011)	0.233 (0.008)	0.244 (0.006)
$\alpha = 3.0$	0.048 (0.007)	**0.274** (0.012)	0.271 (0.008)	0.249 (0.004)	0.272 (0.016)	0.259 (0.008)	0.268 (0.011)

Table 2.7: Average test error, weight sparsity, and neuron sparsity of Resnet-40 models trained on CIFAR 100 with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 2.0$	30.102 (0.234)	**28.636** (0.140)	29.260 (0.306)	29.610 (0.275)	29.044 (0.155)	29.316 (0.154)	29.274 (0.249)
$\alpha = 2.5$	30.326 (0.272)	**29.322** (0.144)	30.140 (0.180)	30.454 (0.295)	30.180 (0.175)	30.426 (0.253)	30.204 (0.159)
$\alpha = 3.0$	30.378 (0.154)	**29.750** (0.258)	31.134 (0.099)	31.482 (0.361)	31.048 (0.118)	31.164 (0.236)	31.108 (0.129)
$\alpha = 3.5$	30.666 (0.267)	**30.588** (0.285)	31.966 (0.260)	32.438 (0.272)	31.930 (0.156)	31.984 (0.182)	31.822 (0.365)
$\alpha = 4.0$	**30.982** (0.277)	31.436 (0.069)	33.106 (0.281)	33.210 (0.230)	32.758 (0.279)	33.240 (0.171)	33.094 (0.219)

Avg. Weight Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 2.0$	**0.286** (0.002)	0.129 (0.024)	0.182 (0.018)	0.164 (0.010)	0.198 (0.012)	0.162 (0.017)	0.187 (0.015)
$\alpha = 2.5$	**0.299** (0.005)	0.233 (0.010)	0.283 (0.005)	0.251 (0.021)	0.292 (0.010)	0.271 (0.015)	0.284 (0.016)
$\alpha = 3.0$	0.303 (0.003)	0.321 (0.008)	0.365 (0.009)	0.355 (0.018)	**0.377** (0.012)	0.363 (0.023)	0.372 (0.010)
$\alpha = 3.5$	0.306 (0.004)	0.409 (0.013)	0.441 (0.014)	0.418 (0.012)	**0.444** (0.014)	0.418 (0.016)	0.442 (0.006)
$\alpha = 4.0$	0.313 (0.010)	0.456 (0.014)	**0.511** (0.015)	0.461 (0.011)	0.501 (0.013)	0.480 (0.017)	0.507 (0.012)

Avg. Neuron Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 2.0$	0.001 (0.001)	0.054 (0.007)	0.074 (0.007)	0.064 (0.008)	**0.083** (0.005)	0.063 (0.004)	0.078 (0.007)
$\alpha = 2.5$	0.003 (0.001)	0.092 (0.005)	0.113 (0.004)	0.093 (0.010)	**0.116** (0.005)	0.103 (0.004)	0.111 (0.005)
$\alpha = 3.0$	0.004 (0.001)	0.126 (0.004)	0.140 (0.005)	0.133 (0.007)	0.145 (0.003)	0.138 (0.009)	**0.146** (0.003)
$\alpha = 3.5$	0.002 (0.001)	0.157 (0.006)	0.166 (0.005)	0.158 (0.005)	**0.182** (0.017)	0.156 (0.004)	0.171 (0.005)
$\alpha = 4.0$	0.005 (0.002)	0.177 (0.007)	**0.195** (0.005)	0.176 (0.007)	0.193 (0.004)	0.180 (0.011)	0.193 (0.004)

Table 2.8: Average test error, weight sparsity, and neuron sparsity of WRN-28-10 models trained on CIFAR 10 with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.01$	**3.822** (0.054)	4.092 (0.159)	4.050 (0.058)	4.036 (0.074)	4.004 (0.104)	3.994 (0.039)	4.152 (0.089)
$\alpha = 0.05$	3.856 (0.089)	3.946 (0.106)	3.874 (0.029)	3.838 (0.067)	3.862 (0.076)	**3.812** (0.097)	3.872 (0.110)
$\alpha = 0.1$	4.000 (0.076)	3.960 (0.062)	**3.784** (0.082)	3.824 (0.088)	3.832 (0.047)	3.800 (0.082)	3.792 (0.113)
$\alpha = 0.2$	4.146 (0.092)	3.928 (0.115)	3.824 (0.034)	3.874 (0.093)	3.780 (0.096)	**3.764** (0.129)	3.962 (0.078)
$\alpha = 0.5$	4.524 (0.090)	4.486 (0.077)	4.444 (0.086)	4.408 (0.063)	4.448 (0.084)	**4.340** (0.115)	4.382 (0.068)
Avg. Weight Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.01$	**0.362** (0.016)	0.045 (0.001)	0.040 (0.002)	0.044 (0.002)	0.039 (0.002)	0.040 (0.001)	0.043 (0.001)
$\alpha = 0.05$	**0.464** (0.003)	0.117 (0.003)	0.145 (0.006)	0.156 (0.005)	0.145 (0.007)	0.145 (0.004)	0.161 (0.006)
$\alpha = 0.1$	**0.483** (0.003)	0.417 (0.005)	0.438 (0.004)	0.450 (0.005)	0.441 (0.005)	0.428 (0.004)	0.446 (0.013)
$\alpha = 0.2$	0.495 (0.003)	0.673 (0.002)	0.669 (0.005)	0.672 (0.003)	0.679 (0.003)	0.666 (0.004)	**0.688** (0.003)
$\alpha = 0.5$	0.503 (0.003)	**0.868** (0.001)	0.864 (0.002)	0.857 (0.001)	0.865 (0.001)	0.858 (0.002)	0.867 (0.001)
Avg. Neuron Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.01$	**0.033** (0.002)	0.018 (0.001)	0.015 (0.001)	0.018 (0.001)	0.014 (0.001)	0.015 (0.001)	0.017 (0.001)
$\alpha = 0.02$	0.050 (0.002)	0.056 (0.001)	0.068 (0.003)	0.074 (0.003)	0.069 (0.004)	0.069 (0.003)	**0.077** (0.002)
$\alpha = 0.1$	0.055 (0.002)	0.178 (0.002)	0.189 (0.002)	0.190 (0.002)	0.188 (0.002)	0.182 (0.003)	**0.191** (0.006)
$\alpha = 0.2$	0.059 (0.001)	0.297 (0.002)	0.294 (0.005)	0.293 (0.001)	0.299 (0.001)	0.289 (0.002)	**0.307** (0.003)
$\alpha = 0.5$	0.061 (0.001)	**0.440** (0.002)	0.434 (0.002)	0.428 (0.001)	0.435 (0.001)	0.429 (0.003)	0.436 (0.001)

Table 2.9: Average test error, weight sparsity, and neuron sparsity of WRN-28-10 models trained on CIFAR 100 with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha=0.01$	**18.696** (0.184)	19.792 (0.084)	19.494 (0.241)	19.498 (0.189)	19.368 (0.188)	19.474 (0.051)	19.632 (0.182)
$\alpha=0.05$	**18.714** (0.203)	19.284 (0.134)	18.816 (0.141)	19.106 (0.277)	18.936 (0.085)	18.846 (0.082)	19.094 (0.272)
$\alpha=0.1$	19.120 (0.387)	19.168 (0.067)	18.648 (0.268)	18.690 (0.181)	**18.446** (0.108)	18.680 (0.292)	18.724 (0.084)
$\alpha=0.2$	20.298 (0.078)	18.902 (0.130)	18.440 (0.115)	18.694 (0.150)	18.502 (0.108)	**18.290** (0.107)	18.614 (0.326)
$\alpha=0.5$	21.370 (0.259)	19.604 (0.107)	19.648 (0.203)	19.732 (0.147)	**19.488** (0.262)	19.552 (0.186)	19.732 (0.156)
Avg. Weight Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha=0.01$	**0.281** (0.017)	0.013 (0.001)	0.011 (0.001)	0.013 (<0.001)	0.011 (0.001)	0.011 (0.001)	0.013 (0.001)
$\alpha=0.05$	**0.412** (0.004)	0.014 (0.001)	0.015 (0.002)	0.017 (0.001)	0.014 (0.001)	0.015 (0.001)	0.018 (0.001)
$\alpha=0.1$	**0.440** (0.013)	0.054 (0.002)	0.070 (0.003)	0.069 (0.001)	0.073 (0.002)	0.066 (0.002)	0.080 (0.001)
$\alpha=0.2$	**0.458** (0.016)	0.332 (0.004)	0.356 (0.005)	0.346 (0.002)	0.355 (0.004)	0.345 (0.003)	0.361 (0.003)
$\alpha=0.5$	0.478 (0.003)	0.697 (0.001)	0.693 (0.004)	0.685 (0.002)	**0.700** (0.002)	0.686 (0.001)	0.698 (0.002)
Avg. Neuron Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha=0.01$	**0.008** (0.001)	0.002 (<0.001)	0.002 (<0.001)	0.003 (<0.001)	0.001 (<0.001)	0.002 (<0.001)	0.002 (<0.001)
$\alpha=0.02$	**0.030** (0.001)	0.003 (<0.001)	0.005 (0.001)	0.006 (<0.001)	0.005 (0.001)	0.005 (0.001)	0.006 (<0.001)
$\alpha=0.1$	0.037 (0.001)	0.033 (0.001)	0.044 (0.002)	0.041 (<0.001)	0.046 (0.001)	0.040 (0.001)	**0.050** (0.001)
$\alpha=0.2$	0.043 (0.003)	0.153 (0.002)	0.157 (0.002)	0.150 (0.001)	0.157 (0.002)	0.148 (0.001)	**0.160** (0.001)
$\alpha=0.5$	0.052 (0.001)	0.303 (0.001)	0.298 (0.001)	0.294 (0.004)	**0.304** (0.002)	0.293 (0.002)	0.303 (0.001)

lasso are higher by at most 1.1%. When $\alpha \leq 1.5$, CGES has the largest weight sparsity while $SGSCAD$, $SGTL_1$ $SGL_1 - SGL_2$ have larger weight sparsity than does group lasso. At $\alpha = 2.0, 2.5$, $SGSCAD$ has the largest weight sparsity. At $\alpha = 3.0$, SGL_1 has the largest weight sparsity with comparable test error as the nonconvex group lasso regularizers. For neuron sparsity, $SGL_1 - L_2$ has the largest at $\alpha = 1.0$ while $SGSCAD$ has the largest at $\alpha = 1.5, 2.0$. However, at $\alpha = 2.5, 3.0$, group lasso has the largest neuron sparsity. For all α's tested, $SGSCAD$ has higher weight sparsity and neuron sparsity than does SGL_1 but with comparable test error.

Table 2.7 reports mean test error, weight sparsity, and neuron sparsity across the Resnet-40 models trained on CIFAR 100 with the lowest test errors from the five runs. Group lasso has the lowest test errors for $\alpha \leq 3.5$ while CGES has the lowest test error at $\alpha = 4.0$. However, the weight sparsity and the neuron sparsity of group lasso are lower than the sparsity of SGL_1 and some of the nonconvex sparse group lasso regularizers. CGES has the lowest neuron sparsity across all α's. Among the nonconvex group lasso penalties, $SGSCAD$ has the best test errors, which are lower than the test errors of SGL_1 for all α's except 2.5.

Table 2.8 reports mean test error, weight sparsity, and neuron sparsity across the WRN-28-10 models trained on CIFAR 10 with the lowest test errors from the five runs. The best test errors are attained by $SGTL_1$ at $\alpha = 0.05, 0.2, 0.5$; by CGES at $\alpha = 0.01$; and by SGL_1 at $\alpha = 0.1$. Weight sparsity of CGES outperforms the other methods only when $\alpha = 0.01, 0.05, 0.1$, but it underperforms when $\alpha \geq 0.2$. Weight sparsity levels between group lasso and nonconvex group lasso are comparable across all α. For neuron sparsity, $SGL_1 - L_2$ attains the largest values at $\alpha = 0.02, 0.1, 0.2$. Nevertheless, the other nonconvex sparse group lasso methods have comparable neuron sparsity. Overall, SGL_1, SGL_0, $SGSCAD$, and $SGTL_1$ outperform group lasso in test error while having similar or higher weight and neuron sparsity.

Table 2.9 reports mean test error, weight sparsity, and neuron sparsity across the WRN-28-

10 models trained on CIFAR 100 with the lowest test errors from the five runs. According to the results, the best test errors are attained by CGES when $\alpha = 0.01, 0.05$; by $SGSCAD$ when $\alpha = 0.1, 0.5$; and by $SGTL_1$ when $\alpha = 0.2$. Although CGES has the largest weight sparsity for $\alpha = 0.01, 0.05, 0.1, 0.2$, we see that its test error increases as α increases. When $\alpha = 0.5$, the best weight sparsity is attained by $SGSCAD$, but the other methods have comparable weight sparsity. The best neuron sparsity is attained by CGES at $\alpha = 0.01, 0.02$; by $SGL_1 - L_2$ at $\alpha = 0.1, 0.2$; and by $SGSCAD$ at $\alpha = 0.5$. The neuron sparsity among the nonconvex sparse group lasso methods are comparable. For $\alpha \leq 0.2$, we see that SGL_1 and nonconvex sparse group lasso outperform group lasso in test error across α while having comparable weight and neuron sparsity.

2.2.2 Algorithm Comparison

We compare the proposed Algorithm 1 with direct stochastic gradient descent, where the gradient of the regularizer is approximated by backpropagation, and proximal gradient descent, discussed in Section 2.1.4, by applying them to SGL_1 on Lenet-5 trained on MNIST. The parameter setting for this CNN is discussed in Section 2.2.1. Table 2.10 reports the mean results for test error, weight sparsity, and neuron sparsity across five models trained after 200 epochs while Figure 2.2 provides visualizations. Table 2.11 and Figure 2.3 record mean statistics for models with the lowest test errors from the five runs. According to the results, proximal stochastic gradient descent attains the highest level of weight sparsity and neuron sparsity for models trained after 200 epochs and models with the lowest test error. However, their test errors are the highest amongst the three algorithms. On the other hand, our proposed algorithm attains the lowest test errors. For models trained after 200 epochs, the weight sparsity and neuron sparsity attained by Algorithm 1 are comparable to the sparsity attained by direct stochastic gradient descent. For models with the lowest test errors generated from their respective runs, the weight sparsity and neuron sparsity by the proposed

Table 2.10: Average test error, weight sparsity, and neuron sparsity of SGL_1-regularized Lenet-5 models trained on MNIST after 200 epochs across 5 runs. The models are trained with different algorithms. Standard deviations are in parentheses. (SGD is stochastic gradient descent.)

Avg. Test Error (%)	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.758 (0.029)	1.306 (0.031)	**0.722** (0.028)
$\alpha = 0.2$	0.760 (0.006)	2.954 (0.051)	**0.704** (0.031)
$\alpha = 0.3$	0.798 (0.023)	4.992 (0.161)	**0.732** (0.045)
$\alpha = 0.4$	0.836 (0.034)	7.304 (0.147)	**0.792** (0.034)
$\alpha = 0.5$	0.772 (0.019)	9.610 (0.170)	**0.720** (0.039)
Avg. Weight Sparsity	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.935 (0.001)	**0.994** (<0.001)	0.889 (0.004)
$\alpha = 0.2$	0.951 (0.002)	**0.997** (<0.001)	0.926 (0.001)
$\alpha = 0.3$	0.960 (<0.001)	**0.998** (<0.001)	0.945 (0.001)
$\alpha = 0.4$	0.963 (0.001)	**0.998** (<0.001)	0.952 (0.001)
$\alpha = 0.5$	0.966 (0.001)	**0.998** (<0.001)	0.954 (0.002)
Avg. Neuron Sparsity	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.735 (0.003)	**0.784** (0.004)	0.691 (0.007)
$\alpha = 0.2$	0.778 (0.004)	**0.902** (0.005)	0.754 (0.003)
$\alpha = 0.3$	0.802 (0.001)	**0.960** (0.002)	0.787 (0.003)
$\alpha = 0.4$	0.813 (0.003)	**0.972** (0.001)	0.805 (0.004)
$\alpha = 0.5$	0.821 (0.004)	**0.976** (0.002)	0.811 (0.004)

Table 2.11: Average test error, weight sparsity, and neuron sparsity of SGL_1-regularized Lenet-5 models trained on MNIST with lowest test errors across 5 runs. The models are trained with different algorithms. Standard deviations are in parentheses. (SGD is stochastic gradient descent.)

Avg. Test Error (%)	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.594 (0.032)	1.152 (0.026)	**0.568** (0.021)
$\alpha = 0.2$	0.634 (0.031)	2.320 (0.042)	**0.582** (0.035)
$\alpha = 0.3$	0.692 (0.028)	3.360 (0.075)	**0.600** (0.030)
$\alpha = 0.4$	0.684 (0.014)	4.272 (0.051)	**0.652** (0.025)
$\alpha = 0.5$	0.636 (0.022)	5.020 (0.094)	**0.616** (0.052)
Avg. Weight Sparsity	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.449 (0.172)	**0.939** (0.011)	0.757 (0.015)
$\alpha = 0.2$	0.531 (0.012)	**0.971** (0.005)	0.845 (0.005)
$\alpha = 0.3$	0.451 (0.217)	**0.992** ($<$0.001)	0.886 (0.004)
$\alpha = 0.4$	0.449 (0.213)	**0.989** (0.005)	0.896 (0.004)
$\alpha = 0.5$	0.559 (0.007)	**0.994** ($<$0.001)	0.905 (0.003)
Avg. Neuron Sparsity	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.317 (0.139)	**0.698** (0.024)	0.497 (0.014)
$\alpha = 0.2$	0.444 (0.015)	**0.743** (0.021)	0.627 (0.011)
$\alpha = 0.3$	0.382 (0.185)	**0.863** (0.003)	0.697 (0.010)
$\alpha = 0.4$	0.399 (0.196)	**0.828** (0.061)	0.721 (0.008)
$\alpha = 0.5$	0.519 (0.013)	**0.883** (0.003)	0.735 (0.004)

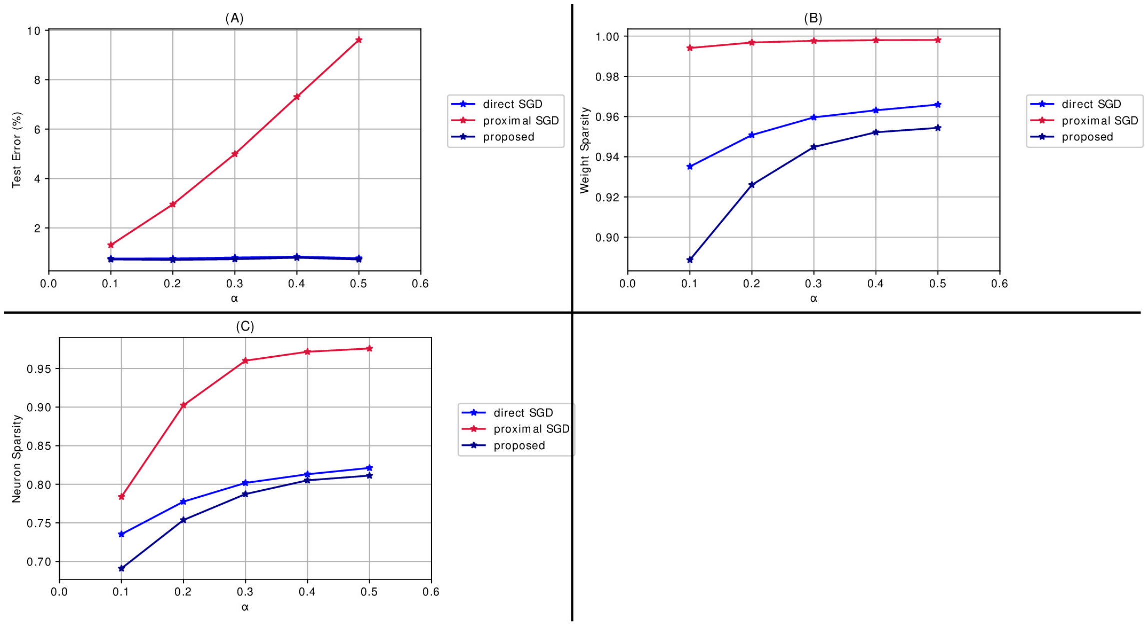


Figure 2.2: Mean results of algorithms applied to SGL_1 for Lenet-5 models trained on MNIST for 200 epochs across 5 runs when varying the regularization parameter $\lambda = \alpha/60000$ when $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. **(A)** Mean test error. **(B)** Mean weight sparsity. **(C)** Mean neuron sparsity.

algorithm are better than the sparsity by direct stochastic gradient descent. Therefore, our proposed algorithm generates the most accurate model with satisfactory sparsity among the three algorithms for sparse regularization.

2.3 Proofs

We provide proofs for the results discussed in Section 2.1.5.

2.3.1 Proof of Theorem 2.1

By (2.17a)-(2.17b), for each $k \in \mathbb{N}$, we have

$$F_\beta(V^k, W^{k+1}) \leq F_\beta(V^k, W) \tag{2.22}$$

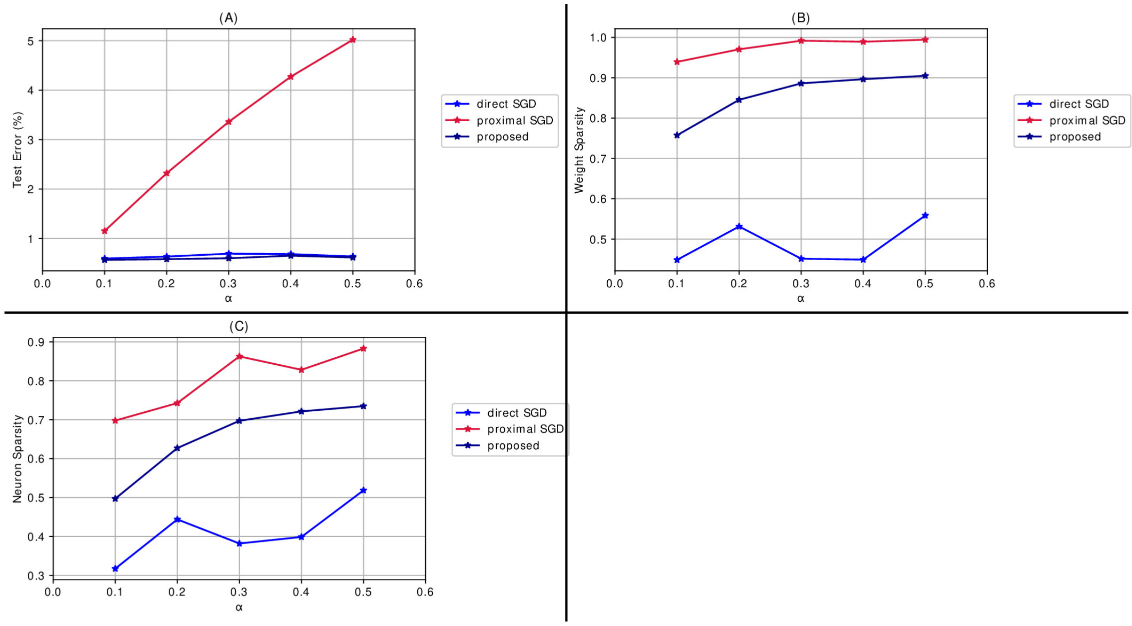


Figure 2.3: Mean results of algorithms applied to SGL_1 for Lenet-5 models trained on MNIST with lowest test errors across 5 runs when varying the regularization parameter $\lambda = \alpha/60000$ when $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. **(A)** Mean test error. **(B)** Mean weight sparsity. **(C)** Mean neuron sparsity.

for all W, and

$$F_\beta(V^{k+1}, W^{k+1}) \leq F_\beta(V, W^{k+1}) \tag{2.23}$$

for all V. By (2.23), we have

$$F_\beta(V^+, W^+) \leq F_\beta(V^k, W^+) \tag{2.24}$$

for each $k \in \mathbb{N}$. Altogether, we have

$$F_\beta(V^+, W^+) \leq F_\beta(V^k, W^k) \tag{2.25}$$

for each $k \in \mathbb{N}$, so $\{F_\beta(V^k, W^k)\}_{k=1}^{\infty}$ is nonincreasing. Since $F_\beta(V^k, W^k) \geq 0$ for all $k \in \mathbb{N}$, its limit $\lim\limits_{k\to\infty} F_\beta(V^k, W^k)$ exists. From (2.22)-(2.24), we have

$$F_\beta(V^+, W^+) \leq F_\beta(V^k, W^+) \leq F_\beta(V^k, W^k).$$

Taking the limit gives us

$$\lim_{k\to\infty} F_\beta(V^k, W^+) = \lim_{k\to\infty} F_\beta(V^k, W^k). \tag{2.26}$$

Since (V^*, W^*) is an accumulation point of $\{(V^k, W^k)\}_{k=1}^{\infty}$, there exists a subsequence K such that

$$\lim_{k\in K\to\infty} (V^k, W^k) = (V^*, W^*). \tag{2.27}$$

Because $r(\cdot)$ is lower semicontinuous and $\lim\limits_{k\in K\to\infty} V^k = V^*$, there exists $k' \in K$ such that $k \geq k'$ implies $r(V_l^k) \geq r(V_l^*)$ for each $l = 1, \ldots, L$. Using this result along with (2.23), we obtain

$$\begin{aligned}
F_\beta(V, W^k) &\geq F_\beta(V^k, W^k) \\
&= \tilde{\mathcal{L}}(W^k) + \sum_{l=1}^{L} \left[\lambda \left(\mathcal{R}_{GL}(W_l^k) + r(V_l^k)\right) + \frac{\beta}{2}\|V_l^k - W_l^k\|_2^2\right] \\
&\geq \tilde{\mathcal{L}}(W^k) + \sum_{l=1}^{L} \left[\lambda \left(\mathcal{R}_{GL}(W_l^k) + r(V_l^*)\right) + \frac{\beta}{2}\|V_l^k - W_l^k\|_2^2\right]
\end{aligned}$$

for $k \geq k'$. As $k \in K \to \infty$, we have

$$F_\beta(V, W^*) \geq \tilde{\mathcal{L}}(W^*) + \sum_{l=1}^{L} \left[\lambda \left(\mathcal{R}_{GL}(W_l^*) + r(V_l^*)\right) + \frac{\beta}{2}\|V_l^* - W_l^*\|_2^2\right] = F_\beta(V^*, W^*) \tag{2.28}$$

by continuity, so it follows that $V^* \in \arg\min\limits_{V} F_\beta(V, W^*)$.

For notational convenience, let

$$\tilde{\mathcal{R}}_{\lambda,\beta}(V,W) := \sum_{l=1}^{L}\left[\lambda\mathcal{R}_{GL}(W_l) + \frac{\beta}{2}\|V_l - W_l\|_2^2\right]. \tag{2.29}$$

By (2.22), we have

$$\begin{aligned}\tilde{\mathcal{L}}(W) + \tilde{\mathcal{R}}_{\lambda,\beta}(V^k, W) &= F_\beta(V^k, W) - \lambda\sum_{i=1}^{L} r(V_l^k)\\ &\geq F_\beta(V^k, W^+) - \lambda\sum_{i=1}^{L} r(V_l^k) = \tilde{\mathcal{L}}(W^+) + \tilde{\mathcal{R}}_{\lambda,\beta}(V^k, W^+).\end{aligned} \tag{2.30}$$

Because $\lim_{k\in K\to\infty} V^k$ exists, the sequence $\{V^k\}_{k\in K}$ is bounded. If $r(\cdot)$ is ℓ_0, transformed ℓ_1, or SCAD, then $\{r(V^k)\}_{k\in K}$ is bounded. If $r(\cdot)$ is ℓ_1, then $r(\cdot)$ is coercive. If $r(\cdot)$ is $\ell_1 - \alpha\ell_2$, then $r(\cdot)$ is bounded above by ℓ_1. Overall, this follows that $\{r(V^k)\}_{k\in K}$ bounded as well. Hence, there exists a further subsequence $\overline{K} \subset K$ such that $\lim_{k\in\overline{K}\to\infty} r(V^k)$ exists. So, we obtain

$$\begin{aligned}\lim_{k\in\overline{K}\to\infty} \tilde{\mathcal{L}}(W^+) + \tilde{\mathcal{R}}_{\lambda,\beta}(V^k, W^+) &= \lim_{k\in\overline{K}\to\infty} F_\beta(V^k, W^+) - \lambda\sum_{i=1}^{L} r(V_l^k)\\ &= \lim_{k\in\overline{K}\to\infty} F_\beta(V^k, W^+) - \lim_{k\in\overline{K}\to\infty} \lambda\sum_{i=1}^{L} r(V_l^k)\\ &= \lim_{k\in\overline{K}\to\infty} F_\beta(V^k, W^k) - \lim_{k\in\overline{K}\to\infty} \lambda\sum_{i=1}^{L} r(V_l^k)\\ &= \lim_{k\in\overline{K}\to\infty} F_\beta(V^k, W^k) - \lambda\sum_{i=1}^{L} r(V_l^k)\\ &= \lim_{k\in\overline{K}\to\infty} \tilde{\mathcal{L}}(W^k) + \tilde{\mathcal{R}}_{\lambda,\beta}(V^k, W^k)\\ &= \tilde{\mathcal{L}}(W^*) + \tilde{\mathcal{R}}_{\lambda,\beta}(W^*, V^*)\end{aligned} \tag{2.31}$$

after applying (2.26) in the third inequality and by continuity in the last equality.

Taking the limit over the subsequence $\overline{K}$ in (2.30) and applying (2.31), we obtain

$$\tilde{\mathcal{L}}(W) + \tilde{\mathcal{R}}_{\lambda,\beta}(V^*, W) \geq \tilde{\mathcal{L}}(W^*) + \tilde{\mathcal{R}}_{\lambda,\beta}(W^*, V^*) \tag{2.32}$$

by continuity. Adding $\sum_{l=1}^{L} r(V_l^*)$ on both sides yields

$$F_\beta(V^*, W) \geq F_\beta(V^*, W^*), \tag{2.33}$$

which follows that $W^* \in \arg\min_W F_\beta(V^*, W)$. This completes the proof.

2.3.2 Proof of Theorem 2.2

Because (V^*, W^*) is an accumulation point, there exists a subsequence K such that $\lim\limits_{k\in K\to\infty}(V^k, W^k) = (V^*, W^*)$. If $\{F_{\beta_k}(V^k, W^k)\}_{k=1}^{\infty}$ is uniformly bounded, there exists M such that $F_{\beta_k}(V^k, W^k) \leq M$ for all $k \in \mathbb{N}$. Then we have

$$\begin{aligned} M \geq F_{\beta_k}(V^k, W^k) &= \tilde{\mathcal{L}}(W) + \sum_{l=1}^{L} \left[\lambda \mathcal{R}_{GL}(W_l) + \lambda r(V_l) + \frac{\beta_k}{2}\|V_l - W_l\|_2^2\right] \\ &\geq \frac{\beta_k}{2}\sum_{l=1}^{L}\|V_l - W_l\|_2^2 \end{aligned}$$

As a result,

$$\sum_{l=1}^{L}\|V_l^k - W_l^k\|_2^2 \leq \frac{2}{\beta_k}M. \tag{2.34}$$

Taking the limit over $k \in K$, we have

$$\sum_{l=1}^{L}\|V_l^* - W_l^*\|_2^2 = 0,$$

which follows that $V^* = W^*$. As a result, (V^*, W^*) is a feasible solution to (2.15).

Chapter 3

Nonconvex Regularization for Network Slimming

One interesting yet straightforward approach in sparsifying CNNs is *network slimming* [134]. This method imposes ℓ_1 regularization on the scaling factors in the batch normalization layers. Due to ℓ_1 regularization, scaling factors corresponding to insignificant channels are pushed towards zeroes, narrowing down the important channels to retain, while the CNN model is being trained. Once the insignificant channels are pruned, the compressed model may need to be retrained since pruning can degrade its original accuracy. Overall, network slimming yields a compressed model with low run-time memory and number of computing operations. Since its inception, network slimming helps develop lightweight CNNs for various image classification tasks, such as traffic sign classification [245], facial expression recognition [151], and semantic segmentation. [90].

To improve the performance of network slimming, we propose replacing ℓ_1 regularization with an alternative regularization that promotes better sparsity and/or accuracy. Typically, better sparsity-promoting regularizers are nonconvex. Hence, we examine the ℓ_p penalty [48, 50,

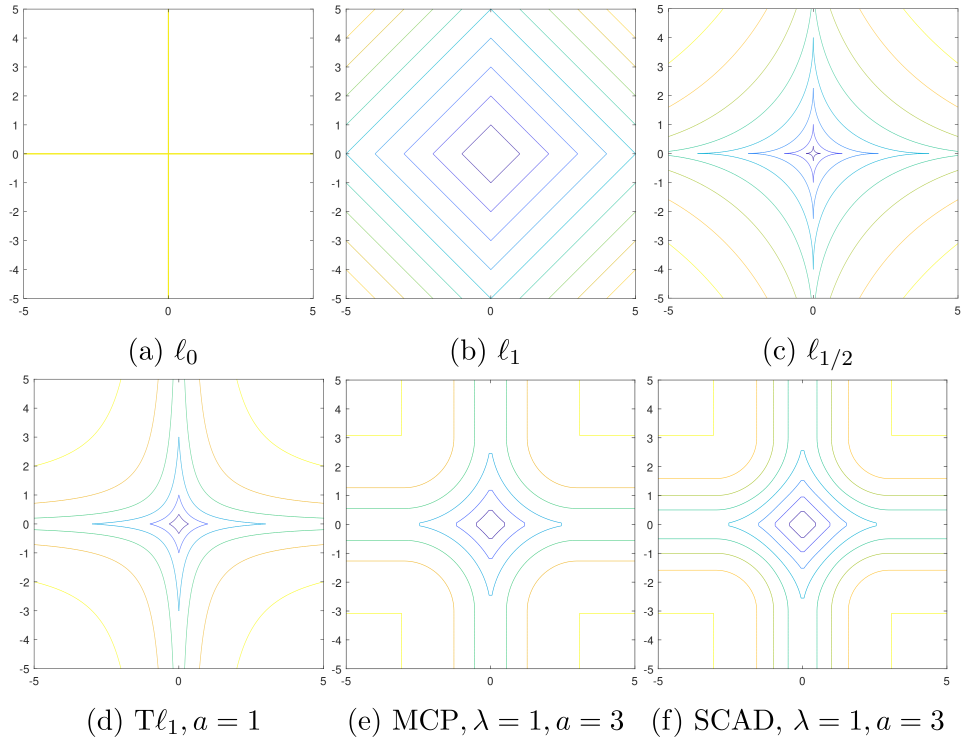


Figure 3.1: Contour plots of sparse regularizers.

222], transformed ℓ_1 (Tℓ_1) penalty [246, 247], the minimax concave penalty (MCP) [244], and the smoothly clipped absolute deviation (SCAD) penalty [69] due to their recent successes and popularity. These four regularizers have explicit formulas for their subgradients, which allow us to directly perform subgradient descent [190] when training CNNs.

3.1 Regularization Penalty

Let $z = (z_1, \ldots, z_n) \in \mathbb{R}^n$. The ℓ_1 penalty is described by

$$\|z\|_1 = \sum_{i=1}^{n} |z_i|, \tag{3.1}$$

while the ℓ_0 penalty is described by

$$\|z\|_0 = \sum_{i=1}^{n} \mathbb{1}_{\{z_i \neq 0\}}, \quad \text{where} \quad \mathbb{1}_{\{z_i \neq 0\}} = \begin{cases} 1 & \text{if } z_i \neq 0 \\ 0 & \text{if } z_i = 0. \end{cases} \tag{3.2}$$

Although ℓ_1 regularization is popular in sparse optimization in various applications such as compressed sensing [32, 31, 233] and compressive imaging [103, 148], it may not actually yield the sparsest solution [48, 139, 137, 222, 247]. Moreover, it is sensitive to outliers and it may yield biased solutions [69].

A nonconvex alternative to the ℓ_1 penalty is the ℓ_p penalty

$$\|z\|_p = \left(\sum_{i=1}^{n} |z_i|^p \right)^{1/p} \tag{3.3}$$

for $p \in (0, 1)$. The ℓ_p penalty interpolates ℓ_0 and ℓ_1 because as $p \to 0^+$, we have $\ell_p \to \ell_0$, and as $p \to 1^-$, we have $\ell_p \to \ell_1$. It recovers sparser solution than ℓ_1 for certain compressed sensing problems [50, 49]. Empirical studies [50, 223] demonstrate that for $p \in [1/2, 1)$, as p decreases, the solution becomes sparser by ℓ_p minimization, but for $p \in (0, 1/2)$, the performance becomes no longer significant. Moreover, it is used in image deconvolution [109, 33], hyperspectral unmixing [181], computed topography reconstruction [162], and image segmentation [126, 216]. Numerically, in compressed sensing, a small value ϵ is added to z_i to avoid blowup in the subgradient when $z_i = 0$. In this work, we will examine across different values of p since ℓ_p regularization may work differently in deep learning than in other areas.

Although ℓ_p may yield sparser solutions than ℓ_1, it is still biased because parameters with large weights could be overpenalized [70]. Hence, a better regularizer should also be unbiased. In fact, Fan and Li [69] suggested three properties that a regularizer should have: (1) continuity to avoid model instability; (2) sparsity to reduce model complexity; and (3) unbiasedness to avoid modeling bias due to overpenalization of large parameters. Hence, we consider regularizers that have all three properties, such as Tℓ_1, MCP, and SCAD.

The Tℓ_1 penalty is formulated as

$$P_a(z) = \sum_{i=1}^{n} \frac{(a+1)|z_i|}{a+|z_i|} \tag{3.4}$$

for $a > 0$. Tℓ_1 interpolates ℓ_0 and ℓ_1 because as $a \to 0^+$, we have T$\ell_1 \to \ell_0$, and as $a \to +\infty$, we have T$\ell_1 \to \ell_1$. It was validated to have the three aforementioned properties [149]. The Tℓ_1 penalty outperforms ℓ_1 and ℓ_p in compressed sensing problems with both coherent and incoherent sensing matrices [246, 247]. Additionally, the Tℓ_1 penalty yields satisfactory, sparse solutions in matrix completion [248] and deep learning [153].

The MCP penalty [244] is provided by

$$p_{\lambda,a}(z) = \sum_{i=1}^{n} \left[\left(\lambda|z_i| - \frac{z_i^2}{2a} \right) \mathbb{1}_{\{|z_i| \leq a\lambda\}} + \frac{a\lambda^2}{2} \mathbb{1}_{\{|z_i| > a\lambda\}} \right], \tag{3.5}$$

where $\lambda \geq 0$ and $a > 1$. The parameter λ acts as a regularization parameter while the parameter a controls the level of sparsity, where the smaller a is, the sparser the solution becomes. In fact, a allows MCP to roughly interpolate between ℓ_0 and ℓ_1. Originally, MCP is developed for variable selection [244], but it has been utilized in various other applications such as image restoration [235] and matrix completion [102].

Lastly, the SCAD penalty [69] is given by

$$\begin{aligned} &\tilde{p}_{\lambda,a}(z) \\ &= \sum_{i=1}^{n} \left[\lambda|z_i| \mathbb{1}_{\{|z_i| \leq \lambda\}} + \frac{2a\lambda|z_i| - z_i^2 - \lambda^2}{2(a-1)} \mathbb{1}_{\{\lambda < |z_i| \leq a\lambda\}} + \frac{\lambda^2(a+1)}{2} \mathbb{1}_{\{|z_i| > a\lambda\}} \right], \end{aligned} \tag{3.6}$$

where $\lambda \geq 0$ is the regularization parameter and $a > 2$ controls the level of sparsity similarly to MCP. In both linear and logistic regression problems, SCAD outperforms ℓ_1 in variable selection [69]. Beyond variable selection, it is applied in compressed sensing [158], bioinformatics [22, 209], image processing [80], and wavelet approximation [8].

Figure 3.1 displays the contour plots of the aforementioned regularizers. With ℓ_1 regularization, the solution tends towards one of the corners of the rotated squares, making it sparse. Compared with ℓ_1, the level lines of the nonconvex regularizers bend more inward towards the axes, encouraging the solutions to coincide with one of the corners. In addition, the contour plots of the nonconvex regularizers appear more similar to the contour plot of ℓ_0. Therefore, solutions tend to be sparser with nonconvex regularization than with ℓ_1 regularization.

Throughout the rest of the chapter, we define $\lambda p_{1,a}(\cdot) \coloneqq p_{\lambda,a}(\cdot)$ and $\lambda \tilde{p}_{1,a}(\cdot) \coloneqq \tilde{p}_{\lambda,a}(\cdot)$.

3.2 Proposed Method

3.2.1 Batch Normalization Layer

Batch normalization [97] has been instrumental in speeding the convergence and improving generalization of many deep learning models, especially CNNs [197, 89]. In most state-of-the-arts CNNs, a convolutional layer is always followed by a batch normalization layer. Within a batch normalization layer, features generated by the preceding convolutional layer are normalized by their mean and variance within the same channel. Afterward, a linear transformation is applied to compensate for the loss of their representative abilities.

We mathematically describe the process of the batch normalization layer. First we suppose that we are working with 2D images. Let x' be a feature computed by a convolutional layer. Each entry of x' is denoted by x'_i, where $i = (i_N, i_C, i_H, i_W)$ indexes the features in (N, C, H, W) order. Here, N is the batch axis, C is the image channel axis, H is the

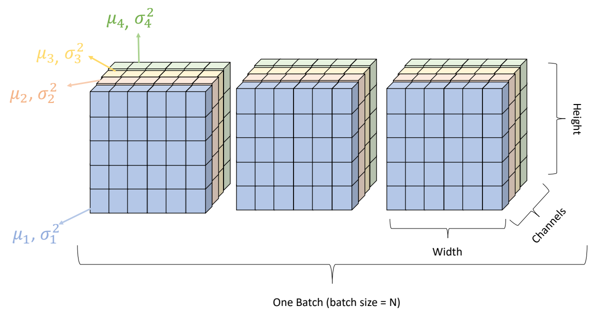


Figure 3.2: Visualization of batch normalization on a feature map. The mean and variance of the values of the pixels of the same colors corresponding to the channels are computed and are used to normalize these pixels.

image height axis, and W is the image width axis. We define the index set $S_i = \{k = (k_N, k_C, k_H, k_W) : k_C = i_C\}$, where k_C and i_C are the respective subindices of k and i along the C axis. In other words, the index set consists of pixels that belong to the same channel. The mean μ_i and variance σ_i^2 are computed as follows:

$$\mu_i = \frac{1}{|S_i|} \sum_{k \in S_i} x'_k, \quad \sigma_i^2 = \frac{1}{|S_i|} \sum_{k \in S_i} (x'_k - \mu_i)^2 + \epsilon \tag{3.7}$$

for some small value $\epsilon > 0$, where $|\mathcal{A}|$ denotes the cardinality of the set $\mathcal{A}$. Then we normalize x'_i by $\hat{x}_i = \frac{x'_i - \mu_i}{\sigma_i}$ for each index i. In short, the mean and variance are computed from pixels of the same channel index and are used to normalize them. Visualization is provided in Figure 3.2. Lastly, the output of the batch normalization layer is computed as a linear transformation of the normalized features:

$$z_i = \gamma_{i_C} \hat{x}_i + \beta_{i_C}, \tag{3.8}$$

where $\gamma_{i_C}, \beta_{i_C} \in \mathbb{R}$ are trainable parameters. Additionally, γ_{i_C} is defined to be the scaling factor related to the channel i_C.

Algorithm 2: Algorithm for minimizing (3.9)

Input: Regularization parameter λ, learning rate η, sparse regularizer $\mathcal{R}$

Initialize $\mathcal{W}^0$, excluding $\{\gamma_l\}_{l=1}^L$, with random values.

Initialize $\{\gamma_l^0\}_{l=1}^L$ with entries 0.5.

1: **for** each epoch $t = 1, \ldots, T$ **do**

2: $\quad \mathcal{W}^t = \mathcal{W}^{t-1} - \frac{\eta}{N}\sum_{i=1}^{N} \nabla \mathcal{L}(h(x_i, \mathcal{W}^{t-1}), y_i)$ by stochastic gradient descent or variant.

3: $\quad \gamma_l^t = \gamma_l^{t-1} - \eta\lambda\partial\mathcal{R}(\gamma_l^{t-1})$ for $l = 1, \ldots, L$.

4: **end for**

3.2.2 Network Slimming with Nonconvex Sparse Regularization

Since the scaling factors γ_{i_C}'s in (3.8) are associated with the channels of a convolutional layer, we aim to penalize them with a sparse regularizer in order to identify which channels are irrelevant to the compressed CNN model. Suppose we have a training dataset that consists of N input-output pairs $\{(x_i, y_i)\}_{i=1}^N$ and a CNN with L convolutional layers, where each is followed by a batch normalization layer. Then we have two sets of vectors $\{\gamma_l\}_{l=1}^L$ and $\{\beta_l\}_{l=1}^L$, where $\gamma_l = (\gamma_{l,1}, \ldots, \gamma_{l,C_l})$ and $\beta_l = (\beta_{l,1}, \ldots, \beta_{l,C_l})$ with C_l being the number of channels in the lth convolutional layer. Let $\mathcal{W}$ be the weight parameters that include $\{\gamma_l\}_{l=1}^L$ and $\{\beta_l\}_{l=1}^L$. Hence, the trainable parameters $\mathcal{W}$ of the CNN are learned by minimizing the following objective function:

$$\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(h(x_i, \mathcal{W}), y_i) + \lambda \sum_{l=1}^{L} \mathcal{R}(\gamma_l), \tag{3.9}$$

where $h(\cdot,\cdot)$ is the output of the CNN used for prediction, $\mathcal{L}(\cdot,\cdot)$ is a loss function, $\mathcal{R}(\cdot)$ is a sparse regularizer, and $\lambda > 0$ is a regularization parameter for $\mathcal{R}(\cdot)$. When $\mathcal{R}(\cdot) = \|\cdot\|_1$, we have the original network slimming method. As mentioned earlier, since ℓ_1 regularization may not yield the sparsest solution and it could potentially be biased, we investigate the method with a nonconvex regularizer, where $\mathcal{R}(\cdot)$ is $\|\cdot\|_p^p$, $P_a(\cdot)$, $p_{1,a}(\cdot)$, or $\tilde{p}_{1,a}(\cdot)$.

To minimize (3.9), stochastic gradient descent is applied to the loss function term while subgradient descent is applied to the regularizer term [190]. The algorithm is summarized in Algorithm 2. Subgradient descent is applicable to the nonconvex regularizers $\mathcal{R}(z)$ for $z \in \mathbb{R}^n$ as it is for ℓ_1. Like ℓ_1, the nonconvex regularizers are of the form $\sum_{i=1}^n r(z_i)$, where $r : \mathbb{R} \to \mathbb{R}$ has the following properties:

i) $r(0) = 0$;

ii) r is an even, proper, and continuous function;

iii) r is increasing on $[0, +\infty)$;

iv) r is differentiable on $(-\infty, 0) \cup (0, +\infty)$.

These properties ensure that r is differentiable everywhere except at 0 and 0 is the global minimum of r while being its only local minimum. As a result, the regularizers are differentiable when $z_i \neq 0$ for all $i = 1, \ldots, n$. Hence, subgradient descent becomes gradient descent at these points. If $z_i = 0$ for at least one index i, then we need to compute its (limiting) subgradient [173, Definition 6.1] and decide a candidate descent direction. Fortunately, because $\mathcal{R}(z) = \sum_{i=1}^n r(z_i)$, we have

$$\partial \mathcal{R}(z) = (\partial r(z_1), \partial r(z_2), \ldots, \partial r(z_n))$$

by [173, Proposition 6.17(e)]. This means that at each component $r(z_i)$, we can compute its subgradient $\partial r(z_i)$ individually and select a descent direction from the set. Since 0 is a local minimum of r, we have $0 \in \partial r(0)$, so we can select 0 as a descent direction for simplicity. Table 1 presents the subgradients of the regularizers.

After the CNN is trained with (3.9) using Algorithm 2, we prune the channels whose scaling factors are small in magnitude, giving us a compressed model. However, the compressed model may lose its original accuracy, so it may need to be retrained but without the sparse regularizer in order to attain its original accuracy or better.

Table 3.1: Sparse regularizers and their (limiting) subgradients.

Name	$\mathcal{R}(z)$	$\partial R(z)$
ℓ_1	$\|z\|_1 = \sum_{i=1}^n \lvert z_i \rvert$	$\partial \|z\|_1 = \left\{ \zeta \in \mathbb{R}^n : \zeta_i = \begin{cases} \text{sgn}(z_i) & \text{if } z_i \neq 0 \\ \zeta_i \in [-1,1] & \text{if } z_i = 0 \end{cases} \right\}$
ℓ_p	$\|z\|_p^p = \sum_{i=1}^n \lvert z_i \rvert^p$	$\partial \|z\|_p^p = \left\{ \zeta \in \mathbb{R}^n : \zeta_i = \begin{cases} \frac{p \cdot \text{sgn}(z_i)}{\lvert z_i \rvert^{1-p}} & \text{if } z_i \neq 0 \\ \zeta_i \in \mathbb{R} & \text{if } z_i = 0 \end{cases} \right\}$
Tℓ_1	$P_a(z) = \sum_{i=1}^n \frac{(a+1)\lvert z_i \rvert}{a + \lvert z_i \rvert}$	$\partial P_a(z) = \left\{ \zeta \in \mathbb{R}^n : \zeta_i = \begin{cases} \frac{a(a+1)\text{sgn}(z_i)}{(a+\lvert z_i \rvert)^2} & \text{if } z_i \neq 0 \\ \zeta_i \in \left[-\frac{a+1}{a}, \frac{a+1}{a}\right] & \text{if } z_i = 0 \end{cases} \right\}$
MCP	$p_{\lambda,a}(z) = \sum_{i=1}^n \left[\left(\lambda \lvert z_i \rvert - \frac{z_i^2}{2a} \right) \mathbb{1}_{\{\lvert z_i \rvert \leq a\lambda\}} + \frac{a\lambda^2}{2} \mathbb{1}_{\{\lvert z_i \rvert > a\lambda\}} \right]$	$\partial p_{\lambda,a}(z) = \left\{ \zeta \in \mathbb{R}^n : \zeta_i = \begin{cases} 0 & \text{if } \lvert z_i \rvert > a\lambda \\ \lambda \text{sgn}(z_i) - \frac{z_i}{a} & \text{if } 0 < \lvert z_i \rvert \leq a\lambda \\ \zeta_i \in [-\lambda, \lambda] & \text{if } z_i = 0 \end{cases} \right\}$
SCAD	$\tilde{p}_{\lambda,a}(z) = \sum_{i=1}^n \left[\lambda \lvert z_i \rvert \mathbb{1}_{\{\lvert z_i \rvert \leq \lambda\}} + \frac{2a\lambda \lvert z_i \rvert - z_i^2 - \lambda^2}{2(a-1)} \mathbb{1}_{\{\lambda < \lvert z_i \rvert \leq a\lambda\}} + \frac{\lambda^2(a+1)}{2} \mathbb{1}_{\{\lvert z_i \rvert > a\lambda\}} \right]$	$\partial \tilde{p}_{\lambda,a}(z) = \left\{ \zeta \in \mathbb{R}^n : \zeta_i = \begin{cases} 0 & \text{if } \lvert z_i \rvert > a\lambda \\ \frac{a\lambda \text{sgn}(z_i) - z_i}{a-1} & \text{if } \lambda < \lvert z_i \rvert \leq a\lambda \\ \lambda \text{sgn}(z_i) & \text{if } 0 < \lvert z_i \rvert \leq \lambda \\ \zeta_i \in [-\lambda, \lambda] & \text{if } z_i = 0 \end{cases} \right\}$

3.3 Experimental Results

We apply the proposed nonconvex network slimming using $\ell_p(0 < p < 1)$, Tℓ_1, MCP, and SCAD regularization on various networks and datasets and compare their results against the original network slimming with ℓ_1 regularization as the baseline.

Code for the experiments is available at
`https://github.com/kbui1993/NonconvexNetworkSlimming`.

3.3.1 Datasets

CIFAR 10/100. The CIFAR 10/100 dataset [110] consists of 50k training color images and 10k test color images with 10/100 classes total. The resolution of each image is 32×32. To preprocess the dataset, we apply the data augmentation techniques (horizontal flipping and translation by 4 pixels) that have been standard in practice [95, 131, 79, 89, 134] followed by global contrast normalization and ZCA whitening [79]. These preprocessing techniques help improve the classification accuracy of CNNs on CIFAR 10 and 100 as demonstrated in [79, 131].

SVHN. The SVHN dataset [164] consists of 32×32 color images. The entire training set has 604,388 images and the test set has has 26,032 images. Before training on the dataset, each image is normalized by the channel means and standard deviations.

We evaluate the proposed methods on VGG-19 [192], DenseNet-40 [94], and ResNet-164 [89], three networks that were examined in [134]. More specifically, we use a variation of VGG-19 from `https://github.com/szagoruyko/cifar.torch`, a 40-layer DenseNet with a growth rate of 12, and a 164-layer pre-activation ResNet with a bottleneck structure.

3.3.2 Implementation Details

Training the Network. To perform a fair comparison between the original network slimming and the proposed nonconvex network slimming, we emulate most of the training settings in the original work [134]. All networks are trained from scratch using stochastic gradient descent. The initial learning rate is set at 0.1, and it is reduced by a factor of 10 at the 50% and 75% of the total number of epochs. In addition, we use weight decay of 10^{-4} and Nesterov momentum [196] of 0.9 without dampening. On CIFAR 10/100, we train for 160 epochs, while on SVHN, we train for 20 epochs. On both datasets, the training batch size is 64. Weight initialization is based on [88] and scaling factor initialization is set to 0.5 as done in [134]. We examine the following regularizers for network slimming: ℓ_1, $\ell_p(p = 0.25, 0.5, 0.75)$, T$\ell_1(a = 0.5, 1.0, 10.0)$, MCP ($a = 5000, 10000, 15000$), and SCAD ($a = 5000, 10000, 15000$). The examined parameter values for these regularizers are chosen because they attain similar model accuracy as the baseline model without scaling factor regularization and they can prune a model by at least 40% of its channels. Lastly, we have the regularization parameter $\lambda = 10^{-4}$ for VGG-19 and DenseNet-40 and $\lambda = 5 \times 10^{-5}$ for ResNet-164. The regularization parameter is chosen by trying to balance between model accuracy and channel sparsity.

Pruning the Network. After a model is trained, its channels are pruned globally. For example, we specify a channel pruning ratio to be 0.35 or a channel pruning percentage to be 35% and determine the 35th percentile among all magnitudes of the scaling factors of the model. The 35th percentile is set as the threshold. Any channels whose scaling factors are below the threshold in magnitude are pruned.

Since the channels are pruned globally, there is a threshold specific for each model: if the pruning ratio is above a certain value, a model becomes over-pruned. That is, the model cannot be used for inference because at least one of its layers has all of its channels removed.

Retraining the Network. We retrain the pruned model without regularization on the scaling factors with the same optimization setting as the first time training it. The purpose of retraining is to at least recover the compressed model's original accuracy prior to pruning.

Performance Metrics. We compare the regularizers' performances based on test accuracy and compression of their respective models.

After pruning a network by its channels, we measure its compression by the remaining number of parameters and floating point operations (FLOPs). The number of parameters relates to the storage cost while the number of FLOPS relates to the computational cost. In our experiments, we report the following percentages:

$$\text{Percentage of parameters pruned} = \left(1 - \frac{\text{\# parameters remaining}}{\text{total \# network parameters}}\right) \times 100\%$$

and

$$\text{Percentage of FLOPs pruned} = \left(1 - \frac{\text{\# FLOPs remaining}}{\text{total \# network FLOPs}}\right) \times 100\%.$$

Since CNNs are highly nonconvex, each run of the same model and regularizer with the same hyperparameters will give a different result. Hence, we train each model of one regularizer five times and compute the mean. Therefore, the mean test accuracies and mean ratios/percentages of parameters/FLOPs pruned are computed from five runs each.

3.3.3 Channel Pruning Results

VGG-19. VGG-19 has about 20 million parameters and 7.97×10^8 FLOPs. Table 3.2 shows the relationships between channel pruning ratios and mean percentages of parameters/FLOPs pruned. Figure 3.3 shows the effect of channel pruning on mean test accuracies.

On CIFAR 10, according to Table 3.2a, most of the nonconvex regularizers prune more parameters than ℓ_1 up to channel pruning ratio 0.50. Although more parameters are pruned, MCP and SCAD require more FLOPs in general compared to ℓ_1. On the other hand, ℓ_p and $\mathrm{T}\ell_1$ outperform ℓ_1 with respect to percentages of parameters/FLOPs pruned for channel pruning ratio at least 0.60. Additionally, the models trained with $\ell_{1/2}$ and $\ell_{3/4}$ can have at least 80% of its channels pruned and still be used for inference even though their test accuracies are low. However, their test accuracies can be improved if the models were retrained. According to Figure 3.3, $\ell_{3/4}$, $\mathrm{T}\ell_1$, MCP, and SCAD are more robust than ℓ_1 to channel pruning since their accuracies drop at higher channel pruning ratios. Although both $\ell_{1/2}$ and $\ell_{1/4}$ compress the model significantly compared to other regularizers, they are very sensitive to channel pruning.

On CIFAR 100, according to Table 3.2b, ℓ_p and $\mathrm{T}\ell_1(a = 0.5, 1.0)$ require less parameters and FLOPs compared to ℓ_1 when the channel pruning ratios are at least 0.40. MCP and SCAD have comparable number of parameters and FLOPs pruned as ℓ_1. Figure 3.3 shows that $\mathrm{T}\ell_1$ is robust against channel pruning, especially when $a = 0.5$. At channel pruning ratio 0.6, the accuracy for $\mathrm{T}\ell_1(a = 0.5)$ does not drop as much compared to other values of a and also other nonconvex regularizers. For the other regularizers, ℓ_1 is outperformed by $\ell_{3/4}$, $\ell_{1/2}$, MCP, and SCAD ($a = 10000, 15000$). Like for CIFAR 10, models trained with either $\ell_{1/2}$ or $\ell_{1/4}$ are still sensitive to channel pruning.

Table 3.2: Effect of channel pruning on the mean pruned parameter / FLOPs percentages (%) on VGG-19 trained on (a) CIFAR 10, (b) CIFAR 100, and (c) SVHN. The mean is computed from five runs for each regularizer. For each channel pruning ratio, **bold** indicates outperforming ℓ_1; * indicates best value; and NA indicates at least one of the five models is over-pruned.

(a)	CIFAR 10								
Channel Pruning Ratio	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
ℓ_1	21.10% / 11.83%	39.34% / 22.18%	54.88% / 31.09%	67.56% / 38.51%	77.53% / 44.78%	84.71% / 49.68%	88.81% / 51.95%	NA	NA
$\ell_{3/4}$	**21.14%** / **12.11%**	**39.55%** / **22.71%**	**55.13%** / **32.06%**	**67.96%** / **39.97%**	**77.99%** / **46.08%**	**85.24%** / **51.37%**	**89.69%** / **54.96%**	NA	NA
$\ell_{1/2}$	**21.12%** / **12.38%**	**39.62%** / **22.65%**	**55.29%*** / **32.06%**	**68.09%*** / **39.75%**	**78.10%** / **46.45%**	**85.43%** / **52.02%**	**90.01%** / **56.12%**	**93.66%** / **65.52%**	NA
$\ell_{1/4}$	19.95% / **15.49%***	37.66% / **29.82%***	52.99% / **42.93%***	66.20% / **54.39%***	77.07% / **64.47%***	**85.76%** / **73.44%***	**92.14%*** / **81.89%***	**96.54%*** / **91.07%***	**99.05%*** / **98.32%***
$\text{T}\ell_1(a = 10.0)$	**21.15%** / **11.92%**	**39.41%** / **22.59%**	**54.95%** / **31.85%**	**67.71%** / **39.39%**	**77.69%** / **45.77%**	**84.89%** / **50.47%**	**89.06%** / **52.75%**	NA	NA
$\text{T}\ell_1(a = 1.0)$	**21.16%** / **12.12%**	**39.35%** / **23.13%**	**54.94%** / **32.60%**	**67.88%** / **40.69%**	**78.06%** / **47.94%**	**85.55%** / **53.40%**	**90.34%** / **57.43%**	NA	NA
$\text{T}\ell_1(a = 0.5)$	20.94% / **12.59%**	39.29% / **23.66%**	**54.92%** / **33.61%**	**67.83%** / **42.34%**	**78.22%*** / **49.58%**	**85.92%*** / **55.36%**	**90.88%** / **59.84%**	NA	NA
$\text{MCP}(a = 15000)$	**21.18%** / 11.56%	**39.48%** / 21.43%	**54.96%** / 30.31%	**67.62%** / 37.75%	77.53% / 43.76%	84.58% / 48.05%	88.58% / 50.69%	NA	NA
$\text{MCP}(a = 10000)$	20.99% / 11.24%	**39.35%** / 21.23%	**54.97%** / 29.94%	**67.64%** / 37.71%	**77.55%** / 43.42%	84.61% / 47.88%	88.63% / 50.49%	NA	NA
$\text{MCP}(a = 5000)$	**21.20%** / 10.97%	**39.71%*** / 20.92%	**55.24%** / 29.31%	**67.87%** / 36.32%	**77.61%** / 41.97%	84.53% / 46.08%	88.56% / 49.49%	NA	NA
$\text{SCAD}(a = 15000)$	21.10% / 11.70%	**39.42%** / 21.83%	**54.97%** / 30.75%	**67.68%** / 37.83%	**77.56%** / 43.62%	84.66% / 48.09%	88.71% / 50.70%	NA	NA
$\text{SCAD}(a = 10000)$	**21.21%** / 11.23%	**39.45%** / 20.95%	**54.95%** / 29.89%	**67.60%** / 37.33%	77.44% / 43.23%	84.53% / 47.52%	88.57% / 50.43%	NA	NA
$\text{SCAD}(a = 5000)$	**21.24%*** / 11.25%	**39.65%** / 21.08%	**55.16%** / 29.64%	**67.77%** / 36.77%	**77.58%** / 42.69%	84.58% / 46.96%	88.62% / 50.19%	NA	NA

(b)	CIFAR 100								
Channel Pruning Ratio	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
ℓ_1	21.91% / 12.44%	40.36% / 22.89%*	55.83% / 28.19%	67.45% / 31.75%	75.35% / 36.08%	NA	NA	NA	NA
$\ell_{3/4}$	**21.98%** / 10.92%	**40.64%** / 20.75%	**56.04%** / **28.64%**	**68.01%** / **34.77%**	**76.54%** / **38.40%**	NA	NA	NA	NA
$\ell_{1/2}$	**22.02%** / 10.89%	**40.85%*** / 20.04%	**56.29%*** / 27.83%	**68.41%** / **34.23%**	**77.19%** / **39.40%**	**83.07%** / **43.82%**	NA	NA	NA
$\ell_{1/4}$	**22.00%** / 10.80%	**40.71%** / 20.52%	**56.21%** / **29.10%**	**68.53%*** / **36.59%**	**78.16%*** / **44.28%***	**85.71%*** / **54.15%***	**91.48%*** / **68.94%***	**96.54%*** / **86.86%***	NA
$\text{T}\ell_1(a = 10.0)$	21.83% / 12.41%	40.34% / 22.62%	55.62% / **29.71%**	**67.80%** / **32.97%**	**76.07%** / **37.00%**	NA	NA	NA	NA
$\text{T}\ell_1(a = 1.0)$	21.87% / 11.21%	**40.47%** / 20.85%	**55.99%** / **29.04%**	**68.22%** / **36.22%**	**77.18%** / **40.47%**	**82.90%** / **43.94%**	NA	NA	NA
$\text{T}\ell_1(a = 0.5)$	21.67% / 11.42%	40.33% / 21.52%	**55.97%** / **30.16%***	**68.49%** / **37.52%***	**77.99%** / **43.24%**	**84.09%** / **47.15%**	NA	NA	NA
$\text{MCP}(a = 15000)$	21.86% / 12.37%	40.28% / 22.40%	55.67% / 28.17%	67.29% / 31.62%	**75.38%** / 35.91%	NA	NA	NA	NA
$\text{MCP}(a = 10000)$	21.90% / 12.40%	40.24% / 22.60%	55.73% / 27.94%	67.28% / 31.60%	75.20% / 35.95%	NA	NA	NA	NA
$\text{MCP}(a = 5000)$	**22.03%*** / 11.90%	**40.49%** / 21.45%	**55.94%** / 26.46%	67.35% / 30.43%	75.03% / 34.78%	NA	NA	NA	NA
$\text{SCAD}(a = 15000)$	**21.96%** / **12.48%***	**40.42%** / 22.36%	55.83% / 28.04%	**67.50%** / 31.70%	75.34% / 35.91%	NA	NA	NA	NA
$\text{SCAD}(a = 10000)$	21.90% / 11.76%	40.28% / 21.82%	55.71% / 27.34%	67.18% / 31.07%	75.02% / 35.56%	NA	NA	NA	NA
$\text{SCAD}(a = 5000)$	**22.01%** / 11.59%	**40.49%** / 20.60%	55.75% / 25.63%	66.91% / 29.91%	74.50% / 34.47%	NA	NA	NA	NA

(c)	SVHN								
Channel Pruning Ratio	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
ℓ_1	19.92% / 15.90%	37.51% / 30.99%	52.91% / 43.85%	66.08% / 55.25%	76.98% / 65.06%	85.60% / 73.63%	92.00% / 80.80%	96.12% / 86.39%	NA
$\ell_{3/4}$	**19.96%** / **16.04%**	**37.60%** / 30.91%	**52.96%** / 43.82%	66.00% / **55.74%**	76.84% / **65.93%**	85.52% / **74.49%**	92.00% / **81.53%**	**96.23%** / **87.23%**	NA
$\ell_{1/2}$	19.80% / **16.80%**	37.35% / **31.70%**	52.74% / **44.89%**	65.84% / **56.70%**	76.79% / **67.00%**	85.52% / **75.75%**	**92.01%** / **83.14%**	**96.31%** / **88.36%**	**98.94%** / **95.45%**
$\ell_{1/4}$	19.36% / **17.72%***	36.77% / **32.96%***	52.07% / **47.02%***	65.13% / **59.17%***	76.01% / **70.57%***	84.86% / **79.92%***	91.59% / **87.85%***	**96.36%** / **93.72%***	**99.08%*** / **98.15%***
$\text{T}\ell_1(a = 10.0)$	**19.94%** / 15.90%	37.39% / 30.91%	52.71% / **44.18%**	65.94% / **55.64%**	76.83% / **65.68%**	85.53% / **73.99%**	91.96% / **80.96%**	**96.19%** / **86.70%**	**98.60%** / **93.70%**
$\text{T}\ell_1(a = 1.0)$	19.71% / **17.01%**	37.17% / **32.34%**	52.55% / **45.74%**	65.70% / **57.43%**	76.70% / **67.21%**	85.44% / **76.19%**	**92.02%*** / **83.11%**	**96.38%** / **88.58%**	NA
$\text{T}\ell_1(a = 0.5)$	**19.99%** / **16.20%**	**37.52%** / **31.27%**	52.70% / **44.95%**	65.70% / **57.29%**	76.68% / **67.60%**	85.40% / **76.66%**	91.98% / **83.79%**	**96.43%*** / **89.53%**	**98.73%** / **94.41%**
$\text{MCP}(a = 15000)$	**20.14%** / 15.43%	**37.86%** / 29.52%	**53.15%** / 42.46%	**66.23%** / 53.76%	**77.07%** / 63.40%	**85.67%** / 71.68%	91.92% / 78.98%	95.87% / 84.33%	**98.61%** / **93.64%**
$\text{MCP}(a = 10000)$	**20.13%** / 15.60%	**37.91%** / 29.62%	**53.41%** / 41.85%	**66.40%** / 52.96%	**77.20%** / 62.47%	**85.72%** / 70.84%	91.88% / 77.91%	95.68% / 83.63%	NA
$\text{MCP}(a = 5000)$	**20.34%*** / 14.91%	**38.20%*** / 28.64%	**53.63%** / 40.96%	**66.72%** / 51.97%	**77.46%** / 61.03%	**85.76%** / 68.51%	91.68% / 75.01%	95.41% / 82.79%	NA
$\text{SCAD}(a = 15000)$	19.92% / **16.25%**	**37.55%** / 30.27%	**53.11%** / 42.75%	**66.19%** / 54.23%	**77.06%** / 63.81%	85.57% / 72.47%	91.91% / 79.38%	95.88% / 84.81%	NA
$\text{SCAD}(a = 10000)$	**19.97%** / 15.32%	**37.60%** / 29.41%	**53.22%** / 41.88%	**66.31%** / 52.80%	**77.15%** / 62.58%	**85.66%** / 70.73%	91.81% / 77.63%	95.65% / 83.85%	NA
$\text{SCAD}(a = 5000)$	**20.28%** / 15.07%	**38.10%** / 28.72%	**53.66%*** / 40.75%	**66.82%*** / 51.56%	**77.50%*** / 61.09%	**85.79%*** / 69.09%	91.73% / 75.80%	95.47% / 83.12%	NA

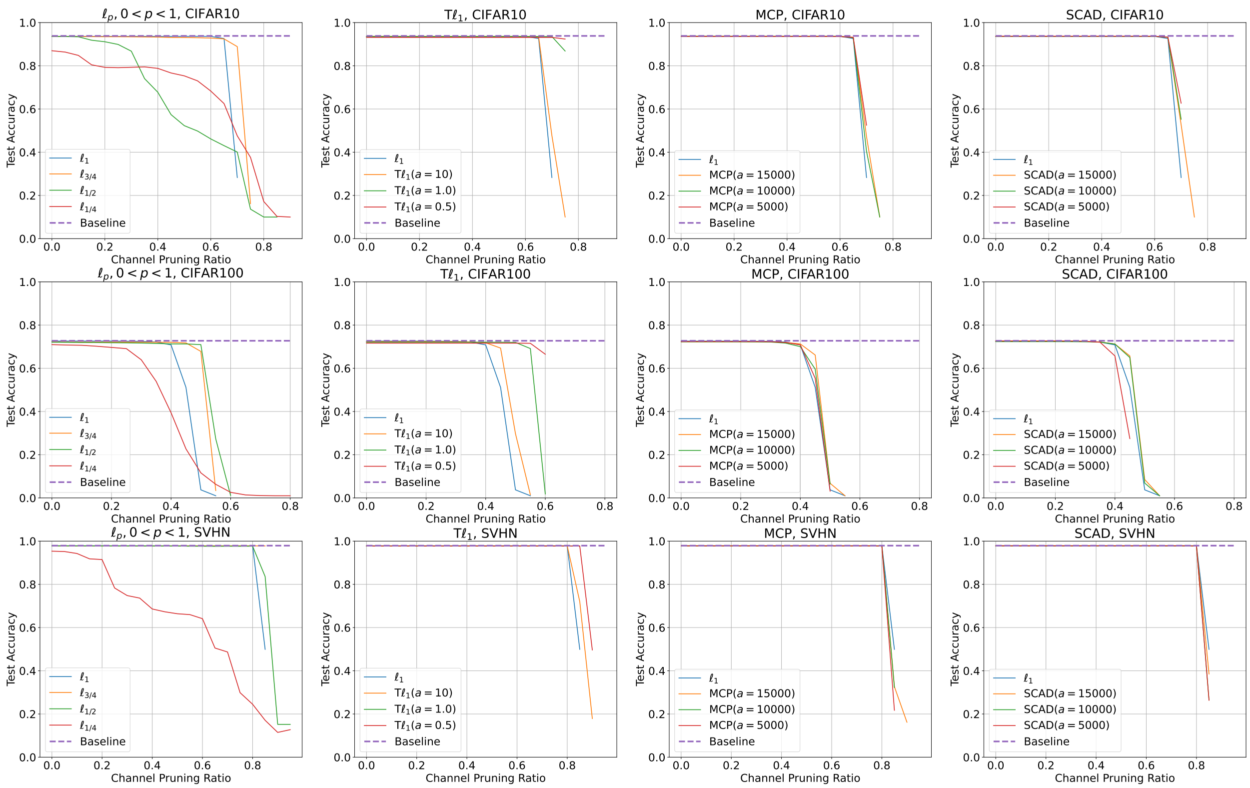


Figure 3.3: Effect of channel pruning on the mean test accuracy of five runs of VGG-19 on CIFAR 10/100 and SVHN. Baseline refers to the mean test accuracy of the unregularized model that is not pruned. Baseline accuracies are 93.83% for CIFAR 10, 72.73% for CIFAR 100, and 97.91% for SVHN.

Lastly, for SVHN, according to Table 3.2c, MCP and SCAD generally outperform ℓ_1 in parameter pruning percentages for channel pruning ratios up to 0.60, but they do not save more on FLOPs. However, FLOPs are reduced more by ℓ_p and Tℓ_1 in general across all channel pruning ratios. By Figure 3.3, $\ell_{3/4}, \ell_{1/2}$, and Tℓ_1 have higher test accuracies than ℓ_1 when the channel pruning ratio is at 0.85.

In general, nonconvex regularizers save more on parameters, FLOPs, or both. It is important to note that Tℓ_1, especially $a = 0.5$, helps preserve model accuracy against channel pruning, and $\ell_{1/4}$ is very sensitive to channel pruning.

DenseNet-40. DenseNet-40 has about 1 million parameters and 5.33×10^8 FLOPs. Table 3.3 shows the relationships between channel pruning ratios and mean percentages of parameters/FLOPs pruned. Figure 3.4 shows the effect of channel pruning on mean test accuracies.

On CIFAR 10, by Table 3.3a, ℓ_p and Tℓ_1 compress the model more in terms of number of parameters and FLOPs than ℓ_1 after channel pruning across the various levels of channel pruning ratios. In general, MCP and SCAD require slightly more FLOPS than ℓ_1, but they require similar number of parameters as ℓ_1. According to Figure 3.4, $\ell_p(p = 1/2, 3/4)$ and Tℓ_1 are more robust to channel pruning than ℓ_1 since their accuracies drop at higher channel pruning ratios, while MCP and SCAD are worse.

For CIFAR 100, Table 3.3b demonstrates that ℓ_p and Tℓ_1 generally reduce more parameters and FLOPs required than ℓ_1 after channel pruning. At channel pruning ratios 0.60 and above, MCP and SCAD reduce only more FLOPs than ℓ_1. In addition, models with MCP and SCAD regularization remain usable for inference after 90% of their channels are pruned, unlike models with ℓ_1 regularization. However, their test accuracies are unacceptable so that the models will need to be retrained to recover its original accuracies. According to Figure 3.4, $\ell_p(p = 1/2, 3/4)$, Tℓ_1, and MCP ($a = 15000$) are more robust to channel pruning than

Table 3.3: Effect of channel pruning on the mean pruned parameter / FLOPs percentages (%) on DenseNet-40 trained on (a) CIFAR 10, (b) CIFAR 100, and (c) SVHN. The mean is computed from five runs for each regularizer. For each channel pruning ratio, **bold** indicates outperforming ℓ_1; * indicates best value; and NA indicates at least one of the five models is over-pruned.

(a)	CIFAR 10								
Channel Pruning Ratio	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
ℓ_1	9.22% / 8.40%	18.35% / 16.63%	27.57% / 24.91%	36.73% / 33.02%	45.95% / 41.49%	55.15% / 49.75%	64.38% / 58.10%	73.75% / 68.18%	83.76% / 79.75%
$\ell_{3/4}$	**9.32%** / **8.53%**	**18.64%** / **16.79%**	**27.87%** / **25.62%**	**37.14%** / **34.10%**	**46.42%** / **42.85%**	**55.62%** / **51.27%**	**64.90%** / **59.71%**	**74.25%** / **68.63%**	**84.02%** / **80.07%**
$\ell_{1/2}$	**9.33%** / **8.65%**	**18.59%** / **17.08%**	**27.97%** / **25.96%**	**37.26%** / **34.71%**	**46.62%** / **43.33%**	**55.88%** / **51.85%**	**65.12%** / **60.32%**	**74.47%** / **69.14%**	**84.36%** / **80.13%**
$\ell_{1/4}$	**9.35%*** / **8.83%***	**18.71%** / **17.63%***	**28.13%*** / **26.39%***	**37.52%*** / **35.27%***	**47.05%*** / **44.74%***	**56.69%*** / **54.33%***	**66.56%*** / **64.34%***	**77.02%*** / **75.42%***	NA
T$\ell_1(a = 10.0)$	9.20% / 8.31%	18.34% / **16.83%**	**27.59%** / **25.32%**	**36.82%** / **33.65%**	**46.08%** / **41.97%**	**55.27%** / **50.17%**	**64.54%** / **58.19%**	**73.89%** / 68.01%	**83.89%** / 79.72%
T$\ell_1(a = 1.0)$	**9.35%*** / **8.67%**	**18.63%** / **17.09%**	**27.85%** / **25.39%**	**37.17%** / **34.04%**	**46.41%** / **42.32%**	**55.73%** / 50.93%	**65.14%** / **59.70%**	**74.46%** / **68.57%**	**84.23%** / **80.19%**
T$\ell_1(a = 0.5)$	**9.35%*** / **8.45%**	**18.72%*** / **16.99%**	**28.08%** / **25.82%**	**37.39%** / **34.47%**	**46.73%** / **43.13%**	**56.16%** / **52.18%**	**65.49%** / **60.60%**	**74.88%** / **69.28%**	**84.45%** / **80.70%**
MCP$(a = 15000)$	9.19% / 8.01%	**18.37%** / 16.21%	**27.59%** / 24.47%	**36.79%** / 32.96%	**45.97%** / 40.97%	55.15% / 49.24%	64.35% / 57.64%	**73.77%** / 68.13%	83.72% / 79.37%
MCP$(a = 10000)$	**9.29%** / 8.23%	**18.45%** / 16.28%	**27.71%** / 24.60%	**36.93%** / **33.05%**	**46.07%** / 41.26%	**55.22%** / 49.32%	**64.40%** / 57.57%	**73.91%** / **68.23%**	**83.85%** / 79.39%
MCP$(a = 5000)$	9.17% / 8.19%	18.25% / 16.11%	27.45% / 24.25%	36.57% / 32.22%	45.75% / 40.39%	54.94% / 48.56%	64.13% / 56.70%	73.75% / 67.59%	**83.92%** / 79.17%
SCAD$(a = 15000)$	9.21% / 8.11%	**18.36%** / 16.21%	27.54% / 24.43%	**36.75%** / 32.57%	45.94% / 40.90%	55.12% / 49.18%	**64.41%** / 57.68%	**73.85%** / 68.10%	**83.80%** / 79.42%
SCAD$(a = 10000)$	9.18% / 8.16%	**18.36%** / 16.54%	**27.60%** / 24.83%	**36.77%** / 32.77%	45.94% / 41.05%	55.10% / 49.04%	64.30% / 57.21%	**73.79%** / 67.98%	83.75% / 79.27%
SCAD$(a = 5000)$	9.06% / 7.78%	18.22% / 15.88%	27.40% / 23.97%	36.54% / 32.07%	45.66% / 39.87%	54.87% / 48.02%	64.08% / 56.01%	73.71% / 67.25%	**83.84%** / 78.76%
(b)	CIFAR 100								
Channel Pruning Ratio	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
ℓ_1	9.18% / 7.46%	18.34% / 15.21%	27.53% / 22.91%	36.69% / 30.44%	45.84% / 37.84%	54.98% / 45.36%	64.12% / 54.09%	73.39% / 65.92%	
$\ell_{3/4}$	**9.19%** / **8.20%**	**18.39%** / **16.12%**	**27.57%** / **24.04%**	**36.76%** / **31.88%**	**45.95%** / **39.91%**	**55.13%** / **47.74%**	**64.33%** / **55.84%**	**73.56%** / **66.30%**	**83.34%** / **79.89%**
$\ell_{1/2}$	**9.20%** / **8.23%**	**18.41%** / **16.41%**	**27.62%** / **24.37%**	**36.85%** / **32.34%**	**46.06%** / **40.58%**	**55.26%** / **48.91%**	**64.44%** / **56.97%**	**73.67%** / **66.98%**	NA
$\ell_{1/4}$	**9.26%*** / **8.33%***	**18.53%*** / **16.76%***	**27.85%*** / **25.00%***	**37.17%*** / **33.70%***	**46.51%*** / **43.03%***	**55.94%*** / **52.75%***	**65.73%*** / **63.59%***	**76.28%*** / **76.02%***	NA
T$\ell_1(a = 10.0)$	**9.19%** / **7.80%**	**18.35%** / 15.19%	**27.55%** / **23.01%**	**36.72%** / **30.60%**	**45.92%** / **38.42%**	**55.08%** / **45.82%**	**64.24%** / 53.94%	**73.49%** / 65.90%	NA
T$\ell_1(a = 1.0)$	**9.26%*** / **8.00%**	**18.46%** / **15.92%**	**27.72%** / **23.79%**	**36.91%** / **31.49%**	**46.15%** / **39.49%**	**55.35%** / **47.34%**	**64.55%** / **55.62%**	**73.78%** / **66.24%**	**83.48%** / **80.01%**
T$\ell_1(a = 0.5)$	**9.25%** / **8.11%**	**18.49%** / **15.98%**	**27.75%** / **24.15%**	**36.98%** / **32.22%**	**46.24%** / **40.44%**	**55.46%** / **48.33%**	**64.71%** / **56.39%**	**73.92%** / **66.20%**	**83.60%*** / **80.15%***
MCP$(a = 15000)$	**9.19%** / **7.72%**	**18.35%** / **15.52%**	27.52% / **23.29%**	36.67% / **30.99%**	45.81% / **38.42%**	**54.99%** / **46.02%**	**64.14%** / **55.17%**	**73.46%** / **66.30%**	**83.35%** / **79.72%**
MCP$(a = 10000)$	9.16% / **7.50%**	18.31% / 15.09%	27.46% / 22.84%	36.61% / **30.61%**	45.79% / **38.36%**	54.94% / **45.92%**	64.10% / **55.76%**	73.37% / **66.94%**	**83.19%** / **79.68%**
MCP$(a = 5000)$	9.16% / **7.53%**	18.32% / 15.00%	27.46% / 22.51%	36.64% / 30.01%	45.78% / **37.87%**	54.93% / **46.00%**	64.12% / **56.81%**	**73.42%** / **67.34%**	**83.46%** / **79.52%**
SCAD$(a = 15000)$	**9.19%** / **7.85%**	**18.36%** / **15.50%**	27.52% / **23.12%**	36.68% / **30.65%**	45.84% / **38.33%**	**54.99%** / **46.03%**	**64.16%** / **55.37%**	**73.45%** / **66.81%**	**83.33%** / **79.72%**
SCAD$(a = 10000)$	9.15% / **7.72%**	18.30% / **15.46%**	27.47% / **23.15%**	36.63% / **30.66%**	45.76% / **38.43%**	54.94% / **46.14%**	**64.14%** / **56.10%**	**73.44%** / **67.22%**	**83.36%** / **79.61%**
SCAD$(a = 5000)$	9.15% / 7.37%	18.31% / 14.96%	27.44% / 22.27%	36.59% / 29.79%	45.76% / 37.50%	54.91% / **45.40%**	64.11% / **55.93%**	**73.42%** / **67.19%**	**83.53%** / **79.75%**
(c)	SVHN								
Channel Pruning Ratio	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90
ℓ_1	9.51% / 9.29%	19.12% / 18.86%	28.61% / 27.83%	38.25% / 37.38%	47.84% / 46.82%	57.48% / 55.88%	67.09% / 65.54%	76.67% / 75.30%	86.15% / 85.06%
$\ell_{3/4}$	**9.63%** / **9.69%**	**19.27%** / **19.42%**	**28.79%** / **28.88%**	**38.42%** / **38.25%**	**48.04%** / **47.95%**	**57.69%** / **57.71%**	**67.25%** / **67.11%**	**76.79%** / **76.51%**	**86.38%** / **85.87%**
$\ell_{1/2}$	**9.62%** / **9.38%**	**19.21%** / **19.20%**	**28.81%** / **28.68%**	**38.44%** / **38.46%**	**48.08%** / **47.85%**	**57.75%** / **57.54%**	**67.43%** / **67.40%**	**77.05%** / **76.96%**	**86.68%** / **86.41%**
$\ell_{1/4}$	**9.68%*** / **9.88%***	**19.34%** / **19.46%***	**29.05%*** / **29.40%***	**38.74%*** / **39.33%***	**48.42%*** / **49.00%***	**58.12%*** / **58.71%***	**67.92%*** / **68.69%***	**77.81%*** / **78.96%***	**87.81%*** / **89.44%***
T$\ell_1(a = 10.0)$	**9.57%** / **9.48%**	**19.13%** / **19.05%**	**28.72%** / **28.73%**	**38.36%** / **38.13%**	**47.87%** / **47.49%**	**57.51%** / **56.91%**	67.06% / **66.32%**	76.64% / **75.74%**	**86.29%** / **85.54%**
T$\ell_1(a = 1.0)$	**9.58%** / **9.33%**	**19.24%** / **19.26%**	**28.92%** / **28.77%**	**38.58%** / **38.59%**	**48.20%** / **48.03%**	**57.82%** / **57.66%**	**67.44%** / **66.97%**	**77.01%** / **76.50%**	**86.57%** / **86.17%**
T$\ell_1(a = 0.5)$	**9.62%** / 9.29%	**19.19%** / 18.82%	**28.81%** / **28.37%**	**38.51%** / **37.98%**	**48.16%** / **47.64%**	**57.83%** / **57.76%**	**67.46%** / **67.27%**	**77.03%** / **77.01%**	**86.70%** / **86.55%**
MCP$(a = 15000)$	**9.65%** / **9.52%**	**19.31%** / **19.09%**	**28.89%** / **28.73%**	**38.40%** / **38.03%**	**47.88%** / **47.53%**	57.44% / **56.81%**	67.05% / **66.62%**	76.60% / **76.05%**	NA
MCP$(a = 10000)$	9.51% / **9.42%**	19.02% / **18.92%**	28.60% / **28.36%**	38.22% / **37.67%**	47.73% / **47.15%**	57.26% / **56.59%**	66.95% / **65.99%**	76.61% / **75.71%**	86.14% / **85.29%**
MCP$(a = 5000)$	**9.55%** / **9.44%**	**19.14%** / **18.89%**	**28.70%** / **28.36%**	38.25% / **37.66%**	**47.89%** / **47.26%**	57.48% / **56.57%**	67.02% / **66.10%**	76.58% / **75.69%**	86.10% / 84.97%
SCAD$(a = 15000)$	**9.55%** / **9.31%**	19.09% / 18.75%	**28.71%** / **28.52%**	**38.26%** / **38.02%**	47.84% / **47.30%**	57.47% / **56.49%**	**67.13%** / **66.27%**	76.57% / **75.62%**	NA
SCAD$(a = 10000)$	**9.66%** / **9.82%**	**19.37%*** / **19.25%**	**28.88%** / **28.99%**	**38.46%** / **38.36%**	**48.06%** / **47.87%**	**57.53%** / **57.31%**	**67.13%** / **66.95%**	76.61% / **76.55%**	NA
SCAD$(a = 5000)$	**9.55%** / **9.31%**	19.09% / 18.75%	**28.71%** / **28.52%**	**38.26%** / **38.02%**	47.84% / **47.30%**	57.47% / **56.49%**	**67.13%** / **66.27%**	76.57% / **75.62%**	NA

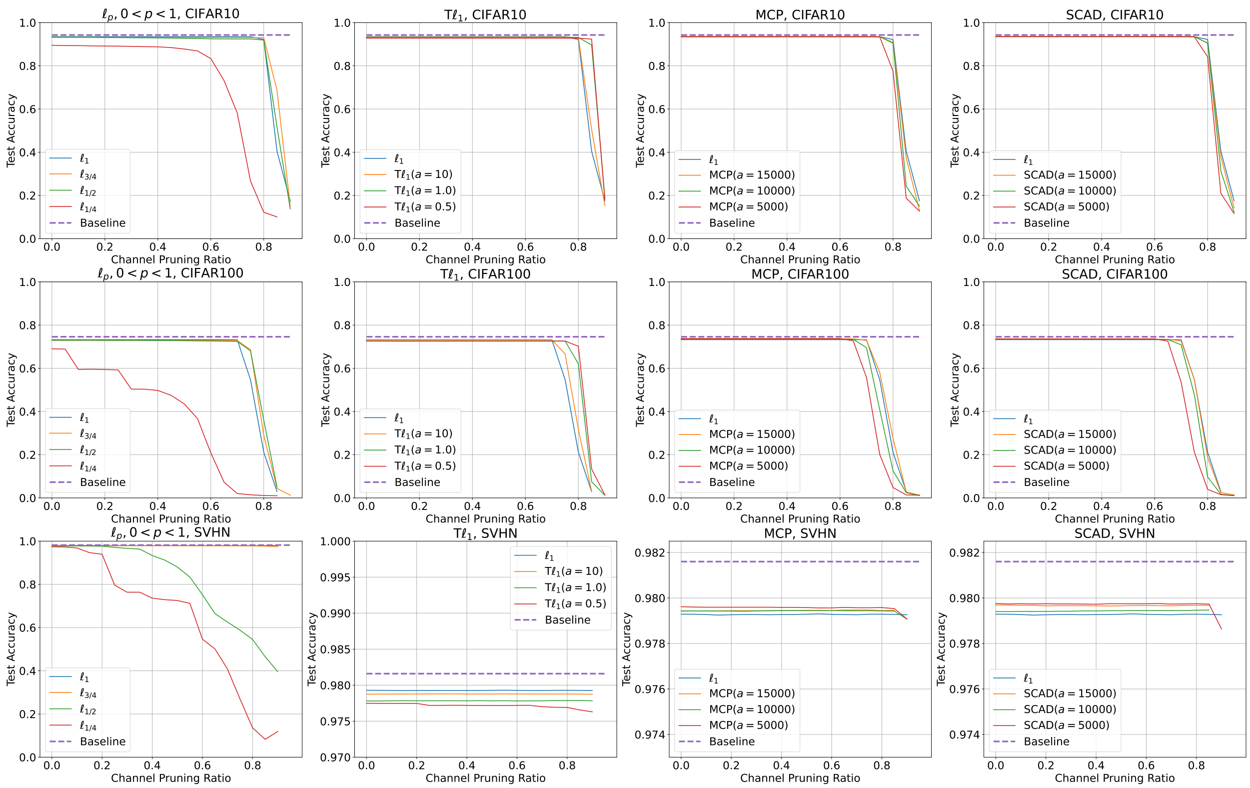


Figure 3.4: Effect of channel pruning on the mean test accuracy of five runs of DenseNet-40 on CIFAR 10/100 and SVHN. Baseline refers to the mean test accuracy of the unregularized model that is not pruned. Baseline accuracies are 94.25% for CIFAR 10, 74.58% for CIFAR 100, and 98.16% for SVHN.

ℓ_1 because their test accuracies drop at higher channel pruning ratios than ℓ_1's.

For SVHN, Table 3.3c shows that ℓ_p and Tℓ_1 have larger parameter/FLOPs pruning percentages than ℓ_1 across different levels of the channel pruning ratios. In general, MCP also saves more on parameters and FLOPs for channel pruning ratio up to 0.50. After 0.50, MCP saves more on only FLOPs. SCAD also generally saves more on FLOPs than ℓ_1. According to Figure 3.4, the test accuracy remains nearly constant for channel pruning ratio up to 0.90 for all regularizers except for $\ell_{1/4}$ and $\ell_{1/2}$. We also observe that across different channel pruning ratios, Tℓ_1 has slightly worse test accuracy than ℓ_1 while MCP and SCAD mostly have better test accuracies than ℓ_1.

In summary, we observe that ℓ_p and Tℓ_1 reduce more parameters and FLOPs required than ℓ_1 after channel pruning, while MCP and SCAD save more on only FLOPs specifically for CIFAR 100 and SVHN. Like for VGG-19, T$\ell_1(a = 0.5)$ is the most robust against channel pruning, whereas $\ell_{1/4}$ is the most sensitive to it.

ResNet-164. ResNet-164 has about 1.70 million parameters and requires 5.00×10^8 FLOPs. Table 3.4 records the mean percentages of parameters/FLOPs pruned for different channel pruning ratios. Figure 3.5 shows the effect of channel pruning on the test accuracies of the regularized models.

On CIFAR 10, Table 3.4a shows a quite noticeable difference in the numbers of parameters and FLOPs pruned between ℓ_1 and ℓ_p or T$\ell_1(a = 0.5, 1.0)$. For example, $\ell_{1/2}$ saves at least 10% more weight parameters and at least 8% more FLOPs than ℓ_1 at channel pruning ratio 0.40 and above. On the other hand, SCAD and MCP are outperformed by ℓ_1 in percentages of parameters/FLOPs pruned. According to Figure 3.5, most of the regularizers do not suffer a significant drop in test accuracy when large number of channels are pruned.

On CIFAR 100, according to Table 3.4b, $\ell_p(p = 1/4, 1/2)$ and T$\ell_1(a = 0.5, 1.0)$ prune at least 3% more parameters and at least 1% more FLOPs than ℓ_1. However, MCP and SCAD

Table 3.4: Effect of channel pruning on the mean pruned parameter / FLOPs percentages (%) on ResNet-164 trained on (a) CIFAR 10, (b) CIFAR 100, and (c) SVHN. The mean is computed from five runs for each regularizer. For each channel pruning ratio, **bold** indicates outperforming ℓ_1; * indicates best value; and NA indicates at least one of the five models is over-pruned.

(a)	CIFAR 10						
Channel Pruning Ratio	0.10	0.20	0.30	0.40	0.50	0.60	0.70
ℓ_1	8.57% / 8.37%	16.67% / 16.62%	24.44% / 24.29%	31.39% / 31.57%	38.50% / 38.31%	46.43% / 45.54%	NA
$\ell_{3/4}$	**10.87%** / **10.07%**	**21.51%** / **19.54%**	**31.23%** / **28.32%**	**40.07%** / **36.66%**	**47.86%** / **43.96%**	**55.15%** / **50.71%**	**62.88%** / **58.36%**
$\ell_{1/2}$	**12.13%** / **10.96%**	**22.88%** / **21.41%**	**33.23%** / **31.09%**	**42.54%** / **39.81%**	**50.88%** / **48.02%**	**58.17%** / **55.12%**	**64.88%** / **61.46%**
$\ell_{1/4}$	**14.26%*** / **13.00%***	**26.44%*** / **24.50%***	**37.64%*** / **34.93%***	**47.82%*** / **44.89%***	**57.10%*** / **54.47%***	**65.58%*** / **64.27%***	NA
Tℓ_1($a=10.0$)	**8.99%** / **8.56%**	**17.92%** / **16.88%**	**25.80%** / 24.18%	**33.26%** / 31.14%	**40.50%** / 38.27%	**47.52%** / 44.88%	NA
Tℓ_1($a=1.0$)	**11.99%** / **10.74%**	**22.86%** / **20.72%**	**33.08%** / **29.61%**	**42.64%** / **38.44%**	**51.03%** / **46.02%**	**58.52%** / **53.03%**	NA
Tℓ_1($a=0.5$)	**12.51%** / **11.29%**	**23.95%** / **21.61%**	**34.43%** / **31.01%**	**44.10%** / **39.82%**	**52.93%** / **47.65%**	**60.81%** / **54.86%**	**67.15%** / **61.20%**
MCP($a=15000$)	8.07% / 7.90%	16.00% / 15.54%	23.46% / 22.69%	30.50% / 29.08%	36.84% / 35.05%	**47.73%** / **45.63%**	NA
MCP($a=10000$)	7.10% / 7.61%	13.90% / 14.43%	20.58% / 21.06%	26.87% / 27.32%	32.74% / 33.29%	NA	NA
MCP($a=5000$)	4.19% / 5.55%	8.64% / 10.97%	12.85% / 16.08%	17.06% / 21.00%	23.89% / 29.34%	NA	NA
SCAD($a=15000$)	7.71% / 7.65%	15.53% / 15.44%	22.58% / 22.83%	29.51% / 29.44%	36.44% / 35.82%	**47.47%** / **46.15%**	NA
SCAD($a=10000$)	7.19% / 7.33%	13.99% / 14.30%	20.51% / 20.88%	26.71% / 27.26%	32.79% / 33.27%	NA	NA
SCAD($a=5000$)	4.62% / 5.68%	8.98% / 11.02%	13.24% / 16.23%	17.45% / 21.35%	23.94% / 29.47%	NA	NA

(b)	CIFAR 100						
Channel Pruning Ratio	0.10	0.20	0.30	0.40	0.50	0.60	0.70
ℓ_1	4.01% / 7.42%	7.98% / 14.48%	11.88% / 20.94%	15.72% / 26.88%	NA	NA	NA
$\ell_{3/4}$	**4.95%** / **7.55%**	**9.84%** / **14.90%**	**14.70%** / **22.08%**	**19.15%** / **28.15%**	**24.58%** / **35.33%**	NA	NA
$\ell_{1/2}$	**5.72%** / **8.53%**	**11.35%** / **16.51%**	**16.66%** / **23.82%**	**21.86%** / **30.73%**	**26.64%** / **36.87%**	NA	NA
$\ell_{1/4}$	**11.13%*** / **11.46%***	**20.98%*** / **21.85%***	**30.00%*** / **31.48%***	**37.85%*** / **41.10%***	NA	NA	NA
Tℓ_1($a=10.0$)	**4.08%** / 7.07%	**8.29%** / 13.87%	**12.36%** / 20.03%	**16.36%** / 25.92%	NA	NA	NA
Tℓ_1($a=1.0$)	**6.08%** / **8.09%**	**11.96%** / **15.67%**	**17.38%** / **22.93%**	**22.99%** / **29.81%**	**28.27%** / **36.36%**	NA	NA
Tℓ_1($a=0.5$)	**6.37%** / **9.25%**	**12.68%** / **17.30%**	**18.82%** / **25.19%**	**24.87%** / **31.89%**	**30.54%*** / **38.63%***	NA	NA
MCP($a=15000$)	3.64% / 6.64%	7.25% / 12.89%	10.92% / 18.99%	15.22% / 25.05%	NA	NA	NA
MCP($a=10000$)	3.51% / 6.65%	7.01% / 12.53%	10.42% / 18.48%	14.45% / 24.81%	NA	NA	NA
MCP($a=5000$)	3.32% / 6.37%	6.52% / 12.13%	9.67% / 17.58%	NA	NA	NA	NA
SCAD($a=15000$)	3.62% / 6.56%	7.20% / 13.01%	10.88% / 19.36%	15.16% / 25.79%	NA	NA	NA
SCAD($a=10000$)	3.53% / 6.36%	6.99% / 12.61%	10.36% / 18.46%	14.54% / 25.33%	NA	NA	NA
SCAD($a=5000$)	3.31% / 5.92%	6.59% / 11.83%	9.77% / 17.33%	14.15% / 25.57%	NA	NA	NA

(c)	SVHN						
Channel Pruning Ratio	0.10	0.20	0.30	0.40	0.50	0.60	0.70
ℓ_1	12.32% / 17.02%*	22.70% / 29.19%	32.63% / 41.26%	41.88% / 52.39%	50.14% / 62.14%	NA	NA
$\ell_{3/4}$	**13.09%** / 15.50%	**25.49%** / **29.87%**	**36.84%** / **42.16%**	**47.02%** / **53.46%**	**55.77%** / **63.17%**	NA	NA
$\ell_{1/2}$	**13.80%** / 15.21%	**26.62%** / **29.57%**	**38.20%** / **42.21%**	**48.80%** / **53.60%**	**58.45%** / **63.91%***	**66.65%** / **72.62%**	NA
$\ell_{1/4}$	**15.16%*** / 15.61%	**29.05%*** / **29.73%**	**41.52%*** / **42.43%**	**52.47%*** / **53.73%***	**62.39%*** / **63.68%**	**71.50%*** / **72.79%***	NA
Tℓ_1($a=10.0$)	12.13% / 16.66%	**23.13%** / **30.10%***	**33.11%** / **41.50%**	**42.70%** / **52.97%**	**50.87%** / **62.07%**	**58.16%** / **69.81%**	NA
Tℓ_1($a=1.0$)	**13.45%** / 15.39%	**25.82%** / **29.90%**	**37.29%** / **42.59%**	**47.70%** / **53.87%**	**56.79%** / **63.43%**	NA	NA
Tℓ_1($a=0.5$)	**14.35%** / 15.83%	**26.94%** / **29.53%**	**38.69%** / **42.68%***	**48.83%** / **53.70%**	**58.31%** / **63.81%**	**66.44%** / **72.28%**	NA
MCP($a=15000$)	12.07% / 15.25%	**23.19%** / 28.99%	**32.89%** / 40.96%	41.67% / 51.50%	49.89% / 60.89%	**57.23%** / **68.84%**	NA
MCP($a=10000$)	11.39% / 15.19%	22.09% / 28.56%	32.33% / 40.67%	41.32% / 51.23%	49.08% / 60.14%	NA	NA
MCP($a=5000$)	9.90% / 13.98%	19.13% / 26.99%	27.85% / 38.51%	35.80% / 48.73%	43.23% / 57.77%	NA	NA
SCAD($a=15000$)	11.45% / 15.70%	22.01% / 28.82%	32.14% / 40.65%	41.05% / 51.61%	49.47% / 61.02%	**56.76%** / **68.83%**	NA
SCAD($a=10000$)	12.30% / 16.86%	22.63% / 29.36%	32.39% / 40.89%	41.23% / 51.75%	NA	NA	NA
SCAD($a=5000$)	10.42% / 15.04%	19.82% / 27.80%	28.52% / 38.81%	36.76% / 49.44%	NA	NA	NA

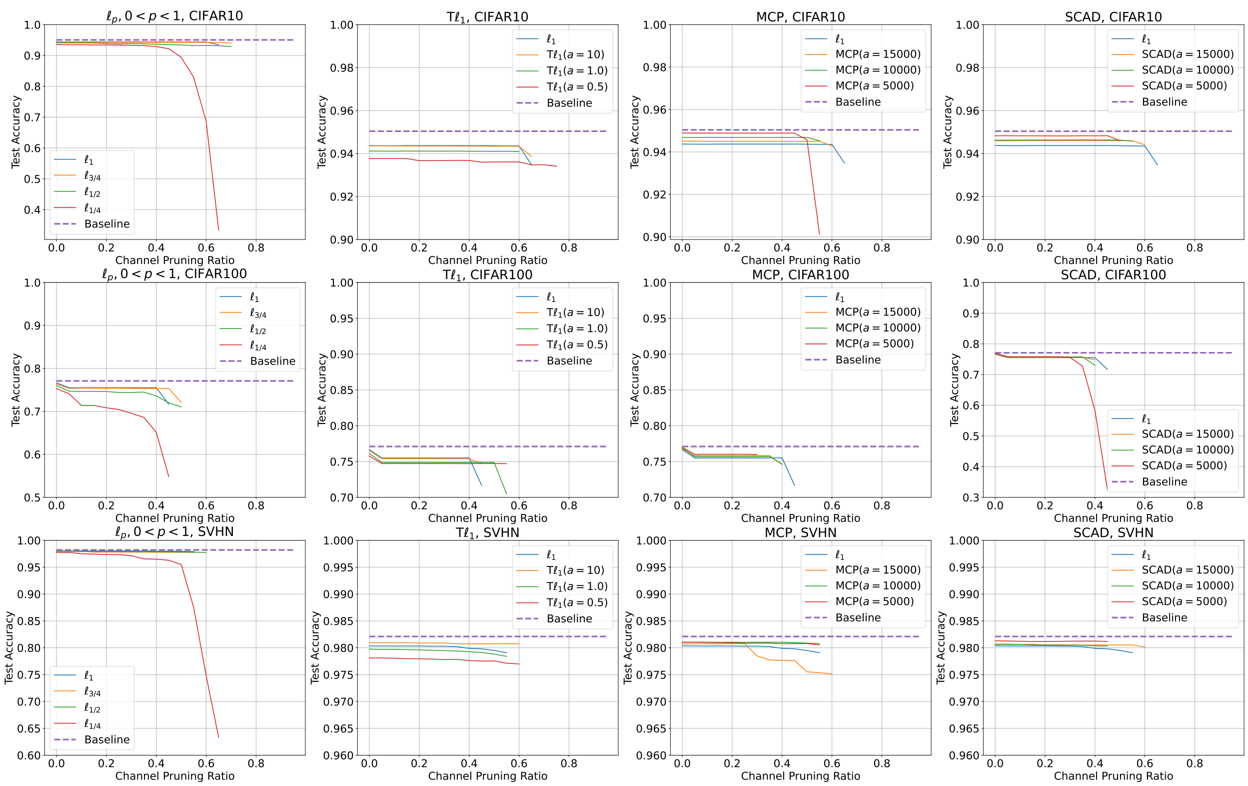


Figure 3.5: Effect of channel pruning on the mean test accuracy of five runs of ResNet-164 on CIFAR 10/100 and SVHN. Baseline refers to the mean test accuracy of the unregularized model that is not pruned. Baseline accuracies are 95.04% for CIFAR 10, 77.10% for CIFAR 100, and 98.21% for SVHN.

are outperformed by ℓ_1 again for percentages of parameters and FLOPs pruned. In Figure 3.5, we observe that most of the regularizers are robust against channel pruning since the test accuracies do not drop severely at higher channel pruning ratios.

On SVHN, Table 3.4c reports that ℓ_p and Tℓ_1 save more parameters and FLOPs than ℓ_1 for channel pruning ratio at least 0.20, while that MCP and SCAD do not. Like for CIFAR 10 and CIFAR 100, most regularizers yield models whose test accuracies are robust against channel pruning according to Figure 3.5.

In general, the test accuracies of ResNet-164 models with any regularizers, except for $\ell_{1/4}$, are stable against channel pruning. In addition, ℓ_p and T$\ell_1(a = 0.5, 1.0)$ prune more parameters and FLOPs than ℓ_1. Overall, MCP and SCAD do not perform well on ResNet-164.

3.3.4 Retraining After Pruning

Because the test accuracy drops after channel pruning for VGG-19 and DenseNet-40 trained on CIFAR 10/100, we retrain the models without regularization on the scaling factors and examine whether or not the original test accuracy is recovered. For brevity, we analyze ℓ_1 and the nonconvex regularizers whose possible channel pruning percentages are at least the same as ℓ_1's.

VGG-19. The results for VGG-19 on CIFAR 10/100 are presented in Table 3.5. Generally, we observe that the test accuracy after retraining is better than the original test accuracy before channel pruning and retraining. For CIFAR 10, after the models are retrained with 70% of their channels pruned, only ℓ_1 $\ell_{3/4}$, T$\ell_1(a = 1.0, 10.0)$, MCP $(a = 10000, 15000)$, and SCAD $(a = 10000, 15000)$ exceed the baseline test accuracy of 93.83%. Among the nonconvex regularizers, $\ell_{3/4}$ and T$\ell_1(a = 1.0, 10.0)$ yield more compressed models in terms of both parameters and FLOPS but have slightly lower test accuracies than ℓ_1. On the other

Table 3.5: Results from five retrained VGG-19 on CIFAR 10/100 after pruning. Baseline refers to the VGG-19 model trained without regularization on the scaling factors.

	Number of Parameters/FLOPs	Percentage of Parameters/FLOPs Pruned (%)	Mean Test Accuracy before Retraining (%)	Mean Test Accuracy after Retraining (%)
Baseline	20.04M/7.97 $\times 10^8$	0.00/0.00	93.83	N/A
ℓ_1 (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	93.63	N/A
ℓ_1 (70% Pruned)	2.24M/3.83 $\times 10^8$	88.81/51.93	28.28	93.91
$\ell_{3/4}$ (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	93.53	N/A
$\ell_{3/4}$ (70% Pruned)	2.07M/3.59 $\times 10^8$	89.69/54.96	88.87	93.90
$\ell_{3/4}$ (75% Pruned)	1.79M/3.43 $\times 10^8$	91.06/57.00	16.18	93.79
$\ell_{1/2}$ (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	93.57	N/A
$\ell_{1/2}$ (70% Pruned)	2.00M/3.50 $\times 10^8$	90.01/56.12	40.07	93.77
$\ell_{1/2}$ (75% Pruned)	1.66M/3.25 $\times 10^8$	91.70/59.20	13.65	93.82
$\ell_{1/4}$ (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	86.97	N/A
$\ell_{1/4}$ (70% Pruned)	1.58M/1.44 $\times 10^8$	92.14/81.89	47.59	92.15
$\ell_{1/4}$ (90% Pruned)	0.19M/0.13 $\times 10^8$	99.05/98.32	10.00	81.57
T$\ell_1(a = 10.0)$ (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	93.64	N/A
T$\ell_1(a = 10.0)$ (70% Pruned)	2.19M/3.77 $\times 10^8$	89.06/52.75	47.70	93.86
T$\ell_1(a = 10.0)$ (75% Pruned)	1.84M/3.49 $\times 10^8$	90.82/56.19	10.00	93.72
T$\ell_1(a = 1.0)$ (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	93.55	N/A
T$\ell_1(a = 1.0)$ (70% Pruned)	1.93M/3.39 $\times 10^8$	90.35/57.43	93.54	93.86
T$\ell_1(a = 1.0)$ (75% Pruned)	1.66M/3.24 $\times 10^8$	91.71/59.29	86.83	93.82
T$\ell_1(a = 0.5)$ (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	93.15	N/A
T$\ell_1(a = 0.5)$ (70% Pruned)	1.83M/3.20 $\times 10^8$	90.88/59.84	93.14	93.75
T$\ell_1(a = 0.5)$ (75% Pruned)	1.53M/3.05 $\times 10^8$	92.38/61.74	92.38	93.77
MCP ($a = 15000$) (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	93.65	N/A
MCP ($a = 15000$) (70% Pruned)	2.29M/3.93 $\times 10^8$	88.58/50.69	47.18	93.97
MCP ($a = 15000$) (75% Pruned)	1.89M/3.58 $\times 10^8$	90.58/55.04	10.00	93.68
MCP ($a = 10000$) (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	93.69	N/A
MCP ($a = 10000$) (70% Pruned)	2.28M/3.95 $\times 10^8$	88.63/50.49	40.24	94.12
MCP ($a = 10000$) (75% Pruned)	1.89M/3.62 $\times 10^8$	90.56/54.54	10.00	93.73
SCAD ($a = 15000$) (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	93.64	N/A
SCAD ($a = 15000$) (70% Pruned)	2.26M/3.93 $\times 10^8$	88.71/50.70	52.72	93.94
SCAD ($a = 15000$) (75% Pruned)	1.87M/3.59 $\times 10^8$	90.65/54.97	10.00	93.91
SCAD ($a = 10000$) (0% Pruned)	20.04M/7.97 $\times 10^8$	0.00/0.00	93.60	N/A
SCAD ($a = 10000$) (70% Pruned)	2.29M/3.95 $\times 10^8$	88.57/50.43	55.25	93.88

(a) CIFAR 10

	Number of Parameters/FLOPs	Percentage of Parameters/FLOPs Pruned (%)	Mean Test Accuracy before Retraining (%)	Mean Test Accuracy after Retraining (%)
Baseline	20.08M/7.97 $\times 10^8$	0.00/0.00	72.73	N/A
ℓ_1 (0% Pruned)	20.08M/7.97 $\times 10^8$	0.00/0.00	72.57	N/A
ℓ_1 (45% Pruned)	5.67M/5.26 $\times 10^8$	71.78/34.00	51.16	73.44
ℓ_1 (55% Pruned)	4.31M/4.89 $\times 10^8$	78.53/38.66	1.00	72.98
$\ell_{3/4}$ (0% Pruned)	20.08M/7.97 $\times 10^8$	0.00/0.00	72.14	N/A
$\ell_{3/4}$ (45% Pruned)	5.49M/5.04 $\times 10^8$	72.68/36.75	71.76	73.24
$\ell_{3/4}$ (55% Pruned)	4.10M/4.76 $\times 10^8$	79.59/40.28	3.40	73.26
$\ell_{1/2}$ (0% Pruned)	20.08M/7.97 $\times 10^8$	0.00/0.00	72.06	N/A
$\ell_{1/2}$ (45% Pruned)	5.38M/5.03 $\times 10^8$	73.21/36.95	71.27	73.34
$\ell_{1/2}$ (60% Pruned)	3.40M/4.48 $\times 10^8$	83.07/43.82	1.08	71.59
$\ell_{1/4}$ (0% Pruned)	20.08M/7.97 $\times 10^8$	0.00/0.00	70.95	N/A
$\ell_{1/4}$ (45% Pruned)	5.30M/4.76 $\times 10^8$	73.59/40.26	22.70	72.50
$\ell_{1/4}$ (80% Pruned)	0.69M/1.05 $\times 10^8$	96.54/86.86	1.00	46.97
T$\ell_1(a = 10.0)$ (0% Pruned)	20.08M/7.97 $\times 10^8$	0.00/0.00	72.36	N/A
T$\ell_1(a = 10.0)$ (45% Pruned)	5.53M/5.18 $\times 10^8$	72.45/34.95	69.35	73.39
T$\ell_1(a = 10.0)$ (55% Pruned)	4.21M/4.85 $\times 10^8$	79.05/39.19	1.46	73.17
T$\ell_1(a = 1.0)$ (0% Pruned)	20.08M/7.97 $\times 10^8$	0.00/0.00	72.07	N/A
T$\ell_1(a = 1.0)$ (45% Pruned)	5.39M/4.87 $\times 10^8$	73.16/38.89	72.07	73.03
T$\ell_1(a = 1.0)$ (60% Pruned)	3.43M/4.47 $\times 10^8$	82.90/43.94	1.84	73.06
T$\ell_1(a = 0.5)$ (0% Pruned)	20.08M/7.97 $\times 10^8$	0.00/0.00	71.63	N/A
T$\ell_1(a = 0.5)$ (45% Pruned)	5.29M/4.74 $\times 10^8$	73.66/40.48	71.63	72.69
T$\ell_1(a = 0.5)$ (60% Pruned)	3.19M/4.21 $\times 10^8$	84.09/47.15	66.50	72.81
MCP ($a = 15000$) (0% Pruned)	20.08M/7.97 $\times 10^8$	0.00/0.00	72.26	N/A
MCP ($a = 15000$) (45% Pruned)	5.66M/5.27 $\times 10^8$	71.82/33.87	66.14	73.68
MCP ($a = 15000$) (55% Pruned)	4.30M/4.92 $\times 10^8$	78.58/38.21	1.00	72.94
SCAD ($a = 15000$) (0% Pruned)	20.08M/7.97 $\times 10^8$	0.00/0.00	72.50	N/A
SCAD ($a = 15000$) (45% Pruned)	5.64M/5.26 $\times 10^8$	71.89/33.99	65.72	73.61
SCAD ($a = 15000$) (55% Pruned)	4.32M/4.90 $\times 10^8$	78.48/38.49	1.00	72.67
SCAD ($a = 10000$) (0% Pruned)	20.08M/7.97 $\times 10^8$	0.00/0.00	72.33	N/A
SCAD ($a = 10000$) (45% Pruned)	5.72M/5.32 $\times 10^8$	71.50/33.21	64.98	73.52
SCAD ($a = 10000$) (55% Pruned)	4.37M/4.94 $\times 10^8$	78.22/37.99	1.00	71.98

(b) CIFAR 100

hand, MCP ($a = 15000, 10000$) and SCAD ($a = 15000$) are slightly less compressed than ℓ_1 but have better test accuracies. When 75% of the channels are pruned, their retrained test accuracies decrease slightly due to compressing the models further. Among the nonconvex regularizers, the test accuracy for SCAD ($a = 15000$) is better than the baseline. Moreover, SCAD ($a = 15000$) with 75% of its channels pruned requires less parameters and FLOPS than ℓ_1 with 70% of its channels pruned. For $\ell_{1/4}$, when 90% of the channels are pruned, at least 98% of parameters and FLOPs are pruned, but the test accuracy after retraining is 81.57%. For CIFAR 100, with 45% of the channels pruned, all of the regularizers except for $\ell_{1/4}$ and T$\ell_1(a = 0.5)$ attain better test accuracies than the baseline accuracy of 72.73%. Similar to CIFAR 10, $\ell_{3/4}, \ell_{1/2}$, and T$\ell_1(a = 1.0, 10.0)$ have slightly lower test accuracies than ℓ_1 but have better compression. MCP and SCAD have better test accuracies than ℓ_1 with similar parameter and FLOP compression. When more channels are pruned, most of the regularizers suffer a slight decrease in retrained test accuracies. Only $\ell_{3/4}$ with 55% channels pruned and T$\ell_1(a = 0.5, 1.0)$ with 60% channels pruned experience a modest improvement in test accuracy, but their test accuracies exceed the baseline test accuracy and ℓ_1's test accuracy with 55% channels pruned.

Overall, for $\ell_p(p = 1/2, 3/4)$ and T$\ell_1(a = 0.5, 1.0)$, the retrained models, despite being more compressed than their ℓ_1 counterparts, have slightly lower test accuracies. However, MCP and SCAD have similar compression as ℓ_1 but with better test accuracies after retraining.

DenseNet-40. Table 3.6 reports the results for DenseNet-40 on CIFAR 10/100. Overall, the baseline accuracy is better than all of the retrained test accuracies, but the differences are at most 3.07% for CIFAR 10 and at most 6.82% for CIFAR 100. For CIFAR 10, when 82.5% of the channels are pruned, only MCP ($a = 10000$) and SCAD ($a = 10000$) have better test accuracies than ℓ_1 with similar compression in parameters and FLOPs. For $\ell_p(p = 1/2, 3/4)$ and T$\ell_1(a = 1.0)$, their retrained test accuracies are only slightly lower by at most 0.20%, but this is at the cost of better compression. When 90% of the channels are pruned, the retrained test accuracies decrease slightly more into the range of 91%-92%. Only $\ell_{3/4}$ and $\mathrm{T}\ell_1(a = 0.5, 1.0)$ have better test accuracies than ℓ_1 with much better compression. For CIFAR 100, when 75% of the channels are pruned, $\ell_{3/4}$, $\mathrm{T}\ell_1(a = 10.0)$, MCP, and SCAD have at least the same test accuracies as ℓ_1 with better compression in parameters and FLOPs. However, increasing the channel pruning percentage to 90% causes their retrained test accuracies to deteriorate. As a result, none of the models is able to exceed the test accuracy of the ℓ_1-regularized models retrained with 85% of their channels pruned. For $\ell_{1/2}$ and $\mathrm{T}\ell_1(a = 10.0)$, when 85% of the channels are pruned, their test accuracies exceed ℓ_1. In general, pruning channels for DenseNet at the highest percentage possible can be detrimental to the retrained test accuracy. When channels are pruned at intermediate levels, the nonconvex regularizers can have better retrained test accuracies and/or better compression than ℓ_1.

3.3.5 Scaling Factor Analysis

In order to better understand how ℓ_1 and the nonconvex regularizers affect the scaling factors γ, we plot histograms of the counts of the $\log_{10}(|\gamma|)$ averaged from the five models trained for each model and regularizer. Figures 3.6-3.14 provide the histograms while Table 3.7 records the average number of scaling factors whose magnitudes are less than 10^{-6} and more than 10^{-6}. The value 10^{-6} is chosen because generally, any value below it has negligible effect on the numerical computation [1].

For CIFAR 10, Figures 3.6-3.8 show the histograms while Table 3.7a provides the average counts of the scaling factors based on their magnitudes. For all three networks, we observe the following phenomena. MCP and SCAD have similar scaling factor distributions as ℓ_1 across all given values of a. Moreover, MCP, SCAD, and ℓ_1 have similar number of scaling factors whose magnitudes are less than 10^{-6} as verified by Table 3.7a. This may explain why their compression rates are similar to ℓ_1 in our earlier analyses. For ℓ_p, we see that $\ell_{3/4}$

Table 3.6: Results from five retrained DenseNet-40 on CIFAR 10/100 after pruning. Baseline refers to the DenseNet-40 model trained without regularization on the scaling factors.

	Number of Parameters/FLOPs	Percentage of Parameters/FLOPs Pruned (%)	Mean Test Accuracy before Retraining (%)	Mean Test Accuracy after Retraining (%)
Baseline	1.02M/5.33 × 10^8	0.00/0.00	94.25	N/A
ℓ_1 (0 % Pruned)	1.02M/5.33 × 10^8	0.00/0.00	93.46	N/A
ℓ_1 (82.5% Pruned)	0.24M/1.54 × 10^8	76.21/71.20	78.27	93.46
ℓ_1 (90% Pruned)	0.17M/1.08 × 10^8	83.76/79.75	17.47	91.42
$\ell_{3/4}$ (0% Pruned)	1.02M/5.33 × 10^8	0.00/0.00	93.19	N/A
$\ell_{3/4}$ (82.5% Pruned)	0.24M/1.53 × 10^8	76.57/71.34	90.17	93.33
$\ell_{3/4}$ (90% Pruned)	0.16M/1.06 × 10^8	84.02/80.07	15.06	91.54
$\ell_{1/2}$ (0% Pruned)	1.02M/5.33 × 10^8	0.00/0.00	93.28	N/A
$\ell_{1/2}$ (82.5% Pruned)	0.25M/1.51 × 10^8	76.84/71.76	83.17	93.43
$\ell_{1/2}$ (90% Pruned)	0.16M/1.06 × 10^8	84.36/80.13	13.76	91.31
$\ell_{1/4}$ (0% Pruned)	1.02M/5.33 × 10^8	0.00/0.00	89.48	N/A
$\ell_{1/4}$ (82.5% Pruned)	0.21M/1.14 × 10^8	79.81/78.63	11.29	91.68
$\ell_{1/4}$ (85% Pruned)	0.18M/0.98 × 10^8	82.57/81.64	10.05	91.44
Tℓ_1(a = 10.0) (0% Pruned)	1.02M/5.33 × 10^8	0.00/0.00	93.30	N/A
Tℓ_1(a = 10.0) (82.5% Pruned)	0.24M/1.54 × 10^8	76.33/71.10	83.24	93.38
Tℓ_1(a = 10.0) (90% Pruned)	0.16M/1.08 × 10^8	83.89/79.72	15.35	91.37
Tℓ_1(a = 1.0) (0% Pruned)	1.02M/5.33 × 10^8	0.00/0.00	93.16	N/A
Tℓ_1(a = 1.0) (82.5% Pruned)	0.24M/1.53 × 10^8	76.80/71.35	93.17	93.26
Tℓ_1(a = 1.0) (90% Pruned)	0.16M/1.06 × 10^8	84.23/80.19	18.91	91.70
Tℓ_1(a = 0.5) (0% Pruned)	1.02M/5.33 × 10^8	0.00/0.00	92.78	N/A
Tℓ_1(a = 0.5) (82.5% Pruned)	0.23M/1.50 × 10^8	77.21/71.83	92.74	93.05
Tℓ_1(a = 0.5) (90% Pruned)	0.16M/1.03 × 10^8	84.45/80.70	18.12	91.69
MCP(a = 15000) (0% Pruned)	1.02M/5.33 × 10^8	0.00/0.00	93.48	N/A
MCP(a = 15000)(82.5% Pruned)	0.24M/1.55 × 10^8	76.23/71.00	92.74	93.44
MCP(a = 15000) (90% Pruned)	0.17M/1.10 × 10^8	83.72/79.37	12.92	91.31
MCP(a = 10000) (0% Pruned)	1.02M/5.33 × 10^8	0.00/0.00	93.41	N/A
MCP(a = 10000) (82.5% Pruned)	0.24M/1.53 × 10^8	76.37/71.23	67.36	93.53
MCP(a = 10000) (90% Pruned)	0.16M/1.10 × 10^8	83.85/79.39	15.08	91.24
SCAD(a = 15000) (0% Pruned)	1.02M/5.33 × 10^8	0.00/0.00	93.48	N/A
SCAD(a = 15000) (82.5% Pruned)	0.24M/1.54 × 10^8	76.28/71.02	71.33	93.42
SCAD(a = 15000) (90% Pruned)	0.17M/1.10 × 10^8	83.80/79.42	14.21	91.26
SCAD(a = 10000) (0% Pruned)	1.02M/5.33 × 10^8	0.00/0.00	93.52	N/A
SCAD(a = 10000) (82.5% Pruned)	0.24M/1.55 × 10^8	76.25/70.93	71.49	93.49
SCAD(a = 10000) (90% Pruned)	0.17M/1.10 × 10^8	83.75/79.27	12.27	91.18

(a) CIFAR 10

	Number of Parameters/FLOPs	Percentage of Parameters/FLOPs Pruned (%)	Mean Test Accuracy before Retraining (%)	Mean Test Accuracy after Retraining (%)
Baseline	1.06M/5.33 × 10^8	0.00/0.00	74.58	N/A
ℓ_1 (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	73.24	N/A
ℓ_1 (75% Pruned)	0.35M/2.14 × 10^8	68.73/59.89	54.68	73.73
ℓ_1 (85% Pruned)	0.23M/1.46 × 10^8	78.08/72.60	2.94	72.40
$\ell_{3/4}$ (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	72.97	N/A
$\ell_{3/4}$ (75% Pruned)	0.33M/2.11 × 10^8	68.93/60.40	68.60	73.75
$\ell_{3/4}$ (90% Pruned)	0.18M/1.07 × 10^8	83.34/79.89	1.23	69.33
$\ell_{1/2}$ (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	72.98	N/A
$\ell_{1/2}$ (75% Pruned)	0.33M/2.06 × 10^8	69.03/61.41	68.05	73.39
$\ell_{1/2}$ (85% Pruned)	0.23M/1.42 × 10^8	78.42/73.43	5.05	72.52
$\ell_{1/4}$ (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	69.02	N/A
$\ell_{1/4}$ (75% Pruned)	0.31M/1.62 × 10^8	70.81/69.59	1.45	71.62
$\ell_{1/4}$ (85% Pruned)	0.19M/0.88 × 10^8	82.28/83.54	1.00	67.76
Tℓ_1(a = 10.0) (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	73.18	N/A
Tℓ_1(a = 10.0) (75% Pruned)	0.33M/2.12 × 10^8	68.84/60.18	66.62	73.78
Tℓ_1(a = 10.0) (85% Pruned)	0.23M/1.47 × 10^8	78.21/72.37	3.17	72.69
Tℓ_1(a = 1.0) (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	72.63	N/A
Tℓ_1(a = 1.0) (75% Pruned)	0.33M/2.12 × 10^8	69.16/60.24	72.60	73.42
Tℓ_1(a = 1.0) (90% Pruned)	0.18M/1.07 × 10^8	83.48/80.01	1.24	69.98
Tℓ_1(a = 0.5) (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	72.57	N/A
Tℓ_1(a = 0.5) (75% Pruned)	0.33M/2.10 × 10^8	69.33 /60.56	72.59	73.23
Tℓ_1(a = 0.5) (90% Pruned)	0.17M/1.06 × 10^8	83.61/80.16	1.37	70.16
MCP(a = 15000) (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	73.64	N/A
MCP(a = 15000)(75% Pruned)	0.33M/2.10 × 10^8	68.80/60.61	58.12	73.73
MCP(a = 15000) (90% Pruned)	0.18M/1.08 × 10^8	83.35/79.73	1.27	69.94
MCP(a = 10000) (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	73.40	N/A
MCP(a = 10000) (75% Pruned)	0.33M/2.06 × 10^8	68.73/61.36	40.76	73.95
MCP(a = 10000) (90% Pruned)	0.18M/1.08 × 10^8	83.19/79.68	1.10	69.10
SCAD(a = 15000) (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	73.41	N/A
SCAD(a = 15000) (75% Pruned)	0.33M/2.09 × 10^8	68.79/60.83	54.71	73.97
SCAD(a = 15000) (90% Pruned)	0.18M/1.08 × 10^8	83.33/79.72	1.42	69.87
SCAD(a = 10000) (0% Pruned)	1.06M/5.33 × 10^8	0.00/0.00	73.37	N/A
SCAD(a = 10000) (75% Pruned)	0.33M/2.04 × 10^8	68.80/61.66	47.70	73.75
SCAD(a = 10000) (90% Pruned)	0.18M/1.09 × 10^8	83.36/79.61	1.08	69.73

(b) CIFAR 100

Table 3.7: Counts of scaling factors that are averaged across five runs per model and regularizer.

	VGG-19		DenseNet-40		ResNet-164	
	$\lvert\gamma\rvert \leq 10^{-6}$	$\lvert\gamma\rvert > 10^{-6}$	$\lvert\gamma\rvert \leq 10^{-6}$	$\lvert\gamma\rvert > 10^{-6}$	$\lvert\gamma\rvert \leq 10^{-6}$	$\lvert\gamma\rvert > 10^{-6}$
ℓ_1	3483	2021	7309.2	2050.8	7321.4	4790.6
$\ell_{3/4}$	3244.8	2259.2	6261.4	3098.6	7944.2	4167.8
$\ell_{1/2}$	263.4	5240.6	490.6	8869.4	898	11214
$\ell_{1/4}$	3	5501	4	9356	11.6	12100.4
$\mathrm{T}\ell_1(a = 10.0)$	3559.6	1944.4	7372.8	1987.2	7466.6	4645.4
$\mathrm{T}\ell_1(a = 1.0)$	4021.2	1482.8	7731.4	1628.6	8757.2	3354.8
$\mathrm{T}\ell_1(a = 0.5)$	4216	1288	7839	1521	9192	2920
MCP ($a = 15000$)	3472.4	2031.6	7180.6	2179.4	6805.8	5306.2
MCP ($a = 10000$)	3485	2019	7123.6	2236.4	6438.4	5673.6
MCP ($a = 5000$)	3440.4	2063.6	6880.2	2479.8	5542.6	6569.4
SCAD ($a = 15000$)	3492.6	2011.4	7204.4	2155.6	6818.4	5293.6
SCAD ($a = 10000$)	3460.2	2043.8	7121.4	2238.6	6484.6	5627.4
SCAD ($a = 5000$)	3518.2	1985.8	6947.8	2412.2	5514.6	6597.4

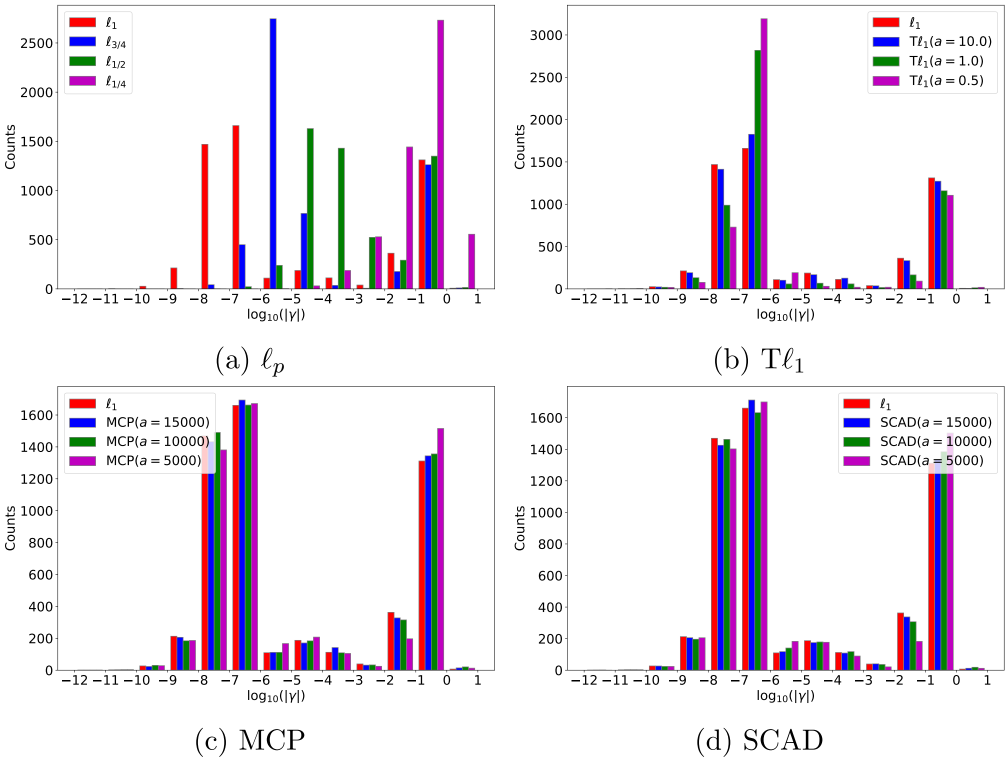
(a) CIFAR 10

	VGG-19		DenseNet-40		ResNet-164	
	$\lvert\gamma\rvert \leq 10^{-6}$	$\lvert\gamma\rvert > 10^{-6}$	$\lvert\gamma\rvert \leq 10^{-6}$	$\lvert\gamma\rvert > 10^{-6}$	$\lvert\gamma\rvert \leq 10^{-6}$	$\lvert\gamma\rvert > 10^{-6}$
ℓ_1	1417.2	4086.8	6382	2978	5030.4	7081.6
$\ell_{3/4}$	1895.8	3608.2	2208.6	7151.4	5584.6	6527.4
$\ell_{1/2}$	151.6	5352.4	94.4	9265.6	430	11682
$\ell_{1/4}$	1.6	5502.4	6	9354	6.6	12105.4
$\mathrm{T}\ell_1(a = 10.0)$	1629.6	3874.4	6555.4	2804.6	5192.8	6919.2
$\mathrm{T}\ell_1(a = 1.0)$	2555.6	2948.4	6919.8	2440.2	6250	5862
$\mathrm{T}\ell_1(a = 0.5)$	2802	2702	6889.6	2470.4	6739	5373
MCP ($a = 15000$)	1495	4009	6192.2	3167.8	4521.8	7590.2
MCP ($a = 10000$)	1440.4	4063.6	6055.6	3304.4	4191.8	7920.2
MCP ($a = 5000$)	1378	4126	5627.4	3732.6	3541.8	8570.2
SCAD ($a = 15000$)	1514.4	3989.6	6190.4	3169.6	4481.6	7630.4
SCAD ($a = 10000$)	1481.6	4022.4	6034.6	3325.4	4211.6	7900.4
SCAD ($a = 5000$)	1262	4242	5595.6	3764.4	3484.6	8627.4

(b) CIFAR 100

	CIFAR 10		CIFAR 100		SVHN	
	$\lvert\gamma\rvert \leq 10^{-6}$	$\lvert\gamma\rvert > 10^{-6}$	$\lvert\gamma\rvert \leq 10^{-6}$	$\lvert\gamma\rvert > 10^{-6}$	$\lvert\gamma\rvert \leq 10^{-6}$	$\lvert\gamma\rvert > 10^{-6}$
ℓ_1	4447.6	1056.4	8447.4	912.6	10058.8	2053.2
$\ell_{3/4}$	3862.2	1641.8	7079	2281	10130.4	1981.6
$\ell_{1/2}$	292	5212	543.4	8816.6	1070.4	11041.6
$\ell_{1/4}$	3.4	5500.6	7.2	9352.8	12.6	12099.4
$\mathrm{T}\ell_1(a = 10.0)$	4505.4	998.6	8497.4	862.6	10184.8	1927.2
$\mathrm{T}\ell_1(a = 1.0)$	4796.8	707.2	8674	686	10813.2	1298.8
$\mathrm{T}\ell_1(a = 0.5)$	4874	630	8746.4	613.6	11002.4	1109.6
MCP ($a = 15000$)	4365.6	1138.4	8419.8	940.2	9930.4	2181.6
MCP ($a = 10000$)	4356.6	1147.4	8390.4	969.6	9841	2271
MCP ($a = 5000$)	4242.6	1261.4	8330	1030	9333.6	2778.4
SCAD ($a = 15000$)	4378.2	1125.8	8405.6	954.4	9894.8	2217.2
SCAD ($a = 10000$)	4361.4	1142.6	8407	953	9858.8	2253.2
SCAD ($a = 5000$)	4244.4	1259.6	8330.2	1029.8	9353.8	2758.2

(c) SVHN



(a) ℓ_p

(b) Tℓ_1

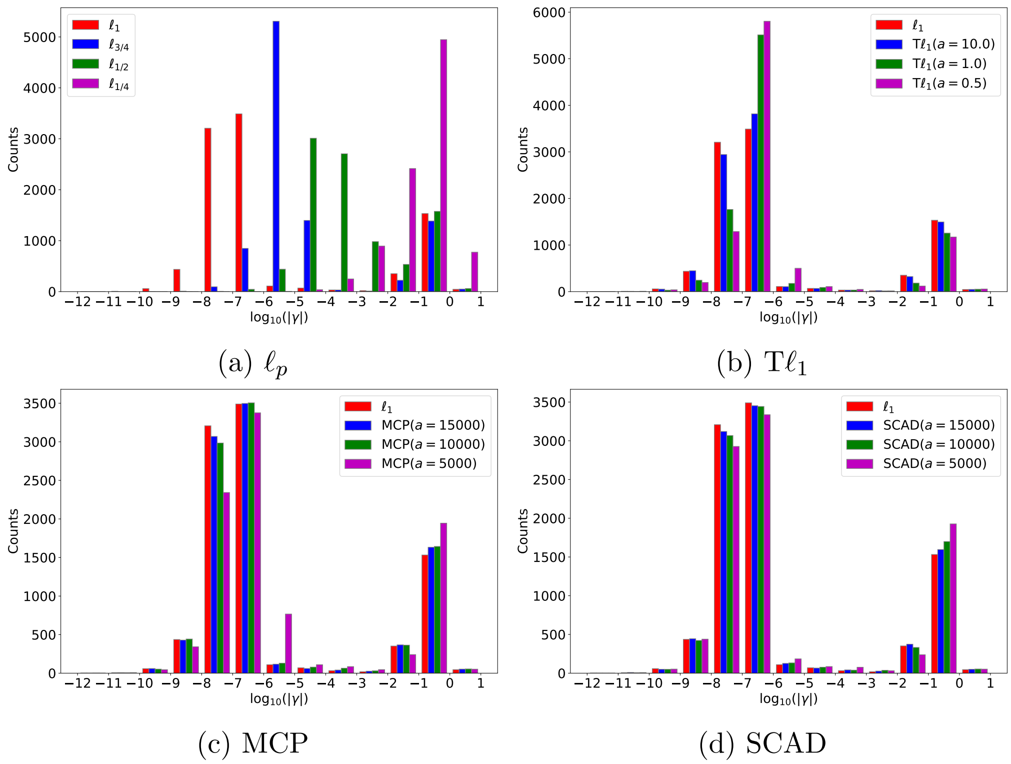
(c) MCP

(d) SCAD

Figure 3.6: Histogram of scaling factors γ in VGG-19 trained on CIFAR 10. The x-axis is $\log_{10}(|\gamma|)$.

has most of its scaling factors within the interval $(10^{-6}, 10^{-5})$. As p decreases, the values of the scaling factors tend farther away from 0. In fact, majority of the scaling factors for $\ell_{1/2}$ and $\ell_{1/4}$ are at least 10^{-6} in magnitude. Specifically for $\ell_{1/4}$, most of the scaling factors have absolute values at least 0.10. Hence, we can see why $\ell_{1/4}$ is sensitive to channel pruning. Lastly, for Tℓ_1, more scaling factors decrease towards 0 in magnitude as a decreases. Moreover, we observe that most of the scaling factors are accumulated within the interval $(10^{-7}, 10^{-6})$. Because Tℓ_1 causes more scaling factors to decrease towards 0 in magnitude, this might explain why Tℓ_1 is robust against channel pruning.

For CIFAR 100, Figures 3.9-3.11 show the histograms of the scaling factors while Table 3.7b records the average counts by magnitudes. Because CIFAR 100 is a more difficult classification dataset compared to CIFAR 10, most of the scaling factors appear to be within the interval $(10^{-1}, 1)$. However, DenseNet-40 shows bimodal distributions for Tℓ_1, MCP, and SCAD. Table 3.7b shows that more than half of the scaling factors are less than 10^{-6} in



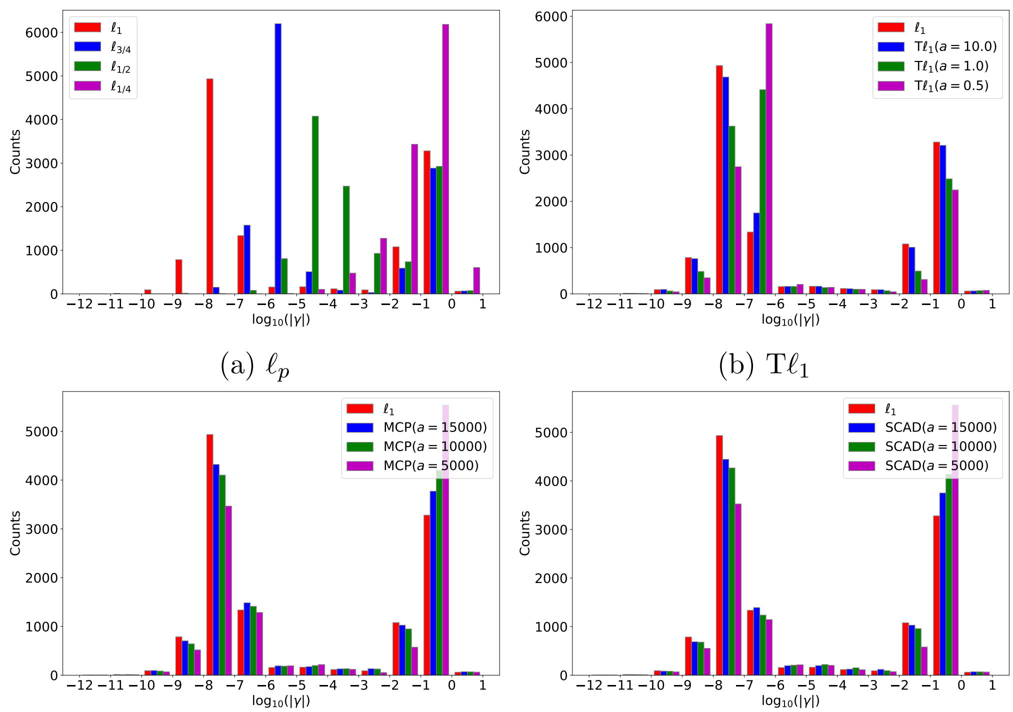
(a) ℓ_p (b) $T\ell_1$

(c) MCP (d) SCAD

Figure 3.7: Histogram of scaling factors γ in DenseNet-40 trained on CIFAR 10. The x-axis is $\log_{10}(|\gamma|)$.

magnitudes in DenseNet-40 for most regularizers, but they are more than 10^{-6} in magnitudes in VGG-19 and ResNet-164 for all regularizers. The distributions of the scaling factors convey why DenseNet-40 can be pruned at higher channel pruning ratios than VGG-19 and ResNet-164, as indicated by the middle rows of Figures 3.3-3.5. Across the three networks, MCP and SCAD have similar distributions with ℓ_1. For $\ell_{3/4}$, a considerable amount of scaling factors are within the interval $(10^{-6}, 10^{-5})$, but as p decreases, the magnitudes of most scaling factors increase. Hence, less than a few hundred scaling factors are below 10^{-6} in magnitudes. As a result, models regularized with $\ell_{1/2}$ and $\ell_{1/4}$ become more sensitive to channel pruning as demonstrated earlier. For $T\ell_1(a = 10.0)$, its distribution of scaling factors is similar to ℓ_1. However, when $a = 0.5, 1.0$, more scaling factors have magnitudes less than 10^{-5}, which demonstrates $T\ell_1$'s robustness to channel pruning when a is small enough.

Figures 3.12-3.14 and Table 3.7c provide statistics about SVHN. For all three networks, $T\ell_1(a = 10.0)$, SCAD, and MCP have similar distributions as ℓ_1. Similar to CIFAR 10 and



(a) ℓ_p (b) $\mathrm{T}\ell_1$

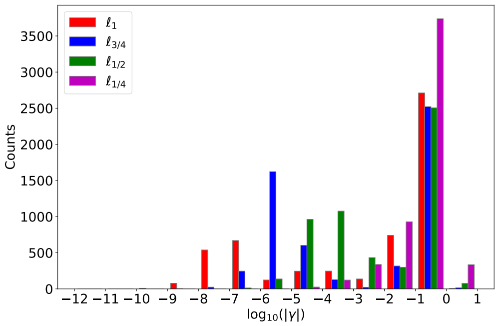
(c) MCP (d) SCAD

Figure 3.8: Histogram of scaling factors γ in ResNet-164 trained on CIFAR 10. The x-axis is $\log_{10}(|\gamma|)$.

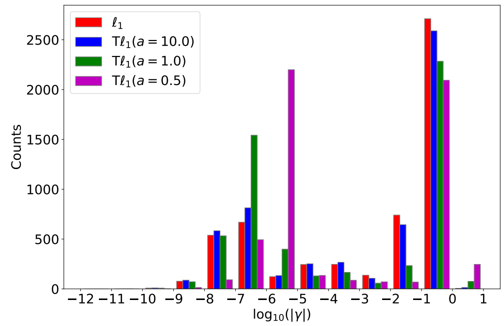
100, $\ell_{3/4}$ has most of its scaling factors to be in the interval $(10^{-6}, 10^{-5})$, but as p decreases for ℓ_p, the magnitudes of the scaling factors increase, resulting in at least 90% of the scaling factors to be at least 10^{-6} in magnitudes as shown in Table 3.7c. For $\mathrm{T}\ell_1(a = 0.5, 1.0)$ on the other hand, most of the scaling factors are in the interval $(10^{-7}, 10^{-6})$.

3.3.6 Comparison with Variational CNN Pruning

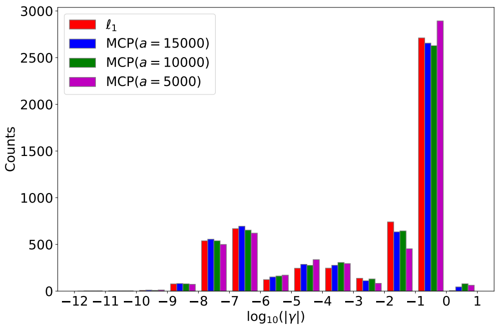
We have shown that network slimming with nonconvex regularizers can outperform the original with ℓ_1 regularization. Now we compare our proposed method with variational CNN pruning (VCP) proposed in [252], a Bayesian version of network slimming. VCP is designed to be robust against channel pruning, so we compare it with $\mathrm{T}\ell_1(a = 0.5, 1.0)$, which is proven to also be robust against channel pruning in our earlier analyses. The comparisons between the two methods are shown in Table 3.8, using results from DenseNet-40 and ResNet-164



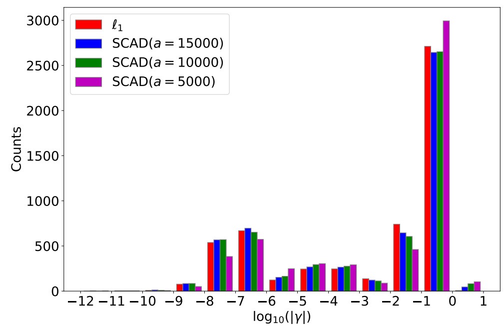
(a) ℓ_p



(b) $\mathrm{T}\ell_1$



(c) MCP



(d) SCAD

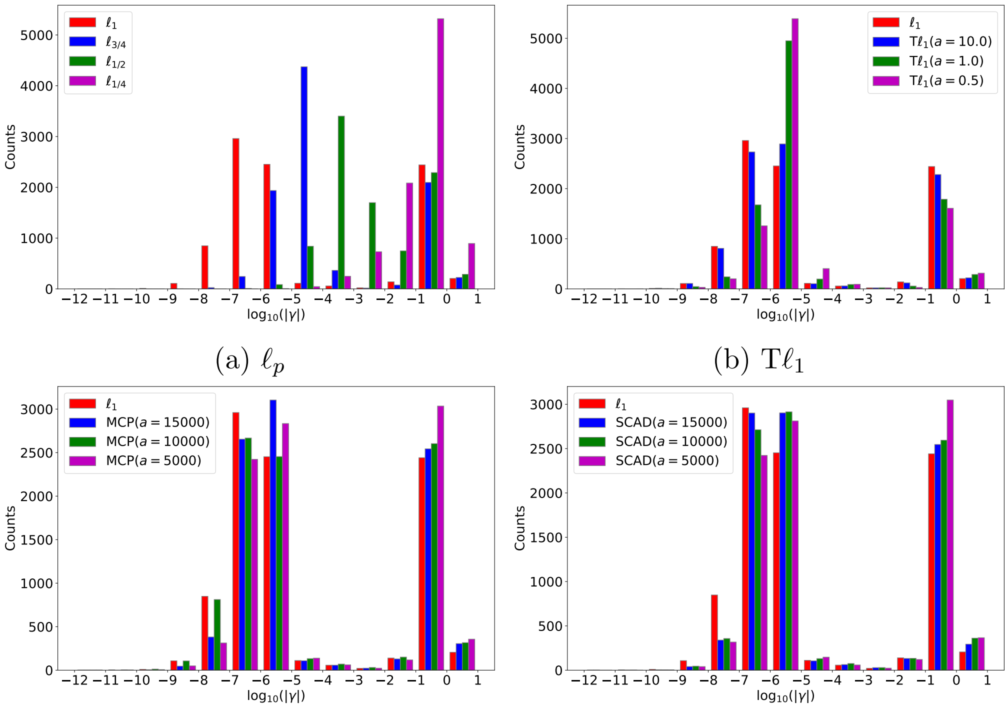
Figure 3.9: Histogram of scaling factors γ in VGG-19 trained on CIFAR 100. The x-axis is $\log_{10}(|\gamma|)$.

Table 3.8: Comparisons between network slimming with $\mathrm{T}\ell_1(a = 0.5, 1.0)$ and variational channel pruning. The results are immediately obtained after channel pruning.

Model	Dataset	Method	Test Accuracy	Percentage of Channels Pruned	Percentage of Parameters Pruned
DenseNet-40	CIFAR 10	VCP [252]	93.16%	60%	59.67%
		$\mathrm{T}\ell_1(a = 1.0)$	93.17%	60%	55.73%
		$\mathrm{T}\ell_1(a = 1.0)$	93.17%	80%	74.46%
		$\mathrm{T}\ell_1(a = 0.5)$	92.78%	60%	56.16%
		$\mathrm{T}\ell_1(a = 0.5)$	92.78%	80%	74.88%
	CIFAR 100	VCP [252]	72.19%	37%	37.73%
		$\mathrm{T}\ell_1(a = 1.0)$	72.63%	40%	36.91%
		$\mathrm{T}\ell_1(a = 1.0)$	72.63%	60%	55.35%
		$\mathrm{T}\ell_1(a = 0.5)$	72.57%	40%	36.98%
		$\mathrm{T}\ell_1(a = 0.5)$	72.58%	60%	55.46%
ResNet-164	CIFAR 10	VCP [252]	93.16%	74%	56.70%
		$\mathrm{T}\ell_1(a = 0.5)$	93.41%	75%	70.39%
	CIFAR 100	VCP [252]	73.76%	47%	17.59%
		$\mathrm{T}\ell_1(a = 1.0)$	74.89%	45%	25.56%
		$\mathrm{T}\ell_1(a = 0.5)$	74.72%	45%	27.74%

trained on CIFAR 10/100.

For DenseNet-40 trained on CIFAR 10, $\mathrm{T}\ell_1$ has a minimally better accuracy with less parameters pruned than VCP with 60% channels pruned. However, we can increase the percentage of channels pruned to 80% for $\mathrm{T}\ell_1$ so that the number of parameters are reduced while maintaining the same accuracy. On CIFAR 100, with similar percentages of channels pruned, $\mathrm{T}\ell_1$



(a) ℓ_p

(b) $\mathrm{T}\ell_1$

(c) MCP

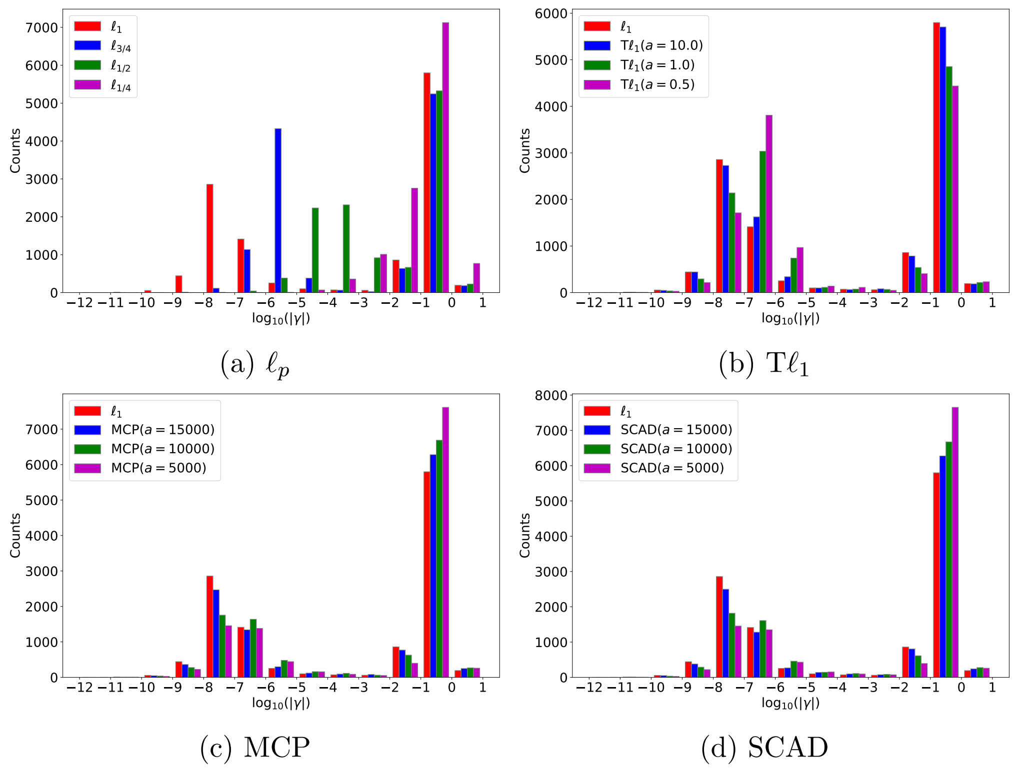
(d) SCAD

Figure 3.10: Histogram of scaling factors γ in DenseNet-40 trained on CIFAR 100. The x-axis is $\log_{10}(|\gamma|)$.

has a much better accuracy than VCP but again with less parameters pruned. Nevertheless, we can increase the percentage of channels pruned to 60% and the accuracy will remain the same with more parameters pruned.

On ResNet-164, with similar percentages of channels pruned, $\mathrm{T}\ell_1$ outperforms VCP by a large margin for both test accuracy and percentage of parameters pruned. For CIFAR 10, only $\mathrm{T}\ell_1(a = 0.5)$ is able to have 75% of the channels pruned, and it saves more parameters by almost 24% with test accuracy better by 0.25%. For CIFAR 100%, with 2% less channels pruned, $\mathrm{T}\ell_1$ prunes at least 7.97% more parameters than VCP while having better accuracy of at least 0.96%.

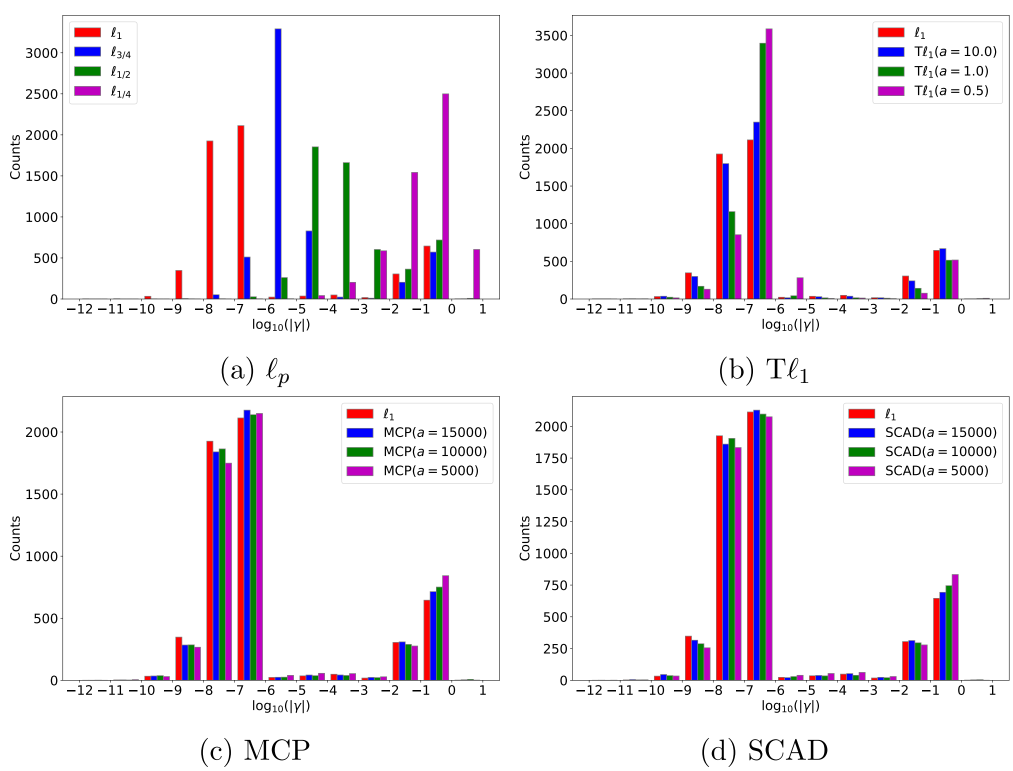
Overall, network slimming with $\mathrm{T}\ell_1$ is competitive against the latest variant of network slimming.



(a) ℓ_p (b) Tℓ_1

(c) MCP (d) SCAD

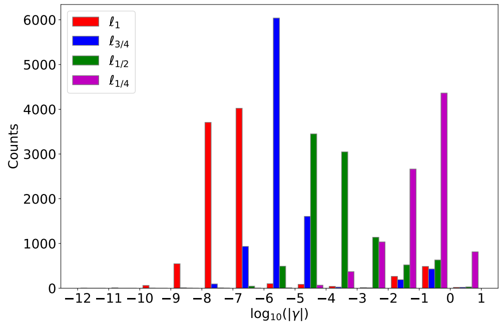
Figure 3.11: Histogram of scaling factors γ in ResNet-164 trained on CIFAR 100. The x-axis is $\log_{10}(|\gamma|)$.



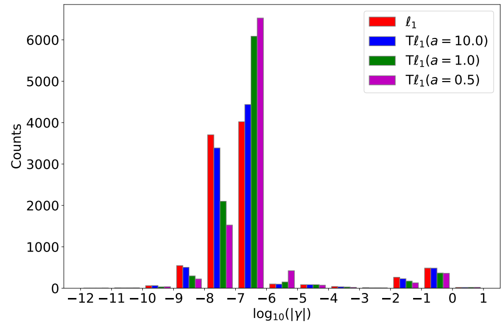
(a) ℓ_p (b) Tℓ_1

(c) MCP (d) SCAD

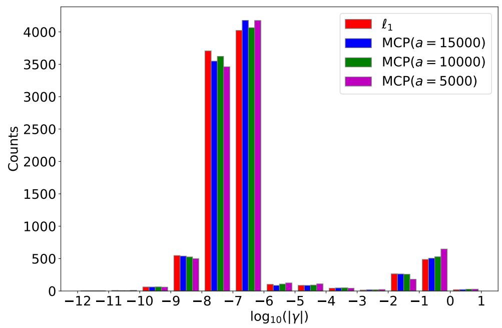
Figure 3.12: Histogram of scaling factors γ in VGG-19 trained on SVHN. The x-axis is $\log_{10}(|\gamma|)$.



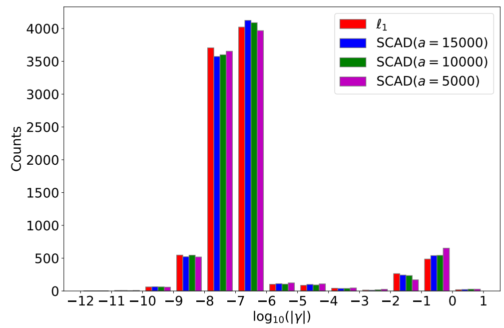
(a) ℓ_p



(b) $\mathrm{T}\ell_1$

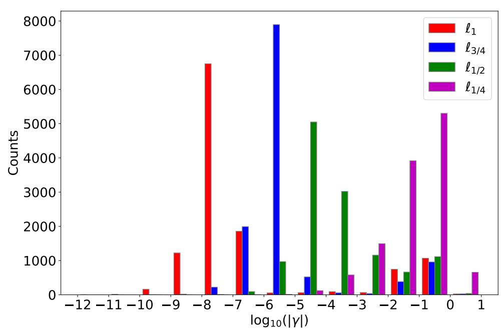


(c) MCP

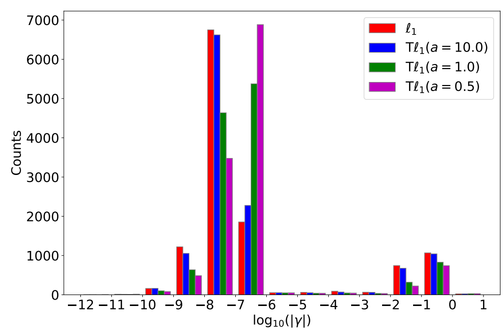


(d) SCAD

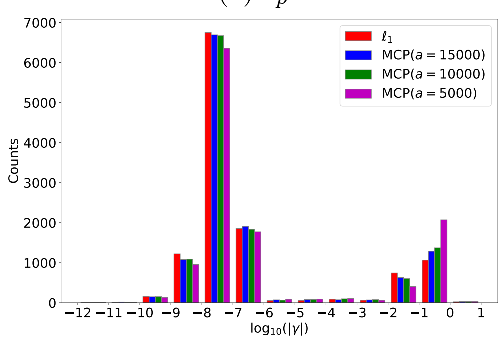
Figure 3.13: Histogram of scaling factors γ in DenseNet-40 trained on SVHN. The x-axis is $\log_{10}(|\gamma|)$.



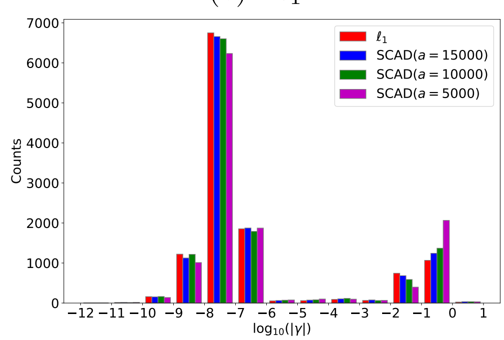
(a) ℓ_p



(b) $\mathrm{T}\ell_1$



(c) MCP



(d) SCAD

Figure 3.14: Histogram of scaling factors γ in ResNet-164 trained on SVHN. The x-axis is $\log_{10}(|\gamma|)$.

Chapter 4

A Proximal Algorithm for Network Slimming

The original optimization algorithm used for network slimming (NS) is subgradient descent [190], but it has theoretical and practical issues. Subgradient descent does not necessarily decrease the loss function value after each iteration, even when performed exactly with full batch of data [18]. Moreover, unless with some additional modifications, such as backtracking line search, subgradient descent may not converge to a critical point [169]. When implemented in practice, barely any of the scaling factors have values exactly at zeroes by the end of training. Consequently, a threshold value needs to be determined in order to remove channels whose scaling factors are below it and the resulting compressed model needs to be retrained to recover its original accuracy.

In this chapter, we design an alternative optimization algorithm for NS. The optimization algorithm is based on proximal alternating linearized minimization (PALM) [20], which is friendly to implement for back-propagation training of CNNs. The algorithm has more theoretical and practical advantages than subgradient descent. Under certain conditions, the proposed algorithm does converge to a critical point. When used in practice, the proposed algorithm enforces the scaling factors of insignificant channels to be exactly at zero by the end of training. Hence, there is no need to set a scaling-factor threshold to identify which channels to remove. Because the model is trained towards an actual sparse structure by the proposed algorithm, the model accuracy is preserved after the insignificant channels are pruned, so fine tuning is unnecessary, unlike for other pruning methods [84, 92, 134, 214]. The only trade-off of the proposed algorithm is a slight decrease in accuracy compared to the original baseline model.

4.1 Proposed Algorithm

In this section, we develop a novel PALM algorithm [20] for NS that consists of two straightforward, general steps per epoch: stochastic gradient descent on the weight parameters, including the scaling factors of the batch normalization layers, and soft-thresholding on the scaling factors. Because this algorithm automatically trains a CNN towards a sparse structure, many of the scaling factors are already zero at the end of training, so their associated channels are safe to remove without damaging the model accuracy. As a result, setting a scaling-factor threshold for channel pruning and fine tuning a compressed CNN are no longer needed, especially for practitioners whose time and resources are limited.

4.1.1 Batch Normalization Layer

Let $z \in \mathbb{R}^{B \times C \times H \times W}$ denote an output feature map, where B is the mini-batch size, C is the number of channels, and H and W are the height and width of the feature map, respectively. Recall that for each channel $i = 1, \dots, C$, the output of a batch normalization (BN) layer on each channel z_i is given by

$$z_i' = \gamma_i \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta_i, \tag{4.1}$$

where μ_B and σ_B are the mean and standard deviation of the inputs across the mini-batch B, ϵ is a small constant for numerical stability, and γ_i and β_i are trainable weight parameters that help restore the representative power of the input z_i. The weight parameter γ_i is defined to be the scaling factor of channel i. The scaling factor γ_i determines how important channel i is to the computation of the CNN as it is multiplied to all pixels of the same channel i within the feature map z.

4.1.2 Numerical Optimization

Let $\{(x_i, y_i)\}_{i=1}^N$ be a given dataset, where x_i is the training input and y_i is its corresponding label or value. Using the dataset $\{(x_i, y_i)\}_{i=1}^N$, we train a CNN with c total channels, where each of its convolutional layers is followed by a BN layer. Let $\gamma \in \mathbb{R}^c$ be the vector of trainable scaling factors of the CNN, where each entry $\gamma_i, i = 1, \ldots, c$, is a scaling factor of channel i. Moreover, let $W \in \mathbb{R}^n$ be a vector of all n trainable weight parameters, excluding the scaling factors, in the CNN. The objective function of the CNN that NS [134] minimizes is

$$\min_{W,\gamma} \frac{1}{N} \sum_{i=1}^N \mathcal{L}(h(x_i, W, \gamma), y_i) + \lambda \|\gamma\|_1, \tag{4.2}$$

where $h(x_i, W, \gamma)$ is the output of the CNN predicted on data point x_i; $\mathcal{L}(h(x_i, W, \gamma), y_i)$ is the loss function between the prediction $h(x_i, W, \gamma)$ and ground truth y_i, such as the cross-entropy loss function; and $\lambda > 0$ is the regularization parameter for the ℓ_1 penalty on γ. In [134], (4.2) is solved by the following gradient descent scheme with step size δ^t for each epoch t:

$$W^{t+1} = W^t - \delta^t \nabla_W \tilde{\mathcal{L}}(W^t, \gamma^t) \tag{4.3a}$$

$$\gamma^{t+1} = \gamma^t - \delta^t \left(\nabla_\gamma \tilde{\mathcal{L}}(W^t, \gamma^t) + \lambda \partial \|\gamma^t\|_1 \right), \tag{4.3b}$$

where $\tilde{\mathcal{L}}(W, \gamma) \coloneqq \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(h(x_i, W, \gamma), y_i)$ and

$$\partial \|\gamma\|_1 = \left\{ \zeta \in \mathbb{R}^c : \zeta_i = \begin{cases} \text{sign}(\gamma_i) & \text{if } \gamma_i \neq 0 \\ \zeta_i \in [-1, 1] & \text{if } \gamma_i = 0 \end{cases} \right\}$$

is the subgradient of the ℓ_1 norm.

By (4.3), we observe that γ is optimized by subgradient descent. Unfortunately, using subgradient descent can lead to practical issues. When $\gamma_i = 0$ for some channel i, the subgradient needs to be chosen precisely. Not all subgradient vectors at a non-differentiable point decrease the value of (4.2) in each epoch [18], so we need to find one that does among the infinite number of choices. In the numerical implementation of NS [1], the subgradient ζ^t is selected such that $\zeta_i^t = 0$ by default when $\gamma_i^t = 0$, but such selection is not verified to decrease the value of (4.2) in each epoch t. Lastly, subgradient descent only pushes the scaling factors of irrelevant channels to be near zero in value but not at zero. For this reason, when pruning a CNN, the user needs to determine the appropriate scaling-factor threshold to remove its channels where no layers have zero channels and then fine tune it to restore its original accuracy. However, if too many channels are pruned that the fine-tuned accuracy is significantly less than the original, the user may need to decrease the threshold and fine tune again, thereby wasting more computational time and resources.

To develop an alternative algorithm that does not possess the practical issues of subgradient descent, we reformulate (4.2) as a constrained optimization problem by introducing an

[1]`https://github.com/Eric-mingjie/network-slimming`

auxiliary variable ξ, giving us

$$\begin{aligned} \min_{W,\gamma,\xi} \quad & \tilde{\mathcal{L}}(W,\gamma) + \lambda \|\xi\|_1 \\ \text{s.t.} \quad & \xi = \gamma. \end{aligned} \tag{4.4}$$

However, we relax the constraint by a quadratic penalty with parameter $\beta > 0$, leading to a new unconstrained optimization problem:

$$\min_{W,\gamma,\xi} \quad \tilde{\mathcal{L}}(W,\gamma) + \lambda \|\xi\|_1 + \frac{\beta}{2}\|\gamma - \xi\|_2^2. \tag{4.5}$$

In (4.2), γ is optimized for both model accuracy and sparsity, which can be difficult to balance when training a CNN. However, in (4.5), γ is optimized for only model accuracy because it is a variable of the overall loss function $\tilde{\mathcal{L}}(W,\gamma)$ while ξ is optimized only for sparsity because it is penalized by the ℓ_1 norm. The quadratic penalty enforces γ and ξ to be similar in values, thereby ensuring γ to be sparse.

Let (W,γ) be a concatenated vector of W and γ. We minimize (4.5) via alternating minimization, so for each epoch t, we solve the following subproblems:

$$(W^{t+1},\gamma^{t+1}) \in \arg\min_{W,\gamma} \tilde{\mathcal{L}}(W,\gamma) + \frac{\beta}{2}\|\gamma - \xi^t\|_2^2 \tag{4.6a}$$

$$\xi^{t+1} \in \arg\min_{\xi} \lambda\|\xi\|_1 + \frac{\beta}{2}\|\gamma^{t+1} - \xi\|_2^2. \tag{4.6b}$$

Below, we describe how to solve each subproblem in details.

(W,γ)-subproblem

The (W,γ)-subproblem given in (4.6a) cannot be solved in closed form because the loss function $\tilde{\mathcal{L}}(W,\gamma)$ is a composition of several nonlinear functions. Typically, when training a

CNN, this subproblem would be solved by (stochastic) gradient descent. To formulate (4.6a) as a gradient descent step, we follow a prox-linear strategy as follows:

$$\begin{aligned}
&(W^{t+1}, \gamma^{t+1}) \in \underset{W,\gamma}{\arg\min}\, \tilde{\mathcal{L}}(W^t, \gamma^t) \\
&+ \langle \nabla \tilde{\mathcal{L}}(W^t, \gamma^t), (W, \gamma) - (W^t, \gamma^t) \rangle \\
&+ \frac{\alpha}{2} \|(W, \gamma) - (W^t, \gamma^t)\|_2^2 + \frac{\beta}{2} \|\gamma - \xi^t\|_2^2 \\
&= \underset{W,\gamma}{\arg\min}\, \tilde{\mathcal{L}}(W^t, \gamma^t) + \langle \nabla_W \tilde{\mathcal{L}}(W^t, \gamma^t), W - W^t \rangle \\
&+ \langle \nabla_\gamma \tilde{\mathcal{L}}(W^t, \gamma^t), \gamma - \gamma^t \rangle \\
&+ \frac{\alpha}{2} \|W - W^t\|_2^2 + \frac{\alpha}{2} \|\gamma - \gamma^t\|_2^2 + \frac{\beta}{2} \|\gamma - \xi^t\|_2^2
\end{aligned} \tag{4.7}$$

where $\alpha > 0$. By differentiating with respect to each variable, setting the partial derivative equal to zero, and solving for the variable, we have

$$W^{t+1} = W^t - \frac{1}{\alpha} \nabla_W \tilde{\mathcal{L}}(W^t, \gamma^t) \tag{4.8a}$$

$$\gamma^{t+1} = \frac{\alpha\gamma^t + \beta\xi^t}{\alpha + \beta} - \frac{1}{\alpha + \beta} \nabla_\gamma \tilde{\mathcal{L}}(W^t, \gamma^t). \tag{4.8b}$$

We see that (4.8a) is gradient descent on W^t with step size $\frac{1}{\alpha}$ while (4.8b) is gradient descent on a weighted average of γ^t and ξ^t with step size $\frac{1}{\alpha+\beta}$. These steps are straightforward to implement in practice when training a CNN because the gradient $(\nabla_W \tilde{\mathcal{L}}(W^t, \gamma^t), \nabla_\gamma \tilde{\mathcal{L}}(W^t, \gamma^t))$ can be approximated by its stochastic gradient estimators obtained from backpropagation.

ξ-subproblem

To solve (4.6b), we perform a proximal update by minimizing the following subproblem:

$$\xi^{t+1} \in \underset{\xi}{\arg\min}\, \lambda \|\xi\|_1 + \frac{\alpha}{2} \|\xi - \xi^t\|_2^2 + \frac{\beta}{2} \|\gamma^{t+1} - \xi\|_2^2. \tag{4.9}$$

Algorithm 3: Proximal network slimming: proximal algorithm for minimizing (4.5)

Input: Regularization parameter λ, proximal parameter α, penalty parameter β

Initialize W^1 with random values.

Initialize γ^1 such that $\gamma_i = 0.5$ for each channel i.

for each epoch $t = 1, \dots, T$ **do**

  $W^{t+1} = W^t - \frac{1}{\alpha}\nabla_W \tilde{\mathcal{L}}(W^t, \gamma^t)$ by stochastic gradient descent or variant.

  $\gamma^{t+1} = \frac{\alpha\gamma^t + \beta\xi^t}{\alpha+\beta} - \frac{1}{\alpha+\beta}\nabla_\gamma \tilde{\mathcal{L}}(W^t, \gamma^t)$ by stochastic gradient descent or variant.

  $\xi^{t+1} = \mathcal{S}\left(\frac{\alpha\xi^t + \beta\gamma^{t+1}}{\alpha+\beta}, \frac{\lambda}{\beta+\alpha}\right)$.

end for

Expanding it gives

$$\begin{aligned}
\xi^{t+1} &= \arg\min_{\xi} \|\xi\|_1 + \frac{\beta}{2\lambda}\|\xi - \gamma^{t+1}\|_2^2 + \frac{\alpha}{2\lambda}\|\xi - \xi^t\|_2^2 \\
&= \arg\min_{\xi} \|\xi\|_1 + \frac{1}{2\left(\frac{\lambda}{\beta+\alpha}\right)}\left\|\xi - \frac{\alpha\xi^t + \beta\gamma^{t+1}}{\alpha+\beta}\right\|_2^2 \\
&= \mathcal{S}\left(\frac{\alpha\xi^t + \beta\gamma^{t+1}}{\alpha+\beta}, \frac{\lambda}{\beta+\alpha}\right),
\end{aligned}$$

where $\mathcal{S}(x, \lambda)$ is the soft-thresholding operator defined by

$$(\mathcal{S}(x,\lambda))_i = \operatorname{sign}(x_i)\max\{0, |x_i| - \lambda\}$$

for each entry i. Therefore, ξ is updated by performing soft thresholding on the weighted average between ξ^t and γ^{t+1}. We summarize the proximal algorithm for NS in Algorithm 3. Throughout the rest of the paper, we will refer to Algorithm 3 as proximal NS.

4.2 Convergence Analysis

To establish global convergence of proximal NS, we present relevant definitions and assumptions.

Definition 4.1 ([10, 20])**.** *A proper, lower-semicontinuous function* $f : \mathbb{R}^m \to (-\infty, \infty]$ *satisfies the Kurdyka-Łojasiewicz (KL) property at a point* $\bar{x} \in dom(\partial f) := \{x \in \mathbb{R}^m : \partial f(x) \neq \varnothing\}$ *if there exist* $\eta \in (0, +\infty]$*, a neighborhood* U *of* $\bar{x}$*, and a continuous concave function* $\phi : [0, \eta) \to [0, \infty)$ *with the following properties:*

i) $\phi(0) = 0$,

ii) ϕ *is continuously differentiable on* $(0, \eta)$,

iii) $\phi'(x) > 0$ *for all* $x \in (0, \eta)$,

iv) for any $x \in U$ *with* $f(\bar{x}) < f(x) < f(\bar{x}) + \eta$*, it holds that*

$$\phi'(f(x) - f(\bar{x}))dist(0, \partial f(x)) \geq 1. \tag{4.10}$$

If f *satisfies the KL property at every point* $x \in dom(\partial f)$*, then* f *is called a KL function.*

Assumption 4.1. *Suppose that*

a) $\tilde{\mathcal{L}}(W, \gamma)$ *is a proper, differentiable, and nonnegative function.*

b) $\nabla\tilde{\mathcal{L}}(W, \gamma)$ *is Lipschitz continuous with constant* L.

c) $\tilde{\mathcal{L}}(W, \gamma)$ *is a KL function.*

Remark 4.1. *Assumption 4.1 (a)-(b) are common in the analysis of nonconvex algorithm (e.g., [10, 20]). For Assumption 4.1, most commonly used loss functions for CNNs are verified to be KL functions [242]. Some neural network architectures do not satisfy Assumption 4.1(a) when they contain nonsmooth functions and operations, such as the* $ReLU$ *activation functions and max poolings. However, these functions and operations can be replaced with their smooth approximations. For example, the smooth approximation of* $ReLU$ *is the softplus function* $\frac{1}{c}\log(1+\exp(cx))$ *for some parameter* $c > 0$ *while the smooth approximation the max*

Table 4.1: The average number of scaling factors equal to zero at the end of training. Channels are pruned when their corresponding scaling factors γ_i are exactly equal to 0. Each architecture is trained five times per dataset.

Architecture	Total Channels/γ_i	CIFAR 10 Avg. Number of $\gamma_i = 0$	CIFAR 100 Avg. Number of $\gamma_i = 0$
VGG-19	5504	4005.8	2974
DenseNet-40	9360	6545.8	5710
ResNet-164	12112	7087.4	5533.2

function for max pooling is the softmax function $\sum_{i=1}^{n} \frac{x_i e^{cx_i}}{\sum_{i=1}^{n} e^{cx_i}}$ *for some parameter* $c > 0$. *Besides, Fu et al.[72] made a similar assumption to establish convergence for their algorithm designed for weight and filter pruning. Regardless, our numerical experiments demonstrate that our proposed algorithm still converges for CNNs containing ReLU activation functions and max pooling.*

For brevity, we denote

$$F(W, \gamma, \xi) = \tilde{\mathcal{L}}(W, \gamma) + \lambda \|\xi\|_1 + \frac{\beta}{2} \|\gamma - \xi\|_2^2.$$

Now, we are ready to present the main theorem:

Theorem 4.1. *Under Assumption 4.1, if* $\{(W^t, \gamma^t, \xi^t)\}_{t=1}^{\infty}$ *generated by Algorithm 3 is bounded and we have* $\alpha > L$, *then* $\{(W^t, \gamma^t, \xi^t)\}_{t=1}^{\infty}$ *converges to a critical point* (W^*, γ^*, ξ^*) *of* F.

The proof is delayed to Section 4.4. The proof needs tools from variational analysis [18, 184] and requires satisfying three conditions [10, 20, 178]: (1) sufficient decrease in F, (2) relative error property of ∂F, and (3) subsequential convergence of the iterates (W^t, γ^t, ξ^t) to a critical point of F.

4.3 Numerical Experiments

We evaluate proximal NS on VGGNet [192], DenseNet [94, 93], and ResNet [89] trained on CIFAR 10/100 [110].

4.3.1 CIFAR 10/100 Datasets

The CIFAR 10/100 dataset [110] consists of 60,000 natural images of resolution 32×32 with 10/100 categories. The dataset is split into two sets: a training set of 50,000 images and a test set of 10,000 images. As done in recent works [89, 134], standard augmentation techniques, such as shifting and mirroring, and normalization, are applied to the images before they are used for training and testing.

4.3.2 Implementation Details

For CIFAR 10/100, the implementation is mostly the same as in [134]. More specifically, we train the networks from scratch for 160 epochs using stochastic gradient descent with initial learning rate at 0.1 that reduces by a factor of 10 at the 80th and 120th epochs. In addition, the models are trained with weight decay 10^{-4} and Nesterov momentum of 0.9 without damping. The training batch size is 64. However, the parameter λ is set differently. For our numerical experiments, we have $\lambda = 0.0025$ and $\beta = 100$ for VGG-19 and $\lambda = 0.001$ and $\beta = 1.0$ for DenseNet-40 and ResNet-164. We have initially $\alpha = 10$ in Algorithm 3, the reciprocal of the learning rate, and it changes accordingly to the learning rate schedule. A model is trained five times for each architecture and dataset to obtain the average statistics. The models are trained on NVIDIA GeForce RTX 2080.

Table 4.2: Results between the different NS methods on CIFAR 10. Note that we train the baseline architectures and original NS five times to obtain the average statistics, while the results for variational NS are originally reported from [252].

Architecture	Method	Avg. Training Time per Epoch (s) Pre-Pruned/Fine Tuned	% Channels Pruned	% Parameters Pruned	% FLOPS Pruned	Test Accuracy (%) Post Pruned/Fine Tuned
VGG-19	Baseline	38.10/—-	N/A	N/A	N/A	93.83/—-
	Original NS [134]	37.70/28.04	72.00	87.13	49.98	17.35/93.89
	Proximal NS (ours)	40.39/22.86	72.78	**91.45**	**60.86**	93.47/93.74
DenseNet-40	Baseline	117.45/—-	N/A	N/A	N/A	94.25/—-
	Original NS [134]	119.49/76.49	70.01	63.24	**56.72**	61.77/94.06
	Variational NS [252]	Not Reported	60.00	59.67	44.78	93.16/—-
	Proximal NS (ours)	118.34/77.70	69.93	**64.12**	54.70	93.65/93.87
ResNet-164	Baseline	146.41/—-	N/A	N/A	N/A	94.75/—-
	Original NS [134]	151.62/121.52	50.00*	22.53	30.46	88.96/95.09
	Variational NS [252]	Not Reported	74.00	**56.70**	**49.08**	93.16/—-
	Proximal NS (ours)	149.86/116.52	58.52	45.59	45.80	93.72/94.14

* This is the maximum possible for all five networks to remain functional for inference.

Table 4.3: Results between the different NS methods on CIFAR 100. Note that we train the baseline architectures and original NS five times to obtain the average statistics, while the results for variational NS are originally reported from [252].

Architecture	Method	Avg. Training Time per Epoch (s) Pre-Pruned/Fine Tuned	% Channels Pruned	% Parameters Pruned	% FLOPS Pruned	Test Accuracy (%) Post Pruned/Fine Tuned
VGG-19	Baseline	37.83	N/A	N/A	N/A	72.73/—-
	Original NS [134]	38.03/29.08	54.00	77.86	38.07	1.00/73.10
	Proximal NS (ours)	40.40/24.77	54.04	**80.14**	**49.17**	71.61/72.72
DenseNet-40	Baseline	117.17	N/A	N/A	N/A	74.55/—-
	Original NS [134]	119.32/80.31	61.00	55.61	**49.16**	34.64/74.62
	Variational NS [252]	Not Reported	37.00	37.73	22.67	72.19/—-
	Proximal NS (ours)	118.66/81.88	61.00	**55.87**	45.42	73.32/73.99
ResNet-164	Baseline	145.37	N/A	N/A	N/A	76.79/—-
	Original NS [134]	150.65/120.21	46.00	17.46	30.96	21.57/77.54
	Variational NS [252]	Not Reported	47.00	17.59	27.16	73.76/—-
	Proximal NS (ours)	149.15/119.21	45.68	**25.39**	**33.97**	75.67/76.93

4.3.3 Results

We apply proximal NS on VGG-19, DenseNet-40, and ResNet-164 to train them on CIFAR 10/100. According to Table 4.1, proximal NS drives a significant number of scaling factors to be exactly at zero for each architecture and dataset. In particular, for VGG-19 and DenseNet-40, at least 54% of the scaling factors are zeroes while for ResNet-164, at least 45% are zeroes. Hence, we can safely remove the channels with zero scaling factors because they are unnecessary for inference. Using proximal NS, we do not need to determine a threshold to figure out how many channels to remove and how much accuracy we need to sacrifice as a result of pruning, like for the original NS algorithm [134].

We compare proximal NS with the original NS [134] and variational NS [252], a Bayesian version of NS. To evaluate the drop in accuracy as a result of pruning, we include the baseline accuracy, where the architecture is trained without any regularization on the scaling factors.

The models trained with original NS and proximal NS are fine tuned with the same setting as the first time training but without ℓ_1 regularization on the scaling factors. The results are reported in Tables 4.2-4.3.

Without fine tuning, proximal NS outperforms both the original and variational NS in test accuracy while reducing a significant amount of parameters and FLOPs. Because proximal NS trains a model towards a sparse structure, the model accuracy is less than the baseline accuracy by at most 1.23% and it remains the same between before and after pruning, a property that the original NS does not have. Although variational NS is designed to preserve test accuracy after pruning, it does not compress as well as proximal NS for all architectures except for ResNet-164 trained on CIFAR 10.

After fine tuning the models trained by the original and proximal NS, their model accuracy improve. However, the fine-tuned models compressed by proximal NS has lower test accuracy than the models from original NS by at most 0.95%. Although it might be preferable to have a more accurate model from original NS, the improvement compared to a pruned model by proximal NS without fine tuning is at most 2% in test accuracy after a few more hours of training. For example, for ResNet-164 trained on CIFAR 10, proximal NS takes about 7 hours to attain an average accuracy of 93.72% while the original NS requires about 12 hours total to achieve 1.37% higher accuracy.

Overall, proximal NS yields a model that is generally more compressed and accurate than the other methods after the first round of training. Fine tuning is optional for proximal NS if a few more hours of training is permitted to improve the test accuracy by at most 2% improvement.

4.4 Proofs

Before we prove Theorem 4.1, we introduce important definitions and lemmas from variational analysis.

Definition 4.2 ([184])**.** *Let $f : \mathbb{R}^n \to (-\infty, +\infty]$ be a proper and lower semicontinuouous function.*

(a) The Fréchet subdifferential of f at the point $x \in dom\ f := \{x \in \mathbb{R}^n : f(x) < \infty\}$ is the set

$$\hat{\partial} f(x) = \left\{ v \in \mathbb{R}^{n^2} : \liminf_{y \neq x, y \to x} \frac{f(y) - f(x) - \langle v, y - x \rangle}{\|y - x\|} \geq 0 \right\}.$$

(b) The limiting subdifferential of f at the point $x \in dom\ f$ is the set

$$\partial f(x) = \left\{ v \in \mathbb{R}^{n^2} : \exists \{(x^t, y^t)\}_{t=1}^{\infty} \ s.t.\ x^t \to x, f(x^t) \to f(x), \hat{\partial} f(x^t) \ni y^t \to y \right\}.$$

We note that the limiting subdifferential is closed [184]:

$$(x^t, y^t) \to (x, y), f(x^t) \to f(x), y^t \in \partial f(x^t) \implies y \in \partial f(x).$$

A point x is a critical point of f if $0 \in \partial f(x)$.

Lemma 4.1 (Strong Convexity Lemma [18])**.** *A function $f(x)$ is called strongly convex with parameter μ if and only if one of the following conditions holds:*

a) $g(x) = f(x) - \frac{\mu}{2}\|x\|_2^2$ is convex.

b) $\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \mu \|x - y\|_2^2 \ \forall x, y.$

c) $f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|_2^2 \ \forall x, y.$

Lemma 4.2 (Descent Lemma [18]). *If $\nabla f(x)$ is Lipschitz continuous with parameter $L > 0$, then*

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|x - y\|_2^2.$$

To establish global convergence of Algorithm 3, we need to satisfy three conditions given in the following theorem:

Theorem 4.2 ([20, 178]). *Let $\Psi : \mathbb{R}^n \to (-\infty, +\infty]$ be a KL function that is proper, lower semicontinuous, and lower bounded. Suppose that $\{x^t\}_{t=1}^{\infty}$ is a bounded sequence generated by an algorithm $\mathcal{A}$ such that it satisfies the following conditions:*

(a) There exists a scalar $\rho_1 > 0$ such that

$$\rho_1 \|x^{t+1} - x^t\|_2^2 \leq \Psi(x^t) - \Psi(x^{t+1})$$

for all $t \in \mathbb{N}$.

(b) There exists a scalar $\rho_2 > 0$ such that for some $y^{t+1} \in \partial\Psi(x^{t+1})$, we have

$$\|y^{t+1}\|_2 \leq \rho_2 \|x^{t+1} - x^t\|_2$$

for all $t \in \mathbb{N}$.

(c) Limiting Continuity: Each limit point of $\{x^t\}_{t=1}^{\infty}$ is a critical point of Ψ.

Then the sequence $\{x^t\}_{t=1}^{\infty}$ converges to a critical point x^* of Ψ.

Before proving Theorem 4.1, we prove some necessary lemmas.

Lemma 4.3 (Sufficient Decrease). *Let $\{(W^t, \gamma^t, \xi^t)\}_{t=1}^{\infty}$ be a sequence generated by Algorithm 3. Under Assumption 4.1, we have*

$$F(W^{t+1}, \gamma^{t+1}, \xi^t) - F(W^t, \gamma^t, \xi^t) \leq \frac{L - 2\alpha}{2} \|(W^{t+1}, \gamma^{t+1}) - (W^t, \gamma^t)\|_2^2, \tag{4.11}$$

$$F(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - F(W^{t+1}, \gamma^{t+1}, \xi^t) \leq -\frac{\alpha}{2} \|\xi^{t+1} - \xi^t\|_2^2 \tag{4.12}$$

for all $t \in \mathbb{N}$. In addition, when $\alpha > L/2$, we have

$$\sum_{t=1}^{\infty} \|(W^{t+1}, \gamma^{t+1}) - (W^t, \gamma^t)\|_2^2 + \|\xi^{t+1} - \xi^t\|_2^2 = \sum_{t=1}^{\infty} \|(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - (W^t, \gamma^t, \xi^t)\|_2^2 < \infty, \tag{4.13}$$

which follows that $\lim_{t\to\infty} \|(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - (W^t, \gamma^t, \xi^t)\|_2 = 0$.

Proof. First we define the function

$$\begin{aligned} &L_t(W, \gamma) = \\ &\tilde{\mathcal{L}}(W^t, \gamma^t) + \langle \nabla \tilde{\mathcal{L}}(W^t, \gamma^t), (W, \gamma) - (W^t, \gamma^t) \rangle + \frac{\alpha}{2} \|(W, \gamma) - (W^t, \gamma^t)\|_2^2 + \frac{\beta}{2} \|\gamma - \xi^t\|_2^2. \end{aligned} \tag{4.14}$$

We observe that L_t is strongly convex with respect to (W, γ) with parameter α. Because $\nabla L_t(W^{t+1}, \gamma^{t+1}) = 0$ by (4.7), we use Lemma 4.1 to obtain

$$\begin{aligned} L_t(W^t, \gamma^t) \geq & L_t(W^{t+1}, \gamma^{t+1}) + \langle \nabla L_t(W^{t+1}, \gamma^{t+1}), (W^t, \gamma^t) - (W^{t+1}, \gamma^{t+1}) \rangle \\ & + \frac{\alpha}{2} \|(W^{t+1}, \gamma^{t+1}) - (W^t, \gamma^t)\|_2^2 \\ \geq & L_t(W^{t+1}, \gamma^{t+1}) + \frac{\alpha}{2} \|(W^{t+1}, \gamma^{t+1}) - (W^t, \gamma^t)\|_2^2, \end{aligned} \tag{4.15}$$

which simplifies to

$$\begin{aligned} &\tilde{\mathcal{L}}(W^t, \gamma^t) + \frac{\beta}{2} \|\gamma^t - \xi^t\|_2^2 - \alpha \|(W^{t+1}, \gamma^{t+1}) - (W^t, \gamma^t)\|_2^2 \geq \\ &\tilde{\mathcal{L}}(W^t, \gamma^t) + \langle \nabla \tilde{\mathcal{L}}(W^t, \gamma^t), (W^{t+1}, \gamma^{t+1}) - (W^t, \gamma^t) \rangle + \frac{\beta}{2} \|\gamma^{t+1} - \xi^t\|_2^2. \end{aligned} \tag{4.16}$$

Since $\nabla\tilde{\mathcal{L}}(W,\gamma)$ is Lipschitz continuous with constant L, we have

$$\begin{aligned}\tilde{\mathcal{L}}(W^{t+1},\gamma^{t+1}) \leq & \tilde{\mathcal{L}}(W^t,\gamma^t) + \langle \nabla\mathcal{L}(W^{t+1},\gamma^{t+1}), (W^{t+1},\gamma^{t+1}) - (W^t,\gamma^t)\rangle \\ & + \frac{L}{2}\|(W^{t+1},\gamma^{t+1}) - (W^t,\gamma^t)\|_2^2\end{aligned} \tag{4.17}$$

by Lemma 4.2. Combining the previous two inequalities gives us

$$\tilde{\mathcal{L}}(W^t,\gamma^t) + \frac{\beta}{2}\|\gamma^t - \xi^t\|_2^2 + \frac{L-2\alpha}{2}\|(W^{t+1},\gamma^{t+1}) - (W^t,\gamma^t)\|_2^2 \geq \tilde{\mathcal{L}}(W^{t+1},\gamma^{t+1}) + \frac{\beta}{2}\|\gamma^{t+1} - \xi^t\|_2^2.$$

Adding the term $\lambda\|\xi\|_1$ on both sides and rearranging the inequality give us (4.11).

By (4.9), we have

$$\lambda\|\xi^{t+1}\|_1 + \frac{\beta}{2}\|\gamma^{t+1} - \xi^{t+1}\|_2^2 + \frac{\alpha}{2}\|\xi^{t+1} - \xi^t\|_2^2 \leq \lambda\|\xi^t\|_1 + \frac{\beta}{2}\|\gamma^{t+1} - \xi^t\|_2^2. \tag{4.18}$$

Adding $\tilde{\mathcal{L}}(W^{t+1},\gamma^{t+1})$ on both sides and rearranging the inequality give (4.12).

Summing up (4.11) and (4.12), we have

$$\frac{2\alpha - L}{2}\|(W^{t+1},\gamma^{t+1}) - (W^t,\gamma^t)\|_2^2 + \frac{\alpha}{2}\|\xi^{t+1} - \xi^t\|_2^2 \leq F(W^t,\gamma^t,\xi^t) - F(W^{t+1},\gamma^{t+1},\xi^{t+1}). \tag{4.19}$$

Note that the left-hand side is nonnegative because we assume that $\alpha > L/2$. Summing from $t = 1$ to $t = N-1$, we have

$$\begin{aligned}\sum_{t=1}^{N-1}\frac{2\alpha - L}{2}\|(W^{t+1},\gamma^{t+1}) - (W^t,\gamma^t)\|_2^2 + \frac{\alpha}{2}\|\xi^{t+1} - \xi^t\|_2^2 &\leq F(W^1,\gamma^1,\xi^1) - F(W^N,\gamma^N,\xi^N) \\ &\leq F(W^1,\gamma^1,\xi^1),\end{aligned} \tag{4.20}$$

where the last inequality is true because $F(W, \gamma, \xi)$ is nonnegative. Taking $N \to \infty$, we have

$$\sum_{t=1}^{\infty} \|(W^{t+1}, \gamma^{t+1}) - (W^t, \gamma^t)\|_2^2 + \|\xi^{t+1} - \xi^t\|_2^2 = \sum_{t=1}^{\infty} \|(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - (W^t, \gamma^t, \xi^t)\|_2^2 < \infty.$$

$\square$

Lemma 4.4 (Relative error property)**.** *Let $\{(W^t, \gamma^t, \xi^t)\}_{t=1}^{\infty}$ be a sequence generated by Algorithm 3. Under Assumption 4.1, for any $t \in \mathbb{N}$, there exists some $w^{t+1} \in \partial F(W^{t+1}, \gamma^{t+1}, \xi^{t+1})$ such that*

$$\|w^{t+1}\|_2 \leq (3\alpha + 2L + \beta) \left\|(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - (W^t, \gamma^t, \xi^t)\right\|_2. \tag{4.21}$$

Proof. We note that

$$\nabla_W \tilde{\mathcal{L}}(W^{t+1}, \gamma^{t+1}) \in \partial_W F(W^{t+1}, \gamma^{t+1}, \xi^{t+1}), \tag{4.22a}$$

$$\nabla_\gamma \tilde{\mathcal{L}}(W^{t+1}, \gamma^{t+1}) + \beta(\gamma^{t+1} - \xi^{t+1}) \in \partial_\gamma F(W^{t+1}, \gamma^{t+1}, \xi^{t+1}), \tag{4.22b}$$

$$\lambda \partial_\xi \|\xi^{t+1}\|_1 - \beta(\gamma^{t+1} - \xi^{t+1}) \in \partial_\xi F(W^{t+1}, \gamma^{t+1}, \xi^{t+1}). \tag{4.22c}$$

By the first-order optimality conditions of (4.7) and (4.9), we obtain

$$\nabla_W \tilde{\mathcal{L}}(W^t, \gamma^t) + \alpha(W^{t+1} - W^t) = 0 \tag{4.23a}$$

$$\nabla_\gamma \tilde{\mathcal{L}}(W^t, \gamma^t) + \alpha(\gamma^{t+1} - \gamma^t) + \beta(\gamma^{t+1} - \xi^t) = 0 \tag{4.23b}$$

$$\lambda \partial_\xi \|\xi^{t+1}\|_1 + \alpha(\xi^{t+1} - \xi^t) - \beta(\gamma^{t+1} - \xi^{t+1}) \ni 0. \tag{4.23c}$$

Combining (4.22a) and (4.23a), (4.22b) and (4.23b), and (4.22c) and (4.23c), we obtain

$$\nabla_W \tilde{\mathcal{L}}(W^{t+1}, \gamma^{t+1}) - \nabla_W \tilde{\mathcal{L}}(W^t, \gamma^t) - \alpha(W^{t+1} - W^t) = w_1^{t+1} \in \partial_W F(W^{t+1}, \gamma^{t+1}, \xi^{t+1}), \tag{4.24a}$$

$$\nabla_\gamma \tilde{\mathcal{L}}(W^{t+1}, \gamma^{t+1}) - \nabla_\gamma \tilde{\mathcal{L}}(W^t, \gamma^t) - \alpha(\gamma^{t+1} - \gamma^t) - \beta(\xi^{t+1} - \xi^t) = w_2^{t+1} \in \partial_\gamma F(W^{t+1}, \gamma^{t+1}, \xi^{t+1}), \tag{4.24b}$$

$$-\alpha(\xi^{t+1} - \xi^t) = w_3^{t+1} \in \partial_\xi F(W^{t+1}, \gamma^{t+1}, \xi^{t+1}), \tag{4.24c}$$

where $w^{t+1} = (w_1^{t+1}, w_2^{t+1}, w_3^{t+1}) \in \partial F(W^{t+1}, \gamma^{t+1}, \xi^{t+1})$. As a result, by triangle inequality and Lipschitz continuity of $\nabla \tilde{\mathcal{L}}$, we have

$$\begin{aligned}
\|w_1^{t+1}\|_2 &\le \alpha \|W^{t+1} - W^t\|_2 + \|\nabla_W \tilde{\mathcal{L}}(W^{t+1}, \gamma^{t+1}) - \nabla_W \tilde{\mathcal{L}}(W^t, \gamma^t)\|_2 \\
&\le \alpha \|W^{t+1} - W^t\| + L\|(W^{t+1}, \gamma^{t+1}) - (W^t, \gamma^t)\|_2 \\
&\le (\alpha + L)\|(W^{t+1}, \gamma^{t+1}) - (W^t, \gamma^t)\|_2 \\
&\le (\alpha + L)\|(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - (W^t, \gamma^t, \xi^t)\|_2,
\end{aligned}$$

$$\begin{aligned}
\|w_2^{t+1}\|_2 &\le \alpha \|\gamma^{t+1} - \gamma^t\|_2 + \beta\|\xi^{t+1} - \xi^t\|_2 + \|\nabla_\gamma \tilde{\mathcal{L}}(W^{t+1}, \gamma^{t+1}) - \nabla_\gamma \tilde{\mathcal{L}}(W^t, \gamma^t)\|_2 \\
&\le (\alpha + L)\|(W^{t+1}, \gamma^{t+1}) - (W^t, \gamma^t)\|_2 + \beta\|\xi^{t+1} - \xi^t\|_2 \\
&\le (\alpha + \beta + L)\|(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - (W^t, \gamma^t, \xi^t)\|_2,
\end{aligned}$$

and

$$\|w_3^{t+1}\|_2 \le \alpha\|\xi^{t+1} - \xi^t\|_2 \le \alpha\|(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - (W^t, \gamma^t, \xi^t)\|_2.$$

Therefore, for all $t \in \mathbb{N}$, we have

$$\begin{aligned}
\|w^{t+1}\|_2 &\le \|w_1^{t+1}\|_2 + \|w_2^{t+1}\|_2 + \|w_3^{t+1}\|_2 \\
&\le (3\alpha + 2L + \beta)\left\|(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - (W^t, \gamma^t, \xi^t)\right\|_2.
\end{aligned}$$

$\square$

Lemma 4.5 (Subsequential convergence). *Under Assumption 4.1, if* $\{(W^t, \gamma^t, \xi^t)\}_{t=1}^{\infty}$ *generated by Algorithm 3 is bounded, then any limit point* (W^*, γ^*, ξ^*) *of* $\{(W^t, \gamma^t, \xi^t)\}_{t=1}^{\infty}$ *is a critical point of* F.

Proof. Because $\{(W^t, \gamma^t, \xi^t)\}_{t=1}^{\infty}$ is bounded, there exists a convergent subsequence $\{(W^{t_k}, \gamma^{t_k}, \xi^{t_k})\}_{k=1}^{\infty}$ such that $(W^{t_k}, \gamma^{t_k}, \xi^{t_k}) \to (W^*, \gamma^*, \xi^*)$. We will show that (W^*, γ^*, ξ^*) is a critical point of F. By Lemma 4.4, there exists $w^{t+1} \in \partial F(W^{t+1}, \gamma^{t+1}, \xi^{t+1})$ such that

$$\|w^{t+1}\|_2 \leq (3\alpha + 2L + \beta) \left\|(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - (W^t, \gamma^t, \xi^t)\right\|_2 .$$

In addition, we have

$$\lim_{t\to\infty} \left\|(W^{t+1}, \gamma^{t+1}, \xi^{t+1}) - (W^t, \gamma^t, \xi^t)\right\|_2 = 0,$$

so we have $\|w^t\|_2 \to 0$. Now we need to show that $F(W^{t_k}, \gamma^{t_k}, \xi^{t_k}) \to F(W^*, \gamma^*, \xi^*)$. By 4.1, F is overall lower semicontinuous, so we have

$$F(W^*, \gamma^*, \xi^*) \leq \liminf_{t\to\infty} F(W^t, \gamma^t, \xi^t). \tag{4.25}$$

Because F is continuous, we have

$$\lim_{k\to\infty} F(W^{t_k}, \gamma^{t_k}, \xi^{t_k}) = F(W^*, \gamma^*, \xi^*).$$

By closedness of subdifferential, we have $0 \in \partial F(W^*, \gamma^*, \xi^*)$, establishing that (W^*, γ^*, ξ^*) is a critical point of F. □

Proof of Theorem 4.1. By Lemma 6, we obtain (4.11) and (4.12), so combining them gives us

$$
\begin{aligned}
F(W^{t+1},\gamma^{t+1},\xi^{t+1}) - F(W^t,\gamma^t,\xi^t) &\le \frac{L-2\alpha}{2}\|(W^{t+1},\gamma^{t+1}) - (W^t,\gamma^t)\|_2^2 - \frac{\alpha}{2}\|\xi^{t+1}-\xi^t\|_2^2 \\
&\le \frac{L-\alpha}{2}\|(W^{t+1},\gamma^{t+1},\xi^{t+1}) - (W^t,\gamma^t,\xi^t)\|_2^2,
\end{aligned} \tag{4.26}
$$

or

$$
\frac{\alpha-L}{2}\|(W^{t+1},\gamma^{t+1},\xi^{t+1}) - (W^t,\gamma^t,\xi^t)\|_2^2 \le F(W^t,\gamma^t,\xi^t) - F(W^{t+1},\gamma^{t+1},\xi^{t+1}).
$$

Because $\alpha > L$, we satisfy condition (a) in Theorem 4.2. Because of Lemma 4.4, condition (b) of Theorem 4.2 is satisfied. Lastly, condition (c) is proven by Lemma 4.5. Satisfying all three conditions means that the sequence $\{(W^t,\gamma^t,\xi^t)\}_{t=1}^{\infty}$ converges to a critical point (W^*,γ^*,ξ^*) of F. $\square$

Chapter 5

Conclusion

In Chapter 2, we propose nonconvex sparse group lasso, a nonconvex extension of sparse group lasso. The ℓ_1 norm in sparse group lasso on the weight parameters is replaced with a nonconvex regularizer whose proximal operator is a thresholding function. Taking advantage of this property, we develop a new algorithm to optimize loss functions regularized with nonconvex sparse group lasso for CNNs in order to attain a sparse network with competitive accuracy. We compare the proposed family of regularizers with various baseline methods on MNIST and CIFAR 10/100 on different CNNs. The experimental results demonstrate that in general, nonconvex sparse group lasso generates a more accurate and/or more compressed CNN than does group lasso. In addition, we compare our proposed algorithm to direct stochastic gradient descent and proximal gradient descent on Lenet-5 trained on MNIST. The results show that the proposed algorithm to solve SGL_1 yields a satisfactorily sparse network with lower test error than do the other two algorithms. According to the numerical results, there is no single sparse regularizer that outperforms all other on any CNN trained on a given dataset. One regularizer may perform well in one case while it may perform worse on a different case. Due to the myriad of sparse regularizers to select from and the various parameters to tune, especially for one CNN trained on a given dataset, one direction is to develop an automatic machine learning framework that efficiently selects the right regularizer and parameters. In recent works, automatic machine learning can be represented as a matrix completion problem [227] and a statistical learning problem [82]. These frameworks can be adapted for selecting the best sparse regularizer, thus saving time for users who are training sparse CNNs.

In Chapter 3, we improve NS by replacing the ℓ_1 regularizer with a sparse, nonconvex regularizer for penalizing the scaling factors in the batch normalization layers. In particular, we investigate $\ell_p (0 < p < 1)$, Tℓ_1, MCP, and SCAD. We apply the proposed methods onto VGG-19, DenseNet-40, and ResNet-164 trained on CIFAR 10/100 and SVHN. We observe that ℓ_p and Tℓ_1 save more on parameters and FLOPs than ℓ_1 with a slight decrease in test accuracy. In addition, Tℓ_1, especially $a = 0.5$, preserves model accuracy against channel pruning. NS with Tℓ_1 is competitive against VCP, another NS variant robust against channel pruning. To attain better accuracy than ℓ_1 while having similar compression, MCP and SCAD perform the best job after their models are pruned and retrained, especially for VGG-19 and DenseNet-40.

In Chapter 4, we develop an alternative NS algorithm called proximal NS that trains a CNN towards a sparse, accurate structure, making fine tuning optional. Our experiments demonstrate that proximal NS can better compress CNNs with accuracy slightly less than the original baseline. One limitation of proximal NS is that its fine-tuned accuracy is less than its original NS counterpart. Hence, we plan to investigate how to improve the algorithm to yield better fine-tuned accuracy.

For future directions, we aim to generalize the NS algorithms to layer normalization [12] and group normalization [218]. In addition, we shall study proximal cooperative neural architecture search [226, 224] and include nonconvex, sparse regularizers, such as $\ell_1 - \ell_2$ [230] and transformed ℓ_1 [247], in proximal NS.

Part II

Image Segmentation

Chapter 6

Introduction

6.1 Motivation and Related Works

Image segmentation is a prevalent, challenging problem in computer vision, aiming to partition an image into several regions that represent specific objects of interest. Each partitioned region has similar features such as edges, colors, and intensities. One segmentation method is the Mumford-Shah (MS) model [163] well-known for its robustness to noise. It finds the optimal piecewise-smooth approximation of an input image that incorporates region and boundary information to facilitate segmentation. Given a bounded, open set $\Omega \subset \mathbb{R}^2$ with Lipschitz boundary and an observed image $f : \Omega \to [0, 1]$, the MS model can be expressed as an energy minimization problem,

$$\min_{u,\Gamma} \frac{\lambda}{2} \int_{\Omega} (f - u)^2 + \frac{\mu}{2} \int_{\Omega \setminus \Gamma} |\nabla u|^2 + \text{Length}(\Gamma), \tag{6.1}$$

where $\lambda, \mu > 0$ are weighing parameters, $\Gamma \subset \Omega$ is a compact curve representing the boundaries separating disparate objects, and $u : \Omega \to \mathbb{R}$ is an approximation of f that is smooth in $\Omega \setminus \Gamma$ but possibly discontinuous across Γ. The middle term $\int_{\Omega \setminus \Gamma} |\nabla u|^2$ ensures that u is piecewise smooth, or more specifically differentiable on $\Omega \setminus \Gamma$. The last term "Length(Γ)" measures the perimeter of Γ that can be mathematically expressed as $\mathcal{H}^1(\Gamma)$, which is the 1-dimensional Hausdorff measure in $\mathbb{R}^2$ [17]. It is challenging to solve for the minimization problem (6.1) due to its nonconvex nature and difficulties in discretizing the unknown set of boundaries. Pock et al. [177] proposed a convex relaxation of (6.1) together with an efficient primal-dual algorithm. For the boundary issue, one early attempt involved a sequence of (local) elliptic variational problems [7] to approximate the energy functional (6.1). Later, nonlocal approximations were adopted in [76, 34] and a finite element approximation was developed in [37].

By relaxing u from piecewise smooth to piecewise constant, Chan and Vese (CV) [44] proposed a two-phase model to segment the image domain Ω into two regions that are inside and outside of the curve Γ. The curve can be represented by a level-set function ϕ that is Lipschitz continuous and satisfies

$$\begin{cases} \phi(x) > 0 & \text{if } x \text{ is inside } \Gamma, \\ \phi(x) = 0 & \text{if } x \text{ is at } \Gamma, \\ \phi(x) < 0 & \text{if } x \text{ is outside } \Gamma. \end{cases}$$

The Heaviside function $H(\phi)$ is defined by $H(\phi) = 1$ if $\phi \geq 0$ and $H(\phi) = 0$ otherwise. The CV model is given by

$$\min_{c_1,c_2,\phi} E_{CV}(c_1, c_2, \phi) := \lambda \int_\Omega |f - c_1|^2 H(\phi) + \lambda \int_\Omega |f - c_2|^2 (1 - H(\phi)) + \nu \int_\Omega |\nabla H(\phi)|, \tag{6.2}$$

where λ, ν are two positive parameters and $c_1, c_2 \in \mathbb{R}$ are mean intensity values of the two regions. Originally, the CV model (6.2) was solved by finite difference methods [43, 74].

Chan et al. [42] proposed a convex relaxation of the CV model, formulated as

$$\min_{u\in[0,1],c_1,c_2} \lambda \int_\Omega (f-c_1)^2 u + (f-c_2)^2(1-u) + \int_\Omega |\nabla u|. \tag{6.3}$$

The segmented regions can be defined by thresholding u as follows:

$$\text{inside}(\Gamma) = \{(x,y)\in\Omega : u(x,y) > \tau\}, \quad \text{outside}(\Gamma) = \{(x,y)\in\Omega : u(x,y) \leq \tau\},$$

with a chosen constant $\tau \in [0,1]$. Since the objective function in (6.3) is convex with respect to u, it can be minimized using popular convex optimization algorithms, such as split Bregman [78], alternating direction method of multipliers (ADMM) [21, 73], and primal-dual hybrid gradient (PDHG) [38, 68]. As a result, (6.3) inspired various segmentation models [14, 36, 105, 117, 179, 236, 237, 240] that can be solved by convex optimization. As an alternative to the level-set formulation (6.2), a diffuse-interface approximation to the CV model was considered in [66], which can be solved efficiently by the Merrimen-Bence-Osher scheme [161].

The CV model can be extended to vector-valued images [43] and to multiphase segmentation [23, 204]. The vector-valued extension is straightforward, i.e., replacing f with a vector-valued input $\mathbf{f}: \Omega \to \mathbb{R}^C$ and replacing c_1, c_2 with vector-valued constants $\mathbf{c}_1, \mathbf{c}_2 \in \mathbb{R}^C$, where C is the number of channels in an image. The multiphase CV model relies on $\log_2(N)$ level-set functions to partition Ω into N regions $\{\Omega_i\}_{i=1}^N$, and hence most CV-based multiphase segmentation methods are limited to power-of-two number of regions so that $\log_2(N)$ is an integer. There are two approaches that can deal with an arbitrary number of regions. One approach represents each region by a single level-set function [187], which unfortunately causes vacuums and overlapping regions to appear. The other approach defines regions by membership functions, referred to as fuzzy region (FR) competition [119].

Another approach of finding a piecewise-constant solution to the MS model is the smoothing-and-thresholding (SaT) framework [29]. In SaT, one first finds a smoothed image u by solving a convex variant of the MS model:

$$\min_u \frac{\lambda}{2}\int_\Omega (f - Au)^2 + \frac{\mu}{2}\int_\Omega |\nabla u|^2 + \int_\Omega |\nabla u|, \tag{6.4}$$

where $\lambda > 0, \mu \geq 0$, and A is a linear operator. Specifically, A is the identity operator if one wants to segment a noisy image f, while it can be a blurring operator for the desire of segmenting a blurry and noisy image f. The middle term $\int_\Omega |\nabla u|^2$ extends the piecewise-smooth regularization $\int_{\Omega\setminus\Gamma} |\nabla u|^2$ in (6.1) to the entire image domain Ω. The last term $\int_\Omega |\nabla u|$ is the total variation (TV) that approximates the length term in (6.1) based on the coarea formula [42]. After obtaining a piecewise-smooth approximation, one segments the image domain into k regions by thresholding u with $k-1$ appropriately selected values. SaT has several advantages over the MS model (6.1) and the CV model (6.2). First, the smoothing stage involves a strictly convex problem (6.4) to guarantee a unique solution that can be found by numerous convex optimization algorithms. Second, the thresholding stage allows for segmenting any number of regions via a clustering algorithm such as k-means clustering [86, 9]. Lastly, thresholding is independent of smoothing; in other words, thresholding can be adjusted to obtain a visually appealing segmentation without going back to smoothing again. SaT was adapted to segment images corrupted by Poisson or multiplicative Gamma noise [39]. For color images, SaT evolved into the "smoothing, lifting, and thresholding" (SLaT) framework [28]. The additional lifting stage in SLaT adds the Lab (perceived lightness, red-green and yellow-blue) color space to provide more discriminatory information than the conventional RGB color space with correlated color channels. The idea of lifting can also improve image segmentation of grayscale images whose pixel intensities vary dramatically, referred to as *intensity inhomogeneity*. Traditional methods that deal with inhomogeneity include preprocessing [91] and intensity correction [118, 210]. By generating an additional image channel [124], SaT/SLaT yields better segmentation results for grayscale images that suffer from intensity inhomogeneity.

6.2 Weighted Anisotropic–Isotropic Total Variation

In (6.3)-(6.4), the TV term $\|\nabla u\|_1 = \int_\Omega |\nabla u|$ approximates the length of the curves that partition the segmented regions. Furthermore, it is the tightest convex relaxation of the jump term $\|\nabla u\|_0$, which counts the number of jump discontinuities. When u is piecewise constant, $\|\nabla u\|_0$ is exactly the total arc length of the curves [194]. Unfortunately, minimizing $\|\nabla u\|_0$ is an NP-hard combinatorial problem, and it is often replaced by $\|\nabla u\|_1$ that is algorithmically and theoretically easier to work with. Numerically, $\|\nabla u\|_1$ can be approximated isotropically [186] or anisotropically [53, 65]:

$$J_{\text{iso}}(u) = \int_\Omega \sqrt{|D_x u|^2 + |D_y u|^2}, \tag{6.5}$$

$$J_{\text{ani}}(u) = \int_\Omega |D_x u| + |D_y u|, \tag{6.6}$$

where D_x and D_y denote the horizontal and vertical partial derivative operators, respectively.

In order to better approximate $\|\nabla u\|_0$, we consider the weighted anisotropic–isotropic TV (AITV),

$$J_{\text{ani}}(u) - \alpha J_{\text{iso}}(u) = \int_\Omega |D_x u| + |D_y u| - \alpha\sqrt{|D_x u|^2 + |D_y u|^2} \tag{6.7}$$

with $\alpha \in [0, 1]$. The AITV term was inspired by recent successes of $L_1 - L_2$ minimization [61, 138, 139, 140, 229, 230] in compressed sensing. Compared with L_1, L_p for $p \in (0, 1)$ [48, 113, 222], and L_0 [201], the $L_1 - L_2$ penalty was shown to have the best performance in recovering sparse solutions when the sensing matrix is highly coherent or violates the restricted isometry property [32]. Figure 6.1 compares L_0, L_1, and $L_1 - \alpha L_2$ by their contour

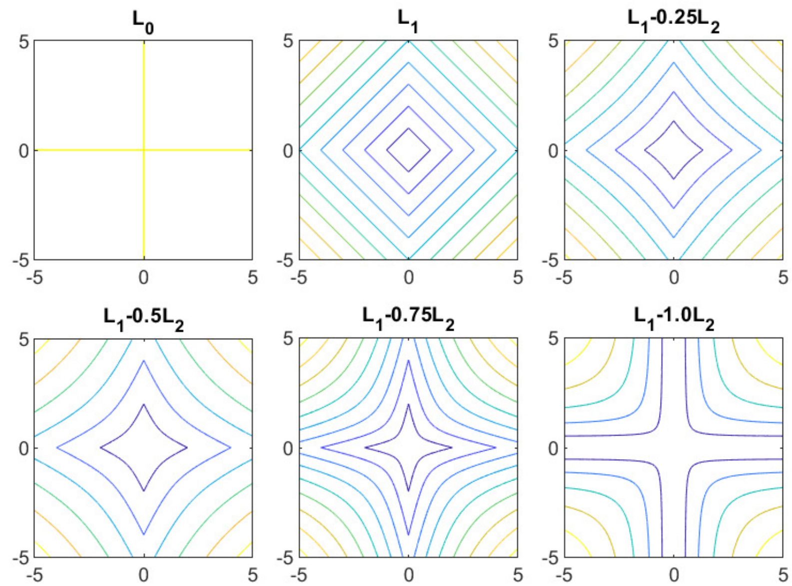


Figure 6.1: Contour lines of $\|x\|_0$ (L_0) and $\|x\|_1 - \alpha\|x\|_2$ ($L_1 - \alpha L_2$), where $x \in \mathbb{R}^2$ and $\alpha \in \{0, 0.25, 0.5, 0.75, 1.0\}$. As α increases, the contour lines of $L_1 - \alpha L_2$ are closer to the ones of L_0.

lines in 2D. We observe that as α increases, the contour lines of $L_1 - \alpha L_2$ are bending more inward and closer to the ones of L_0. This phenomenon illustrates that $L_1 - \alpha L_2$ can encourage sparsity, and the constant α acts like a parameter controlling to what extent. By applying $L_1 - \alpha L_2$ on the gradient, Lou et al. [141] proposed AITV with a difference-of-convex algorithm (DCA) [115, 174, 175] for image denoising, deconvolution, and MRI reconstruction. Later, Li et al. [123] demonstrated the robustness of AITV with respect to impulsive noise corruption of the data. Both works [123, 141] showed that AITV preserves sharper image edges than the anisotropic TV. Moreover, AITV is preferred over the isotropic TV that tends to blur oblique edges [19, 56].

As edges are defined by gradient vectors, it is expected that AITV ($L_1 - \alpha L_2$) should produce sparser gradients and maintain sharper edges compared to TV (L_1). A preliminary work that replaced $\|\nabla u\|_1$ by AITV in (6.3) was conducted by Park et al. [172], showing better segmentation results than TV. However, this approach was limited to pre-determined values of c_1/c_2, grayscale images, and two-phase segmentation (rather than multiphase).

6.3 Organization of Part II

In Part II, we propose incorporating the AITV regularizer in three classes of image segmentation models. In Chapter 7, we propose and analyze the AITV variant of the CV and FR models and develop DCAs to solve them. This chapter is based on [24]. Afterward, Chapter 8 proposes an AITV variant of the SaT/SLaT framework. In this chapter, we develop an efficient ADMM algorithm to solve AITV-regularized MS model. Finally, we conclude in Chapter 9.

Chapter 7

A Weighted Difference of Anisotropic and Isotropic Total Variation for Relaxed Mumford-Shah Image Segmentation

In this chapter, we propose to incorporate the AITV term into both CV and FR models together with an extension to color image segmentation. To solve these models, we develop an alternating minimization framework that involves DCA and PDHG with linesearch (PDHGLS) [156]. We provide convergence analysis of the proposed algorithms. Experimentally, we compare the proposed models with the classic convex approaches and other segmentation methods to showcase the effectiveness and robustness of the AITV penalty. The major contributions of this work are threefold:

- We study the AITV regularization comprehensively in image segmentation, including grayscale/color image and multiphase segmentation.

- We propose an efficient algorithm that combines DCA and PDHGLS with guaranteed convergence. To the best of our knowledge, this paper pioneers the implementation of PDHGLS in image segmentation.
- We conduct extensive experiments to demonstrate the effect of the constant α in AITV on the segmentation performance and the robustness to impulsive noise. We compare the results with the two-stage segmentation methods.

7.1 Notations

For simplicity, we adopt the discrete notations for images and related models. The space $\mathbb{R}^n$ is equipped with the standard inner product $\langle x, y\rangle = \sum_{i=1}^n x_i y_i$ and standard Euclidean norm $\|x\|_2 = \sqrt{\langle x, x\rangle}$ for $x, y \in \mathbb{R}^n$.

Without loss of generality, an image is represented as an $m \times n$ matrix, i.e. the image domain is $\Omega = \{1, 2, \ldots, m\} \times \{1, 2, \ldots, n\}$. We denote $X := \mathbb{R}^{m\times n}$ and the all-ones matrix in X as $\mathbb{1}$. The vector space X is equipped with following inner product and norm:

$$\langle u, v\rangle_X = \sum_{i=1}^m \sum_{j=1}^n u_{i,j} v_{i,j}, \quad \|u\|_X = \sqrt{\sum_{i=1}^m \sum_{j=1}^n u_{i,j}^2} \qquad \forall u, v \in X.$$

We denote D_x, D_y by the horizontal and vertical partial derivative operators, respectively, i.e.,

$$(D_x u)_{i,j} = \begin{cases} u_{i,j+1} - u_{i,j} & \text{if } 1 \le j \le n-1, \\ u_{i,1} - u_{i,n} & \text{if } j = n, \end{cases}$$
$$(D_y u)_{i,j} = \begin{cases} u_{i+1,j} - u_{i,j} & \text{if } 1 \le i \le m-1, \\ u_{1,j} - u_{m,j} & \text{if } i = m. \end{cases}$$

Let $Y \coloneqq X \times X$. Then the discrete gradient operator $D : X \to Y$ is defined as

$$(Du)_{i,j} = ((D_x u)_{i,j}, (D_y u)_{i,j}) \in Y.$$

For any $p = (p_x, p_y), q = (q_x, q_y) \in Y$, the inner product on Y is defined by

$$\langle p, q \rangle_Y = \langle p_x, q_x \rangle_X + \langle p_y, q_y \rangle_X,$$

and the norms on Y are

$$\|p\|_Y = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |(p_x)_{i,j}|^2 + |(p_y)_{i,j}|^2}, \qquad \|p\|_1 = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(|(p_x)_{i,j}| + |(p_y)_{i,j}| \right),$$
$$\|p\|_{2,1} = \sum_{i=1}^{m} \sum_{j=1}^{n} \sqrt{|(p_x)_{i,j}|^2 + |(p_y)_{i,j}|^2} = \sum_{i=1}^{m} \sum_{j=1}^{n} \|((p_x)_{i,j}, (p_y)_{i,j})\|_2.$$

We use a bold letter to denote a 3D tensor, e.g., $\mathbf{u} = (u_1, u_2, \dots, u_N) \in X^N$. We further denote $\mathbf{u}_{<k} \coloneqq (u_1, \dots, u_{k-1})$ and $\mathbf{u}_{>k} \coloneqq (u_{k+1}, \dots, u_N)$ for $1 \leq k \leq N$. The notations $\mathbf{u}_{\leq k}$ and $\mathbf{u}_{\geq k}$ are defined similarly by including u_k. Note that $\mathbf{u}_{<1}$ and $\mathbf{u}_{>N}$ are null or empty variables.

7.2 Anisotropic-Isotropic Chan-Vese Model

Let $f \in X$ be an observed image. Suppose the image domain Ω has $N = 2^M$ non-overlapping regions, i.e. $\Omega = \bigcup_{i=1}^{N} \Omega_i$ and $\Omega_i \cap \Omega_j = \emptyset$ for each $i \neq j$. Let $\mathbf{u} = (u_1, \dots, u_M) \in X^M$ and $\mathbf{c} = (c_1, \dots, c_N) \in \mathbb{R}^N$. We propose an AITV-regularized Chan-Vese (AICV) model for multiphase segmentation as follows:

$$\min_{\substack{\mathbf{u} \in \mathcal{B} \\ \mathbf{c} \in \mathbb{R}^N}} \sum_{k=1}^{M} \left(\|Du_k\|_1 - \alpha \|Du_k\|_{2,1} \right) + \lambda \sum_{\ell=1}^{N} \langle f_\ell(\mathbf{c}), R_\ell(\mathbf{u}) \rangle_X, \tag{7.1}$$

where $\mathcal{B} = \left\{\mathbf{u} \in X^M : (u_k)_{i,j} \in \{0,1\} \forall i,j,k\right\}$, $f_\ell(\mathbf{c}) = (f - c_\ell \mathbb{1})^2$ with square defined elementwise, and $R_\ell(\mathbf{u})$ is a function of $\mathbf{u}$ related to the region Ω_ℓ such that

$$R_\ell(\mathbf{u})_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in \Omega_\ell, \\ 0 & \text{if } (i,j) \notin \Omega_\ell, \end{cases}$$

with $\sum_{\ell=1}^N R_\ell(\mathbf{u}) = \mathbb{1}$. Specifically when $N = 2$ ($M = 1$), we have $R_1(\mathbf{u}) = u_1$ and $R_2(\mathbf{u}) = \mathbb{1} - u_1$. When $N = 4$ ($M = 2$), we have

$$\begin{aligned} R_1(\mathbf{u})_{i,j} &= (u_1)_{i,j}(u_2)_{i,j}, & R_2(\mathbf{u})_{i,j} &= (u_1)_{i,j}[1 - (u_2)_{i,j}], \\ R_3(\mathbf{u})_{i,j} &= [1 - (u_1)_{i,j}](u_2)_{i,j}, & R_4(\mathbf{u})_{i,j} &= [1 - (u_1)_{i,j}][1 - (u_2)_{i,j}]. \end{aligned}$$

When $N = 8$ ($M = 3$), we have

$$\begin{aligned} R_1(\mathbf{u})_{i,j} &= (u_1)_{i,j}(u_2)_{i,j}(u_3)_{i,j}, & R_2(\mathbf{u})_{i,j} &= (u_1)_{i,j}(u_2)_{i,j}[1 - (u_3)_{i,j}], \\ R_3(\mathbf{u})_{i,j} &= (u_1)_{i,j}[1 - (u_2)_{i,j}](u_3)_{i,j}, & R_4(\mathbf{u})_{i,j} &= (u_1)_{i,j}[1 - (u_2)_{i,j}][1 - (u_3)_{i,j}], \\ R_5(\mathbf{u})_{i,j} &= [1 - (u_1)_{i,j}](u_2)_{i,j}(u_3)_{i,j}, & R_6(\mathbf{u})_{i,j} &= [1 - (u_1)_{i,j}](u_2)_{i,j}[1 - (u_3)_{i,j}], \\ R_7(\mathbf{u})_{i,j} &= [1 - (u_1)_{i,j}][1 - (u_2)_{i,j}](u_3)_{i,j}, & R_8(\mathbf{u})_{i,j} &= [1 - (u_1)_{i,j}][1 - (u_2)_{i,j}][1 - (u_3)_{i,j}]. \end{aligned}$$

For $N = 2^M$ with $M \geq 4$, R_ℓ depends on ℓ's binary representation to decide whether to include u_k or $\mathbb{1} - u_k$ as a factor in R_ℓ.

Due to the binary constraint set $\mathcal{B}$, (7.1) is a nonconvex optimization problem, thus numerically difficult to solve. We relax the binary constraint $\{0, 1\}$ by a $[0, 1]$ box constraint, which in turn has $R_\ell(\mathbf{u})_{i,j} \in [0, 1]$. In particular, we rewrite (7.1) as an unconstrained formulation by introducing the indicator function

$$\chi_U(u) = \begin{cases} 0 & \text{if } u_{i,j} \in [0,1] \text{ for all } i,j, \\ +\infty & \text{otherwise.} \end{cases}$$

Hence, a relaxed model of (7.1) can be expressed as

$$\min_{\substack{\mathbf{u}\in X^M \\ \mathbf{c}\in\mathbb{R}^N}} \tilde{F}(\mathbf{u},\mathbf{c}) \coloneqq \sum_{k=1}^{M} \Big(\|Du_k\|_1 - \alpha \|Du_k\|_{2,1} + \chi_U(u_k) \Big) + \lambda \sum_{\ell=1}^{N} \langle f_\ell(\mathbf{c}), R_\ell(\mathbf{u})\rangle_X. \tag{7.2}$$

7.2.1 Numerical Algorithm

We propose an alternating minimization algorithm to find a solution of (7.2) with the following framework:

$$\mathbf{u}^{t+1} \in \arg\min_{\mathbf{u}} \tilde{F}(\mathbf{u},\mathbf{c}^t), \tag{7.3}$$

$$\mathbf{c}^{t+1} \in \arg\min_{\mathbf{c}} \tilde{F}(\mathbf{u}^{t+1},\mathbf{c}), \tag{7.4}$$

where t counts the (outer) iterations. Below, we discuss how to solve each subproblem.

We start with the **c**-subproblem (7.4), as it is simpler than the other. Notice that we can solve c_ℓ separately for each $\ell = 1,\dots,N$, i.e.,

$$c_\ell^{t+1} \in \arg\min_{c_\ell} \lambda \langle f_\ell(\mathbf{c}), R_\ell(\mathbf{u}^{t+1})\rangle_X = \arg\min_{c_\ell} \lambda \sum_{i=1}^{m}\sum_{j=1}^{n} (f_{i,j} - c_\ell)^2 R_\ell(\mathbf{u}^{t+1})_{i,j}. \tag{7.5}$$

If $\sum_{i=1}^m \sum_{j=1}^n R_\ell(\mathbf{u}^{t+1})_{i,j} \neq 0$, we differentiate the objective function in (7.5) with respect to c_ℓ, set the derivative equal to zero, and solve for c_ℓ; otherwise, since the objective function does not depend on c_ℓ, the solution can take on any value, so we set the solution to 0 as a default. In summary, there is a closed-form solution to (7.5) for updating c_ℓ^{t+1}, i.e.,

$$c_\ell^{t+1} = \begin{cases} \dfrac{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{i,j} R_\ell(\mathbf{u}^{t+1})_{i,j}}{\sum_{i=1}^{m}\sum_{j=1}^{n} R_\ell(\mathbf{u}^{t+1})_{i,j}} & \text{if } \sum_{i=1}^{m}\sum_{j=1}^{n} R_\ell(\mathbf{u}^{t+1})_{i,j} \neq 0, \\ 0 & \text{if } \sum_{i=1}^{m}\sum_{j=1}^{n} R_\ell(\mathbf{u}^{t+1})_{i,j} = 0. \end{cases} \tag{7.6}$$

The formula (7.6) implies that c_ℓ^{t+1} is the mean intensity value of the region $\Omega_\ell \subset \Omega$ at the $(t+1)$-th iteration.

The $\mathbf{u}$-subproblem (7.3) is separable with respect to each k, i.e.,

$$u_k^{t+1} \in \arg\min_{u_k} \|Du_k\|_1 - \alpha\|Du_k\|_{2,1} + \chi_U(u_k) + \lambda\langle r_k(\mathbf{c}^t, \mathbf{u}_{<k}^{t+1}, \mathbf{u}_{>k}^t), u_k\rangle_X, \tag{7.7}$$

where $r_k(\mathbf{c}^t, \mathbf{u}_{<k}^{t+1}, \mathbf{u}_{>k}^t)$ is a multivariate polynomial of $(\mathbf{u}_{<k}^{t+1}, \mathbf{u}_{>k}^t)$ obtained by rewriting $\sum_{\ell=1}^N \langle f_\ell(\mathbf{c}), R_\ell(\mathbf{u})\rangle_X$ in (7.2) and getting the coefficients in front of u_k. Because a general form of r_k is complicated, we provide some specific examples in smaller dimensions. When $N = 2$ $(M = 1)$, we have $r_1(\mathbf{c}, \mathbf{u}_{<1}, \mathbf{u}_{>1})_{i,j} = (f_{i,j} - c_1)^2 - (f_{i,j} - c_2)^2$; when $N = 4$ $(M = 2)$, we have

$$\begin{aligned} r_1(\mathbf{c}, \mathbf{u}_{<1}, \mathbf{u}_{>1})_{i,j} &= \left[(f_{i,j} - c_1)^2 - (f_{i,j} - c_2)^2 - (f_{i,j} - c_3)^2 + (f_{i,j} - c_4)^2\right](u_2)_{i,j}, \\ &\quad + (f_{i,j} - c_2)^2 - (f_{i,j} - c_4)^2 \\ r_2(\mathbf{c}, \mathbf{u}_{<2}, \mathbf{u}_{>2})_{i,j} &= \left[(f_{i,j} - c_1)^2 - (f_{i,j} - c_2)^2 - (f_{i,j} - c_3)^2 + (f_{i,j} - c_4)^2\right](u_1)_{i,j} \\ &\quad + (f_{i,j} - c_3)^2 - (f_{i,j} - c_4)^2. \end{aligned}$$

In order to minimize (7.7), we apply a descent algorithm called DCA [115, 174, 175] for solving a difference-of-convex (DC) optimization problem of the form $\min_{u \in X} g(u) - h(u)$, where g and h are proper, lower semicontinuous, and strongly convex functions. The algorithm consists of two steps per iteration with u^0 as initialization:

$$\begin{cases} v^t & \in \partial h(u^t), \\ u^{t+1} & \in \underset{u\in X}{\arg\min}\, g(u) - \langle v^t, u\rangle_X. \end{cases} \tag{7.8}$$

For each $k = 1, \ldots, M$, we can express (7.7) as a DC function $g(u_k) - h(u_k)$ with

$$\begin{cases} g(u_k) & = \|Du_k\|_1 + \chi_U(u_k) + \lambda\langle r_k(\mathbf{c}^t, \mathbf{u}^{t+1}_{<k}, \mathbf{u}^t_{>k}), u_k\rangle_X + c\|u_k\|_X^2, \\ h(u_k) & = \alpha\|Du_k\|_{2,1} + c\|u_k\|_X^2, \end{cases} \tag{7.9}$$

where $c > 0$ enforces strong convexity on the functions g and h. Experimentally, c can be chosen arbitrarily small for better performance. We then compute the subgradient of $h(u)$, i.e.,

$$\alpha\frac{D_x^\top D_x u + D_y^\top D_y u}{\sqrt{|D_x u|^2 + |D_y u|^2}} + 2cu \in \partial h(u).$$

Therefore, the u-subproblem in (7.8) can be expressed as

$$\begin{aligned} u_k^{t+1} = \underset{u_k}{\arg\min} \|Du_k\|_1 &+ \chi_U(u_k) + \lambda\langle r_k(\mathbf{c}^t, \mathbf{u}^{t+1}_{<k}, \mathbf{u}^t_{>k}), u_k\rangle_X + c\|u_k\|_X^2 \\ &- \alpha\langle Du_k, q_k^t\rangle_Y - 2c\langle u_k, u_k^t\rangle_X, \end{aligned} \tag{7.10}$$

where $q_k^t := ((q_x)_k^t, (q_y)_k^t) = (D_x u_k^t, D_y u_k^t)/\sqrt{|D_x u_k^t|^2 + |D_y u_k^t|^2}$. Note that we compute q_k^t elementwise and adopt the convention that if the denominator is zero at some point, the corresponding q_k^t value is set to zero, which aligns with the subgradient definition. To solve the convex problem (7.10), we apply the PDHG algorithm [38, 68, 256] since it was demonstrated in [38] that PDHG solves imaging models with the TV term [186] efficiently.

In general, the PDHG algorithm [38, 68, 256] targets at a saddle-point problem

$$\min_u \max_v \Psi(u) + \langle Au, v\rangle_Y - \Phi(v),$$

where Ψ, Φ are convex functions and A is a linear operator. The PDHG algorithm is outlined as

$$\begin{aligned}
u^{\eta+1} &= (I + \tau\partial\Psi)^{-1}(u^\eta - \tau A^\top v^\eta),\\
\bar{u}^{\eta+1} &= u^{\eta+1} + \theta(u^{\eta+1} - u^\eta),\\
v^{\eta+1} &= (I + \sigma\partial\Phi)^{-1}(v^\eta + \sigma A\bar{u}^{\eta+1}),
\end{aligned}$$

with $\tau, \sigma > 0, \theta \in [0, 1]$. The inverse is defined by the proximal operator, i.e.,

$$(I + \tau\partial\Psi)^{-1}(z) = \min_u \left(\Psi(u) + \frac{\|u - z\|_X^2}{2\tau}\right),$$

and similarly for $(I + \sigma\partial\Phi)^{-1}$.

In order to apply PDHG for the u_k-problem in (7.10), we define its saddle-point formulation:

$$\begin{aligned}
\min_{u_k} \max_{(p_x)_k,(p_y)_k} &\langle D_x u_k, (p_x)_k\rangle_X + \langle D_y u_k, (p_y)_k\rangle_X + \chi_U(u_k)\\
&+ \lambda\langle r_k(\mathbf{c}^t, \mathbf{u}_{<k}^{t+1}, \mathbf{u}_{>k}^t), u_k\rangle_X + c\|u_k\|_X^2 - \alpha\langle Du_k, q_k^t\rangle_Y - 2c\langle u_k,\, u_k^t\rangle_X\\
&- \chi_P((p_x)_k) - \chi_P((p_y)_k),
\end{aligned} \tag{7.11}$$

where $(p_x)_k, (p_y)_k$ are dual variables of $D_x u_k, D_y u_k$, and $P = \{p : |p_{i,j}| \leq 1\ \forall i, j\}$ is a convex set. Please refer to [35, 38] for the derivation of the saddle-point formulation in more details. Then we have

$$\begin{aligned}
\Psi_{k,t}(u_k) = \chi_U(u_k) &+ \lambda\langle r_k(\mathbf{c}^t, \mathbf{u}_{<k}^{t+1}, \mathbf{u}_{>k}^t), u_k\rangle_X + c\|u_k\|_X^2,\\
&- \alpha\langle Du_k, q_k^t\rangle_Y - 2c\langle u_k,\, u_k^t\rangle_X,
\end{aligned}$$

$$Au_k = (D_x u_k, D_y u_k)$$
$$\Phi((p_x)_k, (p_y)_k) = \chi_P((p_x)_k) + \chi_P((p_y)_k).$$

With the initial condition $u_k^{t,0} = u_k^t$, the u-subproblem can be computed as

$$\begin{aligned} u_k^{t,\eta+1} &= (I + \tau\partial\Psi_{k,t})^{-1}\left(u_k^{t,\eta} - \tau\left(D_x^\top(p_x)_k^\eta + D_y^\top(p_y)_k^\eta\right)\right) \\ &= \min_{0\leq (u_k)_{i,j}\leq 1}\Bigg\{\lambda\langle r_k(\mathbf{c}^t, \mathbf{u}_{<k}^{t+1}, \mathbf{u}_{>k}^t), u_k\rangle_X + c\|u_k\|_X^2 \\ &\qquad - \alpha\langle Du_k, q_k^t\rangle_Y - 2c\langle u_k,\, u_k^t\rangle_X \\ &\qquad + \frac{\|u_k - \left(u_k^{t,\eta} - \tau\left(D_x^\top(p_x)_k^\eta + D_y^\top(p_y)_k^\eta\right)\right)\|_X^2}{2\tau}\Bigg\}, \end{aligned} \tag{7.12}$$

where η indexes the inner iteration, as opposed to t for the outer iteration. To solve (7.12), we derive a closed-form solution that is similar to the one for the u-subproblem of (6.3) determined in [77]. In particular, we observe that the objective function in (7.12) is proper, continuous, and strongly convex with respect to u_k, so it has a unique minimizer. By ignoring the constraint and differentiating the objective function in (7.12) with respect to u_k, we obtain

$$\tilde{u}_k^{t,\eta+1} = \frac{2cu_k^t + \frac{1}{\tau}u_k^{t,\eta}}{2c + \frac{1}{\tau}} - \frac{\lambda r_k(\mathbf{c}^t, \mathbf{u}_{<k}^{t+1}, \mathbf{u}_{>k}^t) - \alpha D^\top q_k^t + (D_x^\top(p_x)_k^\eta + D_y^\top(p_y)_k^\eta)}{2c + \frac{1}{\tau}}.$$

If $(\tilde{u}_k^{t,\eta+1})_{i,j}$ lies in the interval $[0,1]$, then the (i,j)-entry of the unique minimizer also coincides with the minimizer of the constrained problem (7.12). If $(\tilde{u}_k^{t,\eta+1})_{i,j}$ is outside of the interval, then the (i,j)-entry of the unique minimizer lies at the interval endpoint closest to the unconstrained minimizer due to the quadratic objective function. As a result, we project $\tilde{u}_k^{t,\eta+1}$ onto $[0,1]$, leading to a closed-form solution for $u_k^{t,\eta+1}$:

$$u_k^{t,\eta+1} = \min\{\max\{\tilde{u}_k^{t,\eta+1}, 0\}, 1\}, \tag{7.13}$$

where min and max are executed elementwise.

It is straightforward to derive closed-form solutions for $(p_x)_k, (p_y)_k$ in (7.11) given by

$$\begin{aligned}(p_x)_k^{\eta+1} &= \mathrm{Proj}_P((p_x)_k^\eta + \sigma D_x \bar{u}_k^{t,\eta+1}),\\ (p_y)_k^{\eta+1} &= \mathrm{Proj}_P((p_y)_k^\eta + \sigma D_y \bar{u}_k^{t,\eta+1})\end{aligned} \tag{7.14}$$

with $\bar{u}_k^{t,\eta+1} = u_k^{t,\eta+1} + \theta(u_k^{t,\eta+1} - u_k^{t,\eta})$ and $\mathrm{Proj}_P(p) = \frac{p}{\max\{|p|,1\}}$. We see that (7.13) is projected gradient descent of the primal variable u with entrywise box constraint $[0, 1]$, while (7.14) is projected gradient ascent of the dual variable (p_x, p_y) that is constrained to the set P. The update order between the primal variable $u_k^{t,\eta}$ and the dual variables $(p_x)_k^\eta, (p_y)_k^\eta$ does not matter for PDHG [38, 156]. To further improve the speed and solution quality of PDHG, we incorporate a linesearch technique [156] that starts with the primal variable, followed by the dual update. The PDHG algorithm with linesearch is referred to as PDHGLS. Both PDHG and PDHGLS provide a saddle-point solution $(u_k^*, (p_x)_k^*, (p_y)_k^*)$ for (7.11) upon convergence [38, 156]. Since (7.10) is convex, u_k^* is indeed its solution, independent of the choice between using PDHG or PDHGLS. We summarize the proposed DCA-PDHGLS algorithm to solve (7.2) in Algorithm 4.

7.2.2 Convergence Analysis

We analyze the convergence of the sequence
$\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^\infty$ generated by (7.3) and (7.4), which are solved by (7.10) and (7.6), respectively. We establish in Lemma 7.1 that the sequence $\{\tilde{F}(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^\infty$ decreases sufficiently, followed by the convergence result in Theorem 7.1.

Lemma 7.1. *Suppose $\alpha \in [0, 1]$ and $\lambda > 0$. Let $\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^\infty$ be a sequence such that $\mathbf{u}^t$ is*

Algorithm 4: DCA-PDHGLS algorithm to solve (7.2)

1 **Input:**

- Image f
- model parameters $\alpha, \lambda > 0$
- strong convexity parameter $c > 0$
- PDHGLS initial step size $\tau_0 > 0$
- PDHGLS primal-dual step size ratio $\beta > 0$
- PDHGLS parameter $\delta \in (0, 1)$
- PDHGLS step size multiplier $\mu \in (0, 1)$

1: Set $u_k^0 = 1$ $(k = 1, \ldots, M)$ for some region $\Sigma \subset \Omega$ and 0 elsewhere.
2: Compute $\mathbf{c}^0 = (c_1^0, \ldots, c_N^0)$ by (7.6).
3: Set $t \coloneqq 0$.
4: **while** stopping criterion for DCA is not satisfied **do**
5:    **for** $k = 1$ to M **do**
6:       Set $u_k^{t,0} \coloneqq u_k^t$ and $(p_x)_k^0 = (p_y)_k^0 = 0$.
7:       Compute $((q_x)_k^t, (q_y)_k^t) = (D_x u_k^t, D_y u_k^t)/\sqrt{|D_x u_k^t|^2 + |D_y u_k^t|^2}$.
8:       Set $\theta_0 = 1$.
9:       Set $\eta \coloneqq 0$.
10:      **while** stopping criterion for PDHGLS is not satisfied **do**
11:          Compute $u_k^{t,\eta+1}$ by (7.13) with $\tau \coloneqq \tau_\eta$.
12:          Set $\tau_{\eta+1} = \tau_\eta \sqrt{1 + \theta_\eta}$.
             Linesearch:
13:          Compute $\theta_{\eta+1} = \frac{\tau_{\eta+1}}{\tau_\eta}$ and $\sigma_{\eta+1} = \beta \tau_{\eta+1}$.
14:          Compute $\bar{u}_k^{t,\eta+1} = u_k^{t,\eta+1} + \theta_{\eta+1}(u_k^{t,\eta+1} - u_k^{t,\eta})$.
15:          Compute $p_k^{\eta+1} \coloneqq ((p_x)_k^{\eta+1}, (p_y)_k^{\eta+1})$ by (7.14) with $\sigma \coloneqq \sigma_{\eta+1}$.
16:          **if** $\sqrt{\beta}\tau_{\eta+1}\|(D_x^\top (p_x)_k^{\eta+1}, D_y^\top (p_y)_k^{\eta+1}) - (D_x^\top (p_x)_k^\eta, D_y^\top (p_y)_k^\eta)\|_Y \leq \delta \|p_k^{\eta+1} - p_k^\eta\|_Y$ **then**
17:             Set $\eta \coloneqq \eta + 1$, and break linesearch
18:          **else**
19:             Set $\tau_{\eta+1} \coloneqq \mu \tau_{\eta+1}$ and go back to line 13.
20:          **end if**
             End of linesearch
21:       **end while**
22:       Set $u_k^{t+1} \coloneqq u_k^{t,\eta}$.
23:    **end for**
24:    Compute $\mathbf{c}^{t+1}$ by (7.6).
25:    Set $t \coloneqq t + 1$.
26: **end while**
Output: $(\mathbf{u}, \mathbf{c}) \coloneqq (\mathbf{u}^t, \mathbf{c}^t)$.

generated by (7.10) *and* $\mathbf{c}^t$ *is generated by* (7.6). *Then we have*

$$\tilde{F}(\mathbf{u}^t, \mathbf{c}^t) - \tilde{F}(\mathbf{u}^{t+1}, \mathbf{c}^{t+1}) \geq 2c \sum_{k=1}^{M} \|u_k^t - u_k^{t+1}\|_X^2.$$

Proof. Since $\mathbf{c}^{t+1}$ satisfies (7.6), we have

$$\tilde{F}(\mathbf{u}^{t+1}, \mathbf{c}^{t+1}) \leq \tilde{F}(\mathbf{u}^{t+1}, \mathbf{c}^{t}). \tag{7.15}$$

Then we estimate

$$\begin{aligned}
&\tilde{F}((\mathbf{u}^{t+1}_{\leq k-1}, \mathbf{u}^{t}_{\geq k}), \mathbf{c}^{t}) - \tilde{F}((\mathbf{u}^{t+1}_{\leq k}, \mathbf{u}^{t}_{\geq k+1}), \mathbf{c}^{t}) \\
=&\|Du^t_k\|_1 - \|Du^{t+1}_k\|_1 - \alpha(\|Du^t_k\|_{2,1} - \|Du^{t+1}_k\|_{2,1}) + \chi_U(u^t_k) - \chi_U(u^{t+1}_k) \\
&+ \lambda \sum_{\ell=1}^{N} \langle f_\ell(\mathbf{c}), R_\ell(\mathbf{u}^{t+1}_{\leq k-1}, \mathbf{u}^{t}_{\geq k}) - R_\ell(\mathbf{u}^{t+1}_{\leq k}, \mathbf{u}^{t}_{\geq k+1}) \rangle_X.
\end{aligned} \tag{7.16}$$

It follows from the first-order optimality condition of (7.10) at u^{t+1}_k that there exists $p^{t+1}_k \in \partial\left(\|Du^{t+1}_k\|_1 + \chi_U(u^{t+1}_k)\right)$ such that

$$0 = p^{t+1}_k - \alpha D^\top q^t_k + 2c(u^{t+1}_k - u^t_k) + \lambda r_k(\mathbf{c}^t, \mathbf{u}^{t+1}_{<k}, \mathbf{u}^{t}_{>k}).$$

Taking the inner product with $u^t_k - u^{t+1}_k$ and rearranging it, we obtain

$$\begin{aligned}
&\lambda \langle r_k(\mathbf{c}^t, \mathbf{u}^{t+1}_{<k}, \mathbf{u}^{t}_{>k}), u^t_k - u^{t+1}_k \rangle_X \\
=&-\langle p^{t+1}_k - \alpha D^\top q^t_k, u^t_k - u^{t+1}_k \rangle_X + 2c\|u^{t+1}_k - u^t_k\|^2_X.
\end{aligned} \tag{7.17}$$

The last term in (7.16) can be simplified to

$$\sum_{\ell=1}^{N} \langle f_\ell, R_\ell(\mathbf{u}^{t+1}_{\leq k-1}, \mathbf{u}^{t}_{\geq k}) - R_\ell(\mathbf{u}^{t+1}_{\leq k}, \mathbf{u}^{t}_{\geq k+1}) \rangle_X = \langle r_k(\mathbf{c}^t, \mathbf{u}^{t+1}_{<k}, \mathbf{u}^{t}_{>k}), u^t_k - u^{t+1}_k \rangle_X,$$

as $R_\ell(\mathbf{u})$ consists of terms with at most one u_k, and the terms without u^t_k and u^{t+1}_k are cancelled out. Together with (7.16) and (7.17), we get

$$\begin{aligned}
&\tilde{F}((\mathbf{u}^{t+1}_{\le k-1}, \mathbf{u}^{t}_{\ge k}), \mathbf{c}^t) - \tilde{F}((\mathbf{u}^{t+1}_{\le k}, \mathbf{u}^{t}_{\ge k+1}), \mathbf{c}^t) \\
=&\|Du_k^t\|_1 - \|Du_k^{t+1}\|_1 - \alpha(\|Du_k^t\|_{2,1} - \|Du_k^{t+1}\|_{2,1}) \\
&+ \chi_U(u_k^t) - \chi_U(u_k^{t+1}) + \lambda\langle r_k(\mathbf{c}^t, \mathbf{u}^{t+1}_{<k}, \mathbf{u}^{t}_{>k}), u_k^t - u_k^{t+1}\rangle_X \\
=&\|Du_k^t\|_1 - \|Du_k^{t+1}\|_1 - \alpha(\|Du_k^t\|_{2,1} - \|Du_k^{t+1}\|_{2,1}) \\
&+ \chi_U(u_k^t) - \chi_U(u_k^{t+1}) - \langle p_k^{t+1} - \alpha D^\top q_k^t, u_k^t - u_k^{t+1}\rangle_X + 2c\|u_k^{t+1} - u_k^t\|_X^2 \\
=& \left[\left(\|Du_k^t\|_1 - \langle p_k^{t+1}, u_k^t - u_k^{t+1}\rangle_X + \chi_U(u_k^t)\right) - \|Du_k^{t+1}\|_1 - \chi_U(u_k^{t+1})\right] \\
&+ \alpha(\|Du_k^{t+1}\|_{2,1} - \langle D^\top q_k^t, u_k^t - u_k^{t+1}\rangle_X - \|Du_k^t\|_{2,1}) + 2c\|u_k^{t+1} - u_k^t\|_X^2.
\end{aligned} \tag{7.18}$$

The definitions of convexity and subgradient yield that

$$\|Du_k^t\|_1 + \chi_U(u_k^t) - \langle p_k^{t+1}, u_k^t - u_k^{t+1}\rangle_X \ge \|Du_k^{t+1}\|_1 + \chi_U(u_k^{t+1}), \tag{7.19}$$

$$\|Du_k^{t+1}\|_{2,1} - \langle D^\top q_k^t, u_k^{t+1} - u_k^t\rangle_X \ge \|Du_k^t\|_{2,1}. \tag{7.20}$$

Combining (7.18)-(7.20), we have

$$\tilde{F}((\mathbf{u}^{t+1}_{\le k-1}, \mathbf{u}^{t}_{\ge k}), \mathbf{c}^t) - \tilde{F}((\mathbf{u}^{t+1}_{\le k}, \mathbf{u}^{t}_{\ge k+1}), \mathbf{c}^t) \ge 2c\|u_k^{t+1} - u_k^t\|_X^2.$$

Summing over $k = 1, \dots, M$ leads to

$$\begin{aligned}
\tilde{F}(\mathbf{u}^t, \mathbf{c}^t) - \tilde{F}(\mathbf{u}^{t+1}, \mathbf{c}^t) &= \sum_{k=1}^{M} \tilde{F}((\mathbf{u}^{t+1}_{\le k-1}, \mathbf{u}^{t}_{\ge k}), \mathbf{c}^t) - \tilde{F}((\mathbf{u}^{t+1}_{\le k}, \mathbf{u}^{t}_{\ge k+1}), \mathbf{c}^t) \\
&\ge 2c\sum_{k=1}^{M} \|u_k^{t+1} - u_k^t\|_X^2.
\end{aligned} \tag{7.21}$$

Therefore, (7.15) and (7.21) establish the desired result. □

Theorem 7.1. *Suppose $\alpha \in [0,1]$ and $\lambda > 0$. Let $\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^\infty$ be a sequence such that $\mathbf{u}^t$ is generated by (7.10) and $\mathbf{c}^t$ is generated by (7.6). We have the following:*

(a) $\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^{\infty}$ *is bounded.*

(b) For $k = 1, \ldots, M$*, we have* $\|u_k^{t+1} - u_k^t\|_X \to 0$ *as* $t \to \infty$*.*

(c) The sequence $\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^{\infty}$ *has a limit point* $(\mathbf{u}^*, \mathbf{c}^*)$ *satisfying*

$$\mathbf{0} \in \partial\|Du_k^*\|_1 - \alpha\partial\|Du_k^*\|_{2,1} + \partial\chi_U(u_k^*) + \lambda r_k(\mathbf{c}^*, \mathbf{u}_{<k}^*, \mathbf{u}_{>k}^*) \tag{7.22}$$

for $k = 1, \ldots, M$*, and*

$$0 \in \frac{\partial \tilde{F}(\mathbf{u}^*, \mathbf{c}^*)}{\partial c_\ell}, \quad \ell = 1, \ldots, N. \tag{7.23}$$

Proof. (a) As each entry of u_k^t is bounded by $[0, 1]$ for $k = 1, \ldots, M$, $\{\mathbf{u}^t\}_{t=1}^{\infty}$ is a bounded sequence. It further follows from (7.6) that $0 \leq |c_\ell^{t+1}| \leq \max_{i,j} |f_{i,j}|$. Therefore, $\{\mathbf{c}^t\}_{t=1}^{\infty}$ is also bounded, and altogether so is the sequence $\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^{\infty}$.

(b) Since $\alpha\|Du_k\|_{2,1} \leq \|Du_k\|_1$ for $\alpha \in [0, 1]$, we have

$$\tilde{F}(\mathbf{u}, \mathbf{c}) \geq \sum_{k=1}^{M} \chi_U(u_k) + \lambda \sum_{\ell=1}^{N} \langle f_\ell, R_\ell(\mathbf{u})\rangle_X \geq 0, \tag{7.24}$$

which implies that $\tilde{F}(\mathbf{u}, \mathbf{c})$ is lower bounded. As it is also decreasing by Lemma 7.1, the sequence $\{\tilde{F}(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^{\infty}$ converges. By a telescope summation of (7.21), we obtain

$$\tilde{F}(\mathbf{u}^1, \mathbf{c}^1) - \lim_{t\to\infty} \tilde{F}(\mathbf{u}^t, \mathbf{c}^t) \geq 2c \sum_{t=1}^{\infty}\sum_{k=1}^{M} \|u_k^t - u_k^{t+1}\|_X^2 = 2c \sum_{k=1}^{M}\sum_{t=1}^{\infty} \|u_k^t - u_k^{t+1}\|_X^2.$$

Therefore, $\sum_{t=1}^{\infty} \|u_k^t - u_k^{t+1}\|_X^2 < \infty$, leading to $\lim\limits_{t\to\infty} \|u_k^t - u_k^{t+1}\|_X^2 = 0$ for $k = 1, \ldots, M$.

(c) By Bolzano-Weierstrass Theorem, the bounded sequence $\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^{\infty}$ has a convergent subsequence $\{(\mathbf{u}^{t_L}, \mathbf{c}^{t_L})\}_{L=1}^{\infty}$ such that $\lim\limits_{L\to\infty} (\mathbf{u}^{t_L}, \mathbf{c}^{t_L}) = (\mathbf{u}^*, \mathbf{c}^*)$. By (b), $\lim\limits_{L\to\infty} u_k^{t_L+1} - u_k^{t_L} = 0$. As $\lim\limits_{L\to\infty} u_k^{t_L+1} = \lim\limits_{L\to\infty} u_k^{t_L} = u_k^*$, we have $\lim\limits_{L\to\infty} \mathbf{u}^{t_L+1} = \mathbf{u}^*$. Since $\mathbf{u}^{t_L}$ is generated by (7.10), all of its entries are bounded by $[0,1]$; otherwise, the objective function would be at $+\infty$. Hence, $\chi_U(u_k^{t_L}) = 0$ and similarly $\chi_U(u_k^{t_L+1}) = 0$ for all L, from which follows that $\chi_U(u_k^*) = 0$. In short, we have

$$\lim_{L\to\infty} \chi_U(u_k^{t_L}) = \chi_U(u_k^*) \quad \text{for } k = 1, \ldots, M. \tag{7.25}$$

Now we establish (7.23) by showing that $\tilde{F}(\mathbf{u}^*, \mathbf{c}^*) \leq \tilde{F}(\mathbf{u}^*, \mathbf{c})$ for all $\mathbf{c} \in \mathbb{R}^n$. On one hand, we have

$$\begin{aligned}
&\lim_{L\to\infty} \tilde{F}(\mathbf{u}^{t_L}, \mathbf{c}^{t_L}) \\
=& \lim_{L\to\infty} \left[\sum_{k=1}^{M} \left(\|Du_k^{t_L}\|_1 - \alpha \|Du_k^{t_L}\|_{2,1} + \chi_U(u_k^{t_L}) \right) + \lambda \sum_{\ell=1}^{N} \langle f_\ell(\mathbf{c}^{t_L}), R_\ell(\mathbf{u}^{t_L}) \rangle_X \right] \\
=& \sum_{k=1}^{M} \lim_{L\to\infty} \left(\|Du_k^{t_L}\|_1 - \alpha \|Du_k^{t_L}\|_{2,1} + \chi_U(u_k^{t_L}) \right) + \lambda \sum_{\ell=1}^{N} \lim_{L\to\infty} \langle f_\ell(\mathbf{c}^{t_L}), R_\ell(\mathbf{u}^{t_L}) \rangle_X \\
=& \sum_{k=1}^{M} \left(\|Du_k^*\|_1 - \alpha \|Du_k^*\|_{2,1} + \chi_U(u_k^*) \right) + \lambda \sum_{\ell=1}^{N} \langle f_\ell(\mathbf{c}^*), R_\ell(\mathbf{u}^*) \rangle_X = \tilde{F}(\mathbf{u}^*, \mathbf{c}^*).
\end{aligned} \tag{7.26}$$

We can take the limit as all the terms of $\tilde{F}$ except for χ_U are continuous with respect to $(\mathbf{u}, \mathbf{c})$. On the other hand, we have

$$\begin{aligned}
&\lim_{L\to\infty} \tilde{F}(\mathbf{u}^{t_L}, \mathbf{c}) \\
=& \lim_{L\to\infty} \left[\sum_{k=1}^{M} \left(\|Du_k^{t_L}\|_1 - \alpha \|Du_k^{t_L}\|_{2,1} + \chi_U(u_k^{t_L}) \right) + \lambda \sum_{\ell=1}^{N} \langle f_\ell(\mathbf{c}), R_\ell(\mathbf{u}^{t_L}) \rangle_X \right] \\
=& \sum_{k=1}^{M} \lim_{L\to\infty} \left(\|Du_k^{t_L}\|_1 - \alpha \|Du_k\|_{2,1} + \chi_U(u_k^{t_L}) \right) + \lambda \sum_{\ell=1}^{N} \lim_{L\to\infty} \langle f_\ell(\mathbf{c}), R_\ell(\mathbf{u}^{t_L}) \rangle_X \\
=& \sum_{k=1}^{M} \left(\|Du_k^*\|_1 - \alpha \|Du_k^*\|_{2,1} + \chi_U(u_k^*) \right) + \lambda \sum_{\ell=1}^{N} \langle f_\ell(\mathbf{c}), R_\ell(\mathbf{u}^*) \rangle_X = \tilde{F}(\mathbf{u}^*, \mathbf{c}).
\end{aligned} \tag{7.27}$$

It follows from (7.4) that for all $L \in \mathbb{N}$, we have

$$\tilde{F}(\mathbf{u}^{t_L}, \mathbf{c}^{t_L}) \leq \tilde{F}(\mathbf{u}^{t_L}, \mathbf{c}) \quad \forall\, \mathbf{c} \in \mathbb{R}^N. \tag{7.28}$$

Combined with (7.26)-(7.27),

$$\tilde{F}(\mathbf{u}^*, \mathbf{c}^*) = \lim_{L\to\infty} \tilde{F}(\mathbf{u}^{t_L}, \mathbf{c}^{t_L}) \leq \lim_{L\to\infty} \tilde{F}(\mathbf{u}^{t_L}, \mathbf{c}) = \tilde{F}(\mathbf{u}^*, \mathbf{c}) \quad \forall\, \mathbf{c} \in \mathbb{R}^N$$

or, equivalently $\tilde{F}(\mathbf{u}^*, \mathbf{c}^*) = \inf_{\mathbf{c}\in\mathbb{R}^N} \tilde{F}(\mathbf{u}^*, \mathbf{c})$. The minimization with respect to $\mathbf{c}$ can be expressed elementwise for each c_ℓ, leading to the optimality condition of (7.23).

For the rest of the proof, we establish (7.22). For each $k = 1, \ldots, M$, the optimality condition at the $(t_L + 1)$th step of (7.10) is

$$\begin{aligned} \mathbf{0} \in\, & \partial(\|Du_k^{t_L+1}\|_1 + \chi_U(u_k^{t_L+1})) + \lambda r_k(\mathbf{c}^{t_L}, \mathbf{u}_{<k}^{t_L+1}, \mathbf{u}_{>k}^{t_L}) + 2c(u_k^{t_L+1} - u_k^{t_L}) \\ & - \alpha D^\top q_k^{t_L}. \end{aligned} \tag{7.29}$$

Denote $s_k^L := -\lambda r_k(\mathbf{c}^{t_L}, \mathbf{u}_{<k}^{t_L+1}, \mathbf{u}_k^{t_L}) - 2c(u_k^{t_L+1} - u_k^{t_L}) + \alpha D^\top q_k^{t_L}$. Then (7.29) implies that

$$s_k^L \in \partial(\|Du_k^{t_L+1}\|_1 + \chi_U(u_k^{t_L+1})). \tag{7.30}$$

Since $r_k(\mathbf{c}, \mathbf{u}_{<k}, \mathbf{u}_{>k})$ is continuous in $(\mathbf{c}, \mathbf{u}_{<k}, \mathbf{u}_{>k})$, we have

$$\lim_{L\to\infty} r_k(\mathbf{c}^{t_L}, \mathbf{u}_{<k}^{t_L+1}, \mathbf{u}_{>k}^{t_L}) = r_k(\mathbf{c}^*, \mathbf{u}_{<k}^*, \mathbf{u}_{>k}^*).$$

To compute the limit of $D^\top q_k^{t_L}$, we recall the multivariate subgradient of

$\partial\|Du_k\|_{2,1} = \prod_{(i,j)} \partial\|(Du_k)_{i,j}\|_2$, where

$$\partial\|(x_1,x_2)\|_2 = \begin{cases} \left\{\frac{(x_1,x_2)}{\sqrt{x_1^2+x_2^2}}\right\} & \text{if } (x_1,x_2) \neq (0,0) \in \mathbb{R}^2, \\ \{(y_1,y_2) \in \mathbb{R}^2 : y_1^2 + y_2^2 \leq 1\} & \text{if } (x_1,x_2) = (0,0). \end{cases}$$

Let $((v^*_{x,k})_{i,j}, (v^*_{y,k})_{i,j}) \coloneqq ((D_x u^*_k)_{i,j}, (D_y u^*_k)_{i,j})$ be the discrete gradient of u^*_k at entry (i,j) for $k = 1, \ldots, M$, which satisfies

$$\partial\|(v^*_{x,k})_{i,j}, (v^*_{y,k})_{i,j}\|_2 =$$
$$\begin{cases} \left\{\frac{((v^*_{x,k})_{i,j},(v^*_{y,k})_{i,j})}{\sqrt{|(v^*_{x,k})_{i,j}|^2+|(v^*_{y,k})_{i,j}|^2}}\right\} & \text{if } ((v^*_{x,k})_{i,j}, (v^*_{y,k})_{i,j}) \neq (0,0), \\ \{(y_1,y_2) \in \mathbb{R}^2 : y_1^2 + y_2^2 \leq 1\} & \text{if } ((v^*_{x,k})_{i,j}, (v^*_{y,k})_{i,j}) = (0,0). \end{cases}$$

Note that we define $q_k^{t_L}$ in the following way

$$(q_k^{t_L})_{i,j} = \begin{cases} \frac{((D_x u_k^{t_L})_{i,j},(D_y u_k^{t_L})_{i,j})}{\sqrt{|(D_x u_k^{t_L})_{i,j}|^2+|(D_y u_k^{t_L})_{i,j}|^2}} & \text{if } ((D_x u_k^{t_L})_{i,j}, (D_y u_k^{t_L})_{i,j}) \neq (0,0), \\ (0,0) & \text{if } ((D_x u_k^{t_L})_{i,j}, (D_y u_k^{t_L})_{i,j}) = (0,0). \end{cases} \tag{7.31}$$

Denote $q^*_k \coloneqq \lim_{L\to\infty} q_k^{t_L}$. Therefore, by (7.31), when $((v^*_x)_{i,j}, (v^*_y)_{i,j}) \neq (0,0)$, we have

$$(q^*_k)_{i,j} = \lim_{L\to\infty} (q_k^{t_L})_{i,j} = \frac{((v^*_{x,k})_{i,j}, (v^*_{y,k})_{i,j})}{\sqrt{|(v^*_{x,k})_{i,j}|^2 + |(v^*_{y,k})_{i,j}|^2}} \in \partial\|((v^*_{x,k})_{i,j}, (v^*_{y,k})_{i,j})\|_2,$$

and when $((v^*_x)_{i,j}, (v^*_y)_{i,j}) = (0,0)$, we have

$$(q_k^{t_L})_{i,j} \in \{(y_1,y_2) \in \mathbb{R}^2 : y_1^2 + y_2^2 \leq 1\} \subseteq \partial\|((v^*_{x,k})_{i,j}, (v^*_{y,k})_{i,j})\|_2$$

for all $L \in \mathbb{N}$ so that taking the limit $L \to \infty$ yields $(q^*_k)_{i,j} \in \partial\|((v^*_{x,k})_{i,j}, (v^*_{y,k})_{i,j})\|_2$. By the chain rule of the subgradient (Corollary 16 in [85]), we have

$$\partial\|(Du_k^*)_{i,j}\|_2 = D^\top \partial\|((v_{x,k}^*)_{i,j}, (v_{y,k}^*)_{i,j})\|_2.$$

Since $D^\top$ is a linear operator (thus continuous), we get

$$\lim_{L\to\infty} D^\top q_k^{t_L} = D^\top q_k^* \in \partial\|Du_k^*\|_{2,1}. \tag{7.32}$$

In short, we obtain that $s_k^* \coloneqq \lim_{L\to\infty} s_k^L = -\lambda r_k(\mathbf{c}^*, \mathbf{u}_{<k}^*, \mathbf{u}_{>k}^*) + \alpha D^\top q_k^*$.

It further follows from (7.30) and the subgradient definition that

$$\begin{aligned} \|Du_k\|_1 + \chi_U(u_k) &\geq \|Du_k^{t_L+1}\|_1 + \chi_U(u_k^{t_L+1}) + \langle s_k^L, u_k - u_k^{t_L+1}\rangle \\ &= \|Du_k^{t_L+1}\|_1 + \langle s_k^L, u_k - u_k^{t_L+1}\rangle \end{aligned} \tag{7.33}$$

for all $u_k \in X$ and $L \in \mathbb{N}$. By continuity, we obtain

$$\begin{aligned} \|Du_k\|_1 + \chi_U(u_k) &\geq \lim_{L\to\infty} \left(\|Du_k^{t_L+1}\|_1 + \langle s_k^L, u_k - u_k^{t_L+1}\rangle\right) \\ &= \|Du_k^*\|_1 + \langle s_k^*, u_k - u_k^*\rangle = \|Du_k^*\|_1 + \chi_U(u_k^*) + \langle s_k^*, u_k - u_k^*\rangle, \end{aligned}$$

where the last equality is due to $\chi_U(u_k^*) = 0$. Since both $\|Du\|_1$ and $\chi_U(u)$ are convex, $s_k^* \in \partial(\|Du_k^*\|_1 + \chi_U(u_k^*)) = \partial\|Du_k^*\|_1 + \partial\chi_U(u_k^*)$. Therefore, we have

$$\begin{aligned} \mathbf{0} &\in \partial\|Du_k^*\|_1 + \partial\chi_U(u_k^*) + \lambda r_k(\mathbf{c}^*, \mathbf{u}_{<k}^*, \mathbf{u}_{>k}^*) - \alpha D^\top q_k^* \\ &\subseteq \partial\|Du_k^*\|_1 - \alpha\partial\|Du_k^*\|_{2,1} + \partial\chi_U(u_k^*) + \lambda r_k(\mathbf{c}^*, \mathbf{u}_{<k}^*, \mathbf{u}_{>k}^*). \end{aligned}$$

This concludes the proof. □

Remark 7.1. *The limit point* $(\mathbf{u}^*, \mathbf{c}^*)$ *is not guaranteed to be a global optimal solution for* (7.2) *because the objective function is nonconvex, and* $(\mathbf{u}^*, \mathbf{c}^*)$ *may not even satisfy a* *first-order optimality condition* $\mathbf{0} \in \partial_{(\mathbf{u},\mathbf{c})} \tilde{F}(\mathbf{u}^*, \mathbf{c}^*)$. *However, according to Theorem 7.1 (c), each coordinate* u_k^* *or* c_ℓ^* *satisfies its respective first-order optimality condition, since* $(\mathbf{u}^*, \mathbf{c}^*) = (u_1^*, \ldots, u_M^*, c_1^*, \ldots, c_N^*)$. *In convex optimization, if* g *is convex, a point* x^* *is a critical point if* $0 \in \partial g(x^*)$. (7.23) *establishes* c_ℓ^* *to be a critical point of the function convex in* c_ℓ,

$$\sum_{i=1}^{m} \sum_{j=1}^{n} (f_{i,j} - c_\ell)^2 R_\ell(\mathbf{u})_{i,j},$$

which is derived from (7.2) *when minimizing for* c_ℓ. *In DC optimization, a point* x^* *is a critical point of DC function* $g - h$ *if* $0 \in \partial g(u^*) - \partial h(u^*)$ *[115]. However, this optimality condition is not as strong as the optimality condition* $0 \in \partial(g-h)(u^*)$ *because* $\partial(g-h)(u^*) \subset \partial g(u^*) - \partial h(u^*)$ *in terms of either the Clarke subdifferential or the Fréchet subdifferential [115].* (7.22) *establishes* u_k^* *to be a DC critical point of the DC function*

$$\underbrace{\|Du_k\|_1 + \chi_U(u_k) + \lambda\langle r_k(\mathbf{c}, \mathbf{u}_{<k}, \mathbf{u}_{>k}), u_k\rangle_X}_{g(u_k)} - \underbrace{\alpha\|Du_k\|_{2,1}}_{h(u_k)},$$

which is derived from (7.2) *when minimizing for* u_k.

7.3 Fuzzy Extension of the AICV Model

One limitation of the CV models is that they are only applicable for image segmentation that has specifically power-of-two number (i.e., 2^M) of regions. To generalize to an arbitrary number of regions N, we associate each region Ω_ℓ with a membership function u_ℓ for $\ell = 1, \ldots, N$. A membership function u_ℓ represents a region Ω_ℓ in the following way:

$$(u_\ell)_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in \Omega_\ell, \\ 0 & \text{if } (i,j) \notin \Omega_\ell. \end{cases}$$

To avoid overlap between u_ℓ's, we enforce the constraint $\sum_{\ell=1}^N u_\ell = \mathbb{1}$, but we relax it with a quadratic penalty to make the model numerically tractable. As such, we propose an AITV extension to the FR model, referred to as AIFR,

$$\begin{aligned}\min_{\substack{\mathbf{u}\in X^N\\ \mathbf{c}\in\mathbb{R}^N}} \hat{F}(\mathbf{u},\mathbf{c}) := &\sum_{\ell=1}^N \left(\|Du_\ell\|_1 - \alpha\|Du_\ell\|_{2,1} + \chi_U(u_\ell)\right) + \lambda\sum_{\ell=1}^N \langle f_\ell(\mathbf{c}), u_\ell\rangle_X \\ &+ \frac{\nu}{2}\left\|\sum_{\ell=1}^N u_\ell - \mathbb{1}\right\|_X^2\end{aligned} \tag{7.34}$$

with $\nu > 0$. Similarly to (7.3)-(7.4), we adopt the alternating minimization framework to solve (7.34), i.e.,

$$\mathbf{u}^{t+1} \in \arg\min_{\mathbf{u}} \hat{F}(\mathbf{u},\mathbf{c}^t), \tag{7.35}$$

$$\mathbf{c}^{t+1} \in \arg\min_{\mathbf{c}} \hat{F}(\mathbf{u}^{t+1},\mathbf{c}). \tag{7.36}$$

The $\mathbf{c}$-subproblem (7.36) has a closed-form solution for $\ell = 1,\dots,N$,

$$c_\ell^{t+1} = \begin{cases} \dfrac{\sum_{i=1}^m\sum_{j=1}^n f_{i,j}(u_\ell^{t+1})_{i,j}}{\sum_{i=1}^m\sum_{j=1}^n (u_\ell^{t+1})_{i,j}} & \text{if } \sum_{i=1}^m\sum_{j=1}^n (u_\ell^{t+1})_{i,j} \neq 0, \\ 0 & \text{if } \sum_{i=1}^m\sum_{j=1}^n (u_\ell^{t+1})_{i,j} = 0. \end{cases} \tag{7.37}$$

For (7.35), we can find u_ℓ^{t+1} coordinatewise with respect to ℓ by solving

$$\begin{aligned}u_\ell^{t+1} \in \arg\min_{u_\ell} &\|Du_\ell\|_1 - \alpha\|Du_\ell\|_{2,1} + \chi_U(u_\ell) + \lambda\langle f_\ell(\mathbf{c}), u_\ell\rangle_X \\ &+ \frac{\nu}{2}\left\|\sum_{j<\ell} u_j^{t+1} + u_\ell + \sum_{j>\ell} u_\ell^t - \mathbb{1}\right\|_X^2.\end{aligned} \tag{7.38}$$

Applying DCA (7.8) to solve for (7.38) gives

$$\begin{aligned} u_\ell^{t+1} = \underset{u_\ell}{\arg\min} & \|Du_\ell\|_1 + \chi_U(u_\ell) + \lambda\langle f_\ell(\mathbf{c}), u_\ell\rangle_X \\ & + \frac{\nu}{2}\left\|\sum_{j<\ell} u_j^{t+1} + u_\ell + \sum_{j>\ell} u_\ell^t - \mathbb{1}\right\|_X^2 + c\|u_\ell\|_X^2 \\ & - \alpha\langle Du_\ell, q_\ell^t\rangle_Y - 2c\langle u_\ell, u_\ell^t\rangle_X, \end{aligned} \tag{7.39}$$

where $q_\ell^t := ((q_x)_\ell^t, (q_y)_\ell^t) = (D_x u_\ell^t, D_y u_\ell^t)/\sqrt{|D_x u_\ell^t|^2 + |D_y u_\ell^t|^2}$ if the denominator is not zero. Similarly to (7.10), we apply PDHGLS to find u_ℓ^{t+1} in (7.39) with the following iteration:

$$\begin{aligned} u_\ell^{t,\eta+1} &= \min\left\{\max\left\{\frac{2cu_\ell^t + \frac{1}{\tau}u_\ell^{t,\eta} + \nu\left(\mathbb{1} - \sum_{j<\ell} u_j^{t+1} - \sum_{j>\ell} u_\ell^t\right)}{2c + \frac{1}{\tau} + \nu}\right.\right. \\ &\qquad \left.\left. - \frac{\lambda f_\ell(\mathbf{c}) - \alpha D^\top q_\ell^t + (D_x^\top (p_x)_\ell^\eta + D_y^\top (p_y)_\ell^\eta)}{2c + \frac{1}{\tau} + \nu}, 0\right\}, 1\right\}, \end{aligned} \tag{7.40}$$

$$\bar{u}_\ell^{t,\eta+1} = u_\ell^{t,\eta+1} + \theta(u_\ell^{t,\eta+1} - u_\ell^{t,\eta}), \tag{7.41}$$

$$(p_x)_\ell^{\eta+1} = \mathrm{Proj}_P((p_x)_\ell^\eta + \sigma D_x \bar{u}_\ell^{t,\eta+1}), \tag{7.42}$$

$$(p_y)_\ell^{\eta+1} = \mathrm{Proj}_P((p_y)_\ell^\eta + \sigma D_y \bar{u}_\ell^{t,\eta+1}) \tag{7.43}$$

for $u_\ell^{t,0} = u_\ell^t$ and $\tau, \sigma > 0$, $\theta \in [0,1]$. The proposed algorithm is referred to as DCA-PDHGLS, summarized in Algorithm 5. Convergence analysis of the sequence $\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^\infty$ generated by (7.39) and (7.37) can be established similarly to the one in Section 2.1.5. Hence, we have the following theorem, but for the sake of brevity, the proof is omitted.

Theorem 7.2. *Suppose $\alpha \in [0,1]$ and $\lambda > 0$. Let $\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^\infty$ be a sequence such that $\mathbf{u}^t$ is generated by (7.39) and $\mathbf{c}^t$ is generated by (7.37). We have the following:*

(a) $\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^\infty$ is bounded.

(b) For $\ell = 1, \ldots, N$, we have $\|u_\ell^{t+1} - u_\ell^t\|_X \to 0$ as $t \to \infty$.

Algorithm 5: DCA-PDHGLS algorithm to solve (7.34)

1 **Input:**

- Image f
- model parameters $\alpha, \lambda > 0$
- strong convexity parameter $c > 0$
- quadratic penalty parameter $\nu > 0$
- PDHGLS initial step size $\tau_0 > 0$
- PDHGLS primal-dual step size ratio $\beta > 0$
- PDHGLS parameter $\delta \in (0, 1)$
- PDHGLS step size multiplier $\mu \in (0, 1)$

1: Set $u_\ell^0 = 1$ ($\ell = 1, \ldots, N$) for some region $\Sigma \subset \Omega$ and 0 elsewhere.
2: Compute $\mathbf{c}^0 = (c_1^0, \ldots, c_N^0)$ by (7.37).
3: Set $t \coloneqq 0$.
4: **while** stopping criterion for DCA is not satisfied **do**
5:   **for** $\ell = 1$ to M **do**
6:     Set $u_\ell^{t,0} \coloneqq u_\ell^t$ and $(p_x)_\ell^0 = (p_y)_\ell^0 = 0$.
7:     Compute $((q_x)_\ell^t, (q_y)_\ell^t) = (D_x u_\ell^t, D_y u_\ell^t)/\sqrt{|D_x u_\ell^t|^2 + |D_y u_\ell^t|^2}$.
8:     Set $\theta_0 = 1$.
9:     Set $\eta \coloneqq 0$.
10:     **while** stopping criterion for PDHGLS is not satisfied **do**
11:       Compute $u_\ell^{t,\eta+1}$ by (7.40) with $\tau \coloneqq \tau_\eta$.
12:       Set $\tau_{\eta+1} = \tau_\eta \sqrt{1 + \theta_\eta}$.
      Linesearch:
13:       Compute $\theta_{\eta+1} = \frac{\tau_{\eta+1}}{\tau_\eta}$ and $\sigma_{\eta+1} = \beta\tau_{\eta+1}$.
14:       Compute $\bar{u}_\ell^{t,\eta+1} = u_\ell^{t,\eta+1} + \theta_{\eta+1}(u_\ell^{t,\eta+1} - u_\ell^{t,\eta})$.
15:       Compute $p_\ell^{\eta+1} \coloneqq ((p_x)_\ell^{\eta+1}, (p_y)_\ell^{\eta+1})$ by (7.41)-(7.43) with $\sigma \coloneqq \sigma_{\eta+1}$.
16:       **if** $\sqrt{\beta}\tau_{\eta+1}\|(D_x^\top (p_x)_\ell^{\eta+1}, D_y^\top (p_y)_\ell^{\eta+1}) - (D_x^\top (p_x)_\ell^\eta, D_y^\top (p_y)_\ell^\eta)\|_Y \le \delta\|p_\ell^{\eta+1} - p_\ell^\eta\|_Y$ **then**
17:         Set $\eta \coloneqq \eta + 1$, and break linesearch
18:       **else**
19:         Set $\tau_{\eta+1} \coloneqq \mu\tau_{\eta+1}$ and go back to line 13.
20:       **end if**
      End of linesearch
21:     **end while**
22:     Set $u_\ell^{t+1} \coloneqq u_\ell^{t,\eta}$.
23:   **end for**
24:   Compute $\mathbf{c}^{t+1}$ by (7.37).
25:   Set $t \coloneqq t + 1$.
26: **end while**
Output: $(\mathbf{u}, \mathbf{c}) \coloneqq (\mathbf{u}^t, \mathbf{c}^t)$.

(c) The sequence $\{(\mathbf{u}^t, \mathbf{c}^t)\}_{t=1}^\infty$ *has a limit point* $(\mathbf{u}^*, \mathbf{c}^*)$ *satisfying*

$$\mathbf{0} \in \partial\|Du_\ell^*\|_1 - \alpha\partial\|Du_\ell^*\|_{2,1} + \partial\chi_U(u_\ell^*) + \lambda f_\ell(\mathbf{c}^*) + \nu\left(\sum_{j=1}^N u_j^* - \mathbb{1}\right), \tag{7.44}$$

$$0 \in \frac{\partial \hat{F}(\mathbf{u}^*, \mathbf{c}^*)}{\partial c_\ell} \quad \forall\, \ell = 1, \ldots, N. \tag{7.45}$$

7.4 Extension to Color Images

Both AICV (7.2) and AIFR (7.34) models can be extended to color image segmentation. Let $\mathbf{f} = (f_r, f_g, f_b) \in X^3$ be a color image and $(c_{\ell,r}, c_{\ell,g}, c_{\ell,b}) \in \mathbb{R}^3$ for $\ell = 1, \ldots, N$. By replacing $f_\ell(\mathbf{c})$ with

$$\mathbf{f}_\ell(\mathbf{c}_r, \mathbf{c}_g, \mathbf{c}_b) = \sum_{\iota \in \{r,g,b\}} (f_\iota - c_{\ell,\iota} \mathbb{1})^2,$$

where $\mathbf{c}_\iota = (c_{1,\iota}, \ldots, c_{N,\iota})$ for $\iota \in \{r, g, b\}$, the AICV model for color segmentation is

$$\min_{\substack{\mathbf{u} \in X^M \\ \mathbf{c}_r, \mathbf{c}_g, \mathbf{c}_b \in \mathbb{R}^N}} \sum_{k=1}^{M} (\|Du_k\|_1 - \alpha \|Du_k\|_{2,1} + \chi_U(u_k)) + \lambda \sum_{\ell=1}^{N} \langle \mathbf{f}_\ell(\mathbf{c}_r, \mathbf{c}_g, \mathbf{c}_b), R_\ell(\mathbf{u}) \rangle_X. \tag{7.46}$$

Similarly, the AIFR model for color segmentation is

$$\begin{aligned} \min_{\substack{\mathbf{u} \in X^N \\ \mathbf{c}_r, \mathbf{c}_g, \mathbf{c}_b \in \mathbb{R}^N}} & \sum_{\ell=1}^{N} (\|Du_\ell\|_1 - \alpha \|Du_\ell\|_{2,1} + \chi_U(u_\ell)) + \lambda \sum_{\ell=1}^{N} \langle \mathbf{f}_\ell(\mathbf{c}_r, \mathbf{c}_g, \mathbf{c}_b), u_\ell \rangle_X \\ & + \frac{\nu}{2} \left\| \sum_{\ell=1}^{N} u_\ell - \mathbb{1} \right\|_X^2 . \end{aligned} \tag{7.47}$$

For (7.46) and (7.47), their respective update formulas for $\mathbf{c}_\iota$ with $\iota \in \{r, g, b\}$ are

$$c_{\ell,\iota} = \begin{cases} \dfrac{\sum_{i=1}^{m} \sum_{j=1}^{n} (f_\iota)_{i,j} R_\ell(\mathbf{u})_{i,j}}{\sum_{i=1}^{m} \sum_{j=1}^{n} R_\ell(\mathbf{u})_{i,j}} & \text{if } \sum_{i=1}^{m} \sum_{j=1}^{n} R_\ell(\mathbf{u})_{i,j} \neq 0, \\ 0 & \text{if } \sum_{i=1}^{m} \sum_{j=1}^{n} R_\ell(\mathbf{u})_{i,j} = 0 \end{cases} \tag{7.48}$$

and

$$c_{\ell,\iota} = \begin{cases} \dfrac{\sum_{i=1}^{m}\sum_{j=1}^{n}(f_\iota)_{i,j}(u_\ell)_{i,j}}{\sum_{i=1}^{m}\sum_{j=1}^{n}(u_\ell)_{i,j}} & \text{if } \sum_{i=1}^{m}\sum_{j=1}^{n}(u_\ell)_{i,j} \neq 0, \\ 0 & \text{if } \sum_{i=1}^{m}\sum_{j=1}^{n}(u_\ell)_{i,j} = 0. \end{cases} \tag{7.49}$$

The update formulas for $\mathbf{u}$ are similar to their grayscale counterparts since only f_ℓ needs to be replaced with $\mathbf{f}_\ell$. Hence, their algorithms are straightforward to derive, thus omitted.

7.5 Numerical Results

In this section, we present extensive experiments on various synthetic and real images to demonstrate the effectiveness of AITV in image segmentation. In particular, we compare the AICV and AIFR models for $\alpha \in \{0, 0.25, 0.5, 0.75, 1.0\}$ with the two-stage segmentation methods that use $L_1 + L_2^2$ [28, 29], L_0 [193, 219], and real-time Mumford-Shah (R_{MS}) [195] penalties. When $\alpha = 0$, the AICV model reduces to the original CV (L_1 CV) model [43, 44], while the AIFR model becomes the fuzzy region competition (L_1 FR) model [119]. The two-stage segmentation methods find a smooth approximation $\bar{f}$ of the underlying image f with certain regularization, followed by k-means clustering on $\bar{f}$ to obtain the segmentation result. Specifically, Cai et al. [28, 29] proposed an $L_1 + L_2^2$ regularization problem[1]

$$\min_u \lambda \|f - u\|_X^2 + \gamma \|Du\|_Y^2 + \|Du\|_{2,1}. \tag{7.50}$$

[1]Code is available at https://xiaohaocai.netlify.app/download/.

Throughout our numerical experiments, we set $\gamma = 1$, which is suggested in [28, 29]. The L_0-regularized model [193, 219] is given by

$$\min_u \lambda\|f - u\|_X^2 + \|D_x u\|_0 + \|D_y u\|_0, \tag{7.51}$$

where $\|\cdot\|_0$ counts the number of nonzero entries of the matrix. The model in (7.51) can be solved in two different ways. One is by alternating minimization with half-quadratic splitting [219][2]. Another approach [193] incorporates weights for a better isotropic discretizatation than the original L_0 model, followed by ADMM[3]. The R_{MS} model [195] replaces the L_0 norm in (7.51) by $R_{MS}(u) = \sum_{i=1}^m \sum_{j=1}^n \min\{\gamma u_{i,j}, 1\}$, thus leading to

$$\min_u \lambda\|f - u\|_X^2 + R_{MS}(D_x u) + R_{MS}(D_y u). \tag{7.52}$$

In our numerical experiments, we consider the piecewise-constant limit case, where $\gamma \to \infty$. Its implementation is described in [195, Algorithm 1]. We refer to the models (7.50), (7.51), and (7.52) as $L_1 + L_2^2$, L_0, and R_{MS}, respectively.

For the proposed Algorithms 4 and 5, we set $c = 10^{-8}$, $\tau_0 = 1/8$, $\beta = 1.0$, $\delta = 0.9999$, and $\mu = 7.5 \times 10^{-5}$, as suggested in [141, 156]. The parameter λ depends on the image, which will be specified for each testing case. For the stopping criteria, we use the relative error

$$\text{relerr}(u, v) = \frac{\|u - v\|_X}{\max\{\|u\|_X, \|v\|_X, \epsilon\}}, \tag{7.53}$$

where ϵ is the machine's precision. Following [141], we choose the stopping criterion for the inner PDHGLS algorithm as $\text{relerr}(u^{t,\eta+1}, u^{t,\eta}) < 10^{-6}$. As for the outer iterations, DCA minimization terminates when $\text{relerr}(u^{t+1}, u^t) < 10^{-6}$ and $\text{relerr}(u^{t+1}, u^t) < 10^{-4}$ for 2-phase and 4-phase AICV models, respectively. For the AIFR models, we use the same stopping

[2]Code is available at `http://www.cse.cuhk.edu.hk/~leojia/projects/L0smoothing/`.
[3]Code is available at `https://github.com/mstorath/Pottslab`.

criterion in [120] for the outer iterations, i.e., when all the relative errors of the membership functions are less than 10^{-4}. We further adjust the maximum number of outer/inner iterations for multiple channels and multiphase segmentation, which are selected empirically for each image.

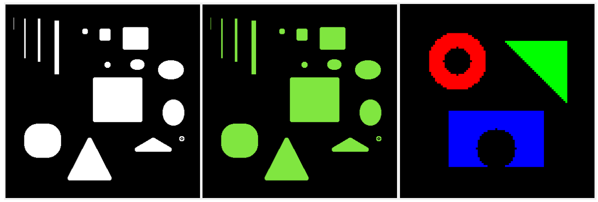
We shall apply postprocessing to define the segmented regions. In particular, we convert the results of Algorithm 4 to a binary output by setting any pixel values greater than or equal to 0.5 to 1, and 0 otherwise. For the results from Algorithm 5, we set a pixel value $(u_\ell)_{i,j}$ to 1 if it is the maximum among all the membership functions $\{u_k\}_{k=1}^N$ at pixel (i,j), and 0 otherwise. For a grayscale image f, we define its reconstructed image

$$\tilde{f} = \sum_{k=1}^{N} c_k \mathbb{1}_{\tilde{\Omega}_k}, \tag{7.54}$$

where $\{c_k\}_{k=1}^N$ and $\{\tilde{\Omega}_k\}_{k=1}^N$ are sets of constants and regions obtained by a segmentation algorithm, respectively, and $\mathbb{1}_{\tilde{\Omega}_k}$ is a binary image corresponding to the region $\tilde{\Omega}_k$. The matrix $\mathbb{1}_{\tilde{\Omega}_k}$ is obtained by thresholding for Algorithms (4) and (5) or by k-means clustering for the two-stage segmentation framework. Specifically for Algorithms 4 and 5, the constants $\{c_k\}_{k=1}^N$ are the final outputs of (7.6) and (7.37), respectively. For the two-stage segmentation framework, we compute a smoothed image of f by one of the models (7.50)-(7.52), thus getting $\bar{f}$, and define the constants in (7.54) by

$$c_k = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} \bar{f}_{i,j}(\mathbb{1}_{\tilde{\Omega}_k})_{i,j}}{\sum_{i=1}^{m}\sum_{j=1}^{n} (\mathbb{1}_{\tilde{\Omega}_k})_{i,j}}, \; k = 1, \ldots, N. \tag{7.55}$$

As k-means clustering applied to $\bar{f}$ does not produce an empty cluster, the denominator of



(a) (b) (c)

Figure 7.1: Synthetic images for image segmentation. (a) Grayscale image for two-phase segmentation. Size: 385×385. (b) Color image for two-phase segmentation. Size: 385×385. (c) Color image for four-phase segmentation. Size: 100×100.

(7.55) is nonzero. Similarly, the color image $\mathbf{f}$ is approximated by $\tilde{\mathbf{f}} = (\tilde{f}_r, \tilde{f}_g, \tilde{f}_b)$ given by

$$\tilde{f}_\iota = \sum_{k=1}^{N} c_{k,\iota} \mathbb{1}_{\tilde{\Omega}_k} \text{ for } \iota \in \{r, g, b\}, \tag{7.56}$$

where $\{c_{k,\iota}\}_{k=1}^N$ is a set of constants for channel ι. For the color versions of Algorithms 4 and 5, the constants are obtained by (7.48) and (7.49), respectively. For the color version of the two-stage segmentation framework, the constants are computed by (7.55) applied to each channel of the smoothed image $\bar{\mathbf{f}} = (\bar{f}_r, \bar{f}_g, \bar{f}_b)$.

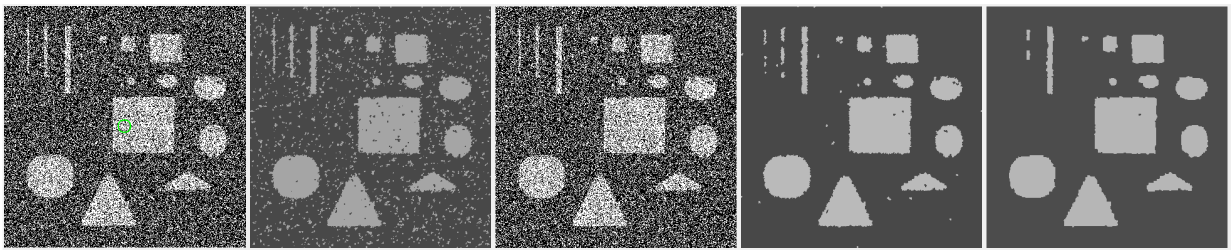
All the algorithms are coded in MATLAB R2019a and all the computations are performed on a Dell laptop with a 1.80 GHz Intel Core i7-8565U processor and 16.0 GB of RAM. The codes are available at `https://github.com/kbui1993/L1mL2Segmentation`.

7.5.1 Synthetic Images

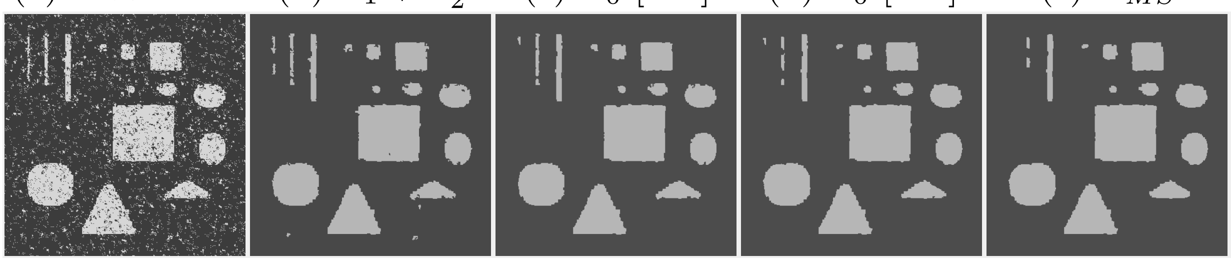
We apply various segmentation algorithms on the synthetic images presented in Figure 7.1. We scale the intensity values of all the images to be $[0, 1]$ to ease the parameter tuning. To demonstrate the robustness of the algorithms with respect to noises, we contaminate the original images with either salt-and-pepper impulsive noise (SPIN) or random-valued impulsive noise (RVIN). To evaluate the model performance, we compute the DICE index

Table 7.1: DICE indices of various segmentation models applied to Figure 7.1a corrupted with different levels of impulsive noise.

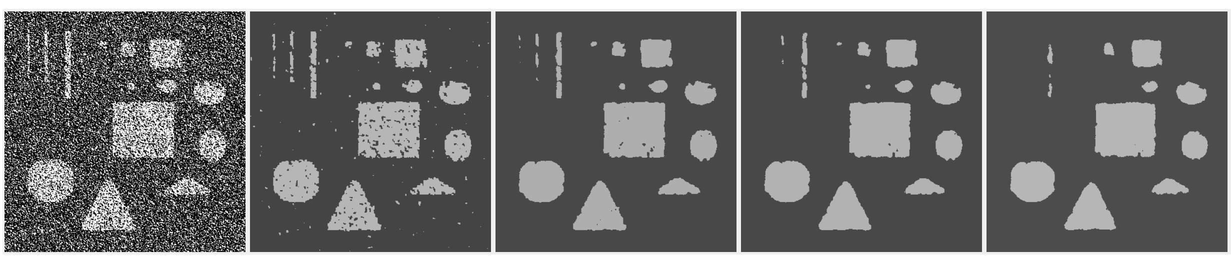
Salt & Pepper (%)	0	10	20	30	40	50	60	70
$L_1 - L_2$ CV	**1**	0.9977	0.9932	0.9854	0.9594	0.9062	0.8138	0.7643
$L_1 - 0.75L_2$CV	**1**	0.9978	0.9929	0.9853	0.9795	0.9727	0.9678	0.9550
$L_1 - 0.5L_2$ CV	**1**	0.9975	0.9941	0.9893	0.9850	0.9801	**0.9726**	**0.9554**
$L_1 - 0.25L_2$ CV	**1**	0.9974	0.9954	0.9910	0.9870	**0.9823**	0.9711	0.9483
L_1 CV	**1**	**0.9981**	0.9960	0.9922	0.9877	0.9802	0.9681	0.9338
$L_1 - L_2$ FR	**1**	0.8753	0.7719	0.6833	0.6129	0.5425	0.4702	0.4138
$L_1 - 0.75L_2$ FR	**1**	0.9896	0.9841	0.9693	0.9585	0.9437	0.9183	0.7775
$L_1 - 0.5L_2$ FR	0.9998	0.9978	0.9956	0.9923	**0.9879**	0.9788	0.9495	0.7760
$L_1 - 0.25L_2$ FR	0.9995	0.9979	**0.9961**	**0.9925**	0.9865	0.9737	0.9347	0.6883
L_1 FR	0.9992	0.9978	0.9949	0.9877	0.9812	0.9663	0.8990	0.5053
$L_1 + L_2^2$	0.9996	0.9961	0.9925	0.9857	0.9733	0.9328	0.8375	0.6840
L_0 [219]	**1**	0.8731	0.7666	0.6736	0.5943	0.5226	0.4601	0.4035
L_0 [193]	0.9995	0.9944	0.9874	0.9792	0.9738	0.9690	0.9605	0.9474
R_{MS}	0.9995	0.9969	0.9947	0.9887	0.9851	0.9784	0.9670	0.9312
Random-valued (%)	**0**	**10**	**20**	**30**	**40**	**50**	**60**	**70**
$L_1 - L_2$ CV	**1**	0.9986	0.9957	0.9909	0.9846	0.9739	0.9534	0.9542
$L_1 - 0.75L_2$ CV	**1**	0.9988	0.9971	0.9948	0.9926	0.9894	**0.9840**	**0.9712**
$L_1 - 0.5L_2$ CV	**1**	0.9989	**0.9973**	0.9958	0.9930	**0.9899**	0.9816	0.9614
$L_1 - 0.25L_2$ CV	**1**	**0.9990**	0.9971	0.9957	**0.9935**	0.9898	0.9808	0.9560
L_1 CV	**1**	0.9984	0.9972	**0.9959**	0.9928	0.9863	0.9700	0.9332
$L_1 - L_2$ FR	**1**	0.9505	0.9053	0.8578	0.8015	0.7369	0.6478	0.5662
$L_1 - 0.75L_2$ FR	**1**	0.9987	0.9971	0.9945	0.9913	0.9879	0.9715	0.5364
$L_1 - 0.5L_2$ FR	0.9998	0.9984	0.9972	0.9955	0.9921	0.9833	0.9538	0.3540
$L_1 - 0.25L_2$ FR	0.9995	0.9983	0.9972	0.9940	0.9880	0.9763	0.9299	0.5984
L_1 FR	0.9992	0.9983	0.9970	0.9925	0.9833	0.9643	0.8800	0.4503
$L_1 + L_2^2$	0.9996	0.9980	0.9960	0.9937	0.9903	0.9858	0.9776	0.9668
L_0 [219]	**1**	0.8753	0.7697	0.6768	0.5981	0.5247	0.4627	0.4054
L_0 [193]	0.9995	0.9966	0.9933	0.9904	0.9874	0.9810	0.9688	0.9462
R_{MS}	0.9995	0.9983	0.9971	0.9954	0.9932	0.9850	0.9731	0.9361



(a) 60% SPIN (b) $L_1 + L_2^2$ (c) L_0 [219] (d) L_0 [193] (e) R_{MS}

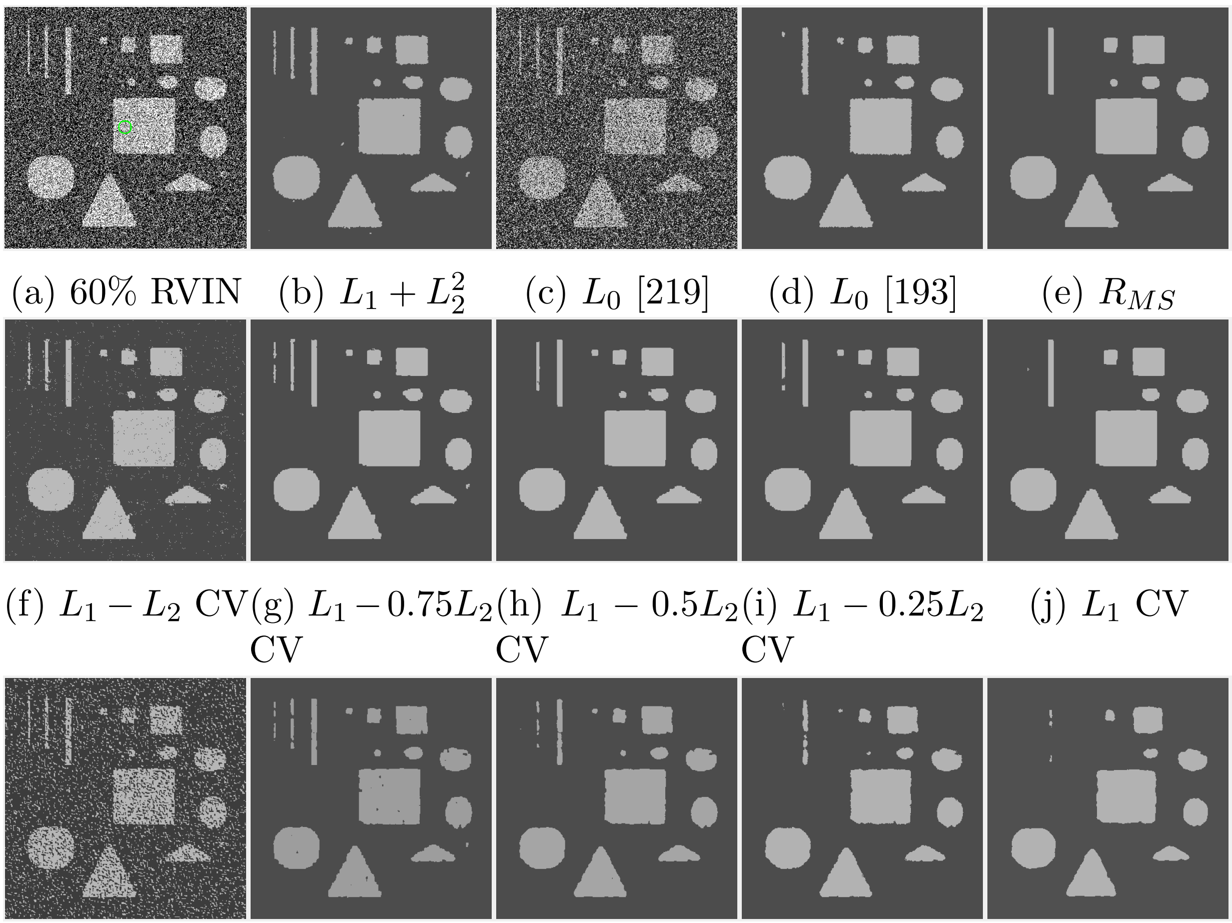


(f) $L_1 - L_2$ CV (g) $L_1 - 0.75L_2$ CV (h) $L_1 - 0.5L_2$ CV (i) $L_1 - 0.25L_2$ CV (j) L_1 CV



(k) $L_1 - L_2$ FR (l) $L_1 - 0.75L_2$ FR (m) $L_1 - 0.5L_2$ FR (n) $L_1 - 0.25L_2$ FR (o) L_1 FR

Figure 7.2: Reconstruction results on Figure 7.1a corrupted with 60% SPIN.



(a) 60% RVIN (b) $L_1 + L_2^2$ (c) L_0 [219] (d) L_0 [193] (e) R_{MS}

(f) $L_1 - L_2$ CV (g) $L_1 - 0.75L_2$ CV (h) $L_1 - 0.5L_2$ CV (i) $L_1 - 0.25L_2$ CV (j) L_1 CV

(k) $L_1 - L_2$ FR (l) $L_1 - 0.75L_2$ FR (m) $L_1 - 0.5L_2$ FR (n) $L_1 - 0.25L_2$ FR (o) L_1 FR

Figure 7.3: Reconstruction results on Figure 7.1a corrupted with 60% RVIN.

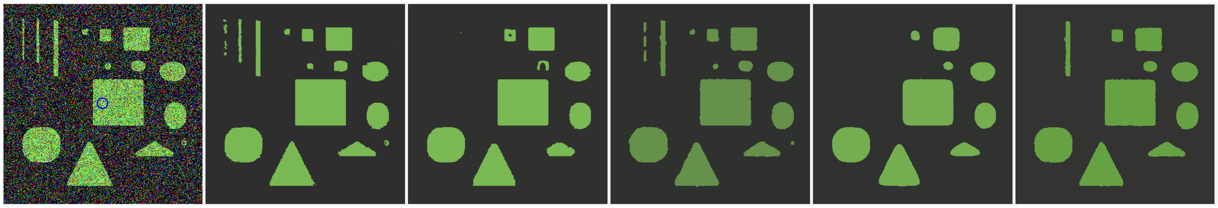
[60] between the segmentation result and the ground truth. The metric is defined by

$$\text{DICE} = 2\frac{\#\{A(i) \cap A'(i)\}}{\#\{A(i)\} + \#\{A'(i)\}},$$

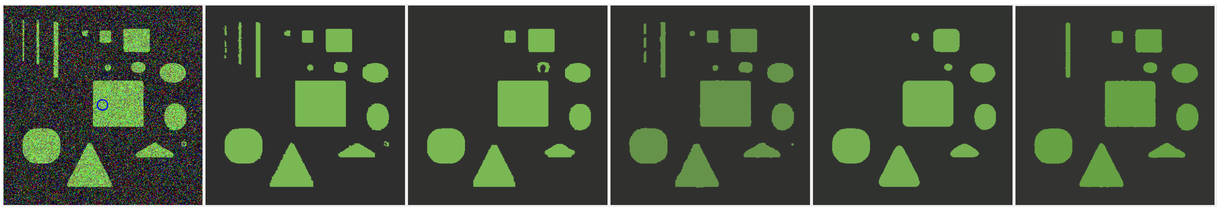
where $A(i)$ is the set of pixels with label i in the ground-truth image f or $\mathbf{f}$, $A'(i)$ is the set of pixels with label i in the segmented image $\tilde{f}$ or $\tilde{\mathbf{f}}$, and $\#\{A\}$ refers to the number of pixels in the set A. If the DICE index equals 1, it means the perfect alignment of the segmentation result to the ground truth. For two-phase segmentation, we compute the DICE index only for the object of interest, not the background. For multiphase segmentation, we compute the mean of the DICE indices across the regions, including the background.

For the two-phase AICV model, the initialization u_1^0 in Algorithm 4 is a binary step function that represents a circle of radius 10 in the center of the image (i.e., taking the value 1 if inside the circle and 0 elsewhere). Since the binary step function forms two regions in an image, it can be used as initialization for the two-phase AIFR model, i.e., u_1^0 and $u_2^0 = \mathbb{1} - u_1^0$ for Algorithm 5. The initialization for the four-phase segmentation requires two step functions, which are set to be two circles of radius 30 shifted by 5 pixels to the right of the image center and another by 5 pixels to the left. The circle functions are used here for simplicity. Contours of the initialization are marked as colored circles in the noisy images.

For Figure 7.1a, we set $\lambda = 2$ for all methods, except for L_0 [219] in which $\lambda = 50$. For the AIFR models, we set $\nu = 10$. The maximum number of inner iterations for the AITV models is 300, while the maximum number of outer iterations is 20 for AICV and 40 for AIFR. Table 7.1 records the DICE indices of the segmentation results for varying levels of both SPIN and RVIN from 0% to 70%. When the noise level is at least 50%, both $L_1 - 0.5L_2$ and $L_1 - 0.25L_2$ CV models outperform L_1 CV. For AIFR, $L_1 - 0.5L_2$ and $L_1 - 0.25L_2$ outperform L_1 across all levels of SPIN corruption. In addition, $L_1 - L_2$ FR is less robust than other values of α when the noise level increases. Most of the best results in the cases of



(a) 40% SPIN (b) $L_1 - L_2$ CV (c) L_1 CV (d) $L_1 - 0.75L_2$ FR (e) L_1 FR (f) $L_1 + L_2^2$



(g) 40% RVIN (h) $L_1 - L_2$ CV (i) L_1 CV (j) $L_1 - 0.75L_2$ FR (k) L_1 FR (l) $L_1 + L_2^2$

Figure 7.4: Reconstruction results on Figure 7.1b corrupted with 40% SPIN (top) and 40% RVIN (bottom).

intermediate to high RVIN noise levels are attained by the proposed models. Figures 7.2-7.3 display the segmentation results of Figure 7.1a corrupted with 60% SPIN and 60% RVIN, respectively. (We note that the contrast of the reconstructed images is different from Figure 7.1a because the impulsive noise in the corrupted image skews the values of the constants $\{c_k\}_{k=1}^N$ computed by the segmentation algorithms. This phenomenon repeats for Figures 7.1b-7.1c.) As α decreases in both the AICV and AIFR models, the results become less noisy, but they have less segmented regions. Therefore, $\alpha = 0.5$ yields the best compromise in the case of SPIN. For RVIN, the AICV and AIFR results are not as noisy as in the case of SPIN, and hence $\alpha = 0.75$ is the best for RVIN. The two-stage methods generally produce noisy results in the presence of SPIN and RVIN.

Figure 7.1b is a color version of Figure 7.1a. We corrupt the image by 0% to 50% SPIN/RVIN for each color channel. When a color image is corrupted with noise, one channel might be noisier than the others. In addition, image structures may vary with color channels, thus making the color extension of finding a balanced segmentation across all the color channels more challenging than for grayscale images. For Figure 7.1b, we set $\lambda = 0.5$ for all methods,

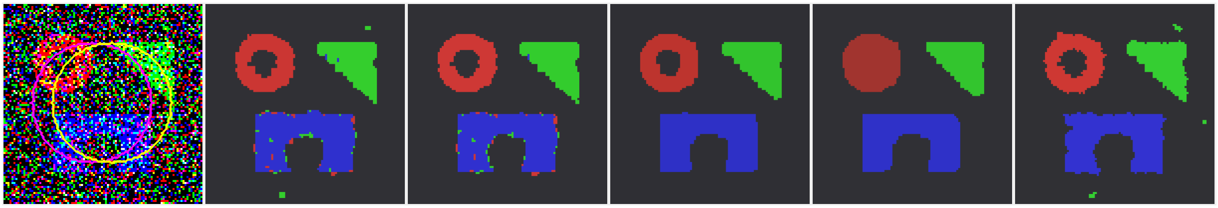
Table 7.2: DICE indices of various segmentation models applied to Figure 7.1b corrupted with different levels of impulsive noise.

Salt & Pepper (%)	0	10	20	30	40	50
$L_1 - L_2$ CV	**1**	**0.9979**	0.9952	**0.9920**	**0.9867**	**0.9775**
$L_1 - 0.75L_2$ CV	0.9994	0.9978	**0.9957**	0.9896	0.9856	0.9737
$L_1 - 0.5L_2$ CV	0.9992	0.9970	0.9910	0.9889	0.9826	0.9512
$L_1 - 0.25L_2$ CV	0.9982	0.9924	0.9904	0.9829	0.9726	0.9308
L_1 CV	0.9938	0.9918	0.9808	0.9755	0.9457	0.9109
$L_1 - L_2$ FR	0.9977	0.9960	0.9931	0.9685	0.8187	0.7273
$L_1 - 0.75L_2$ FR	0.9979	0.9955	0.9920	0.9873	0.9795	0.9626
$L_1 - 0.5L_2$ FR	0.993	0.9908	0.9802	0.9720	0.9635	0.9409
$L_1 - 0.25L_2$ FR	0.9818	0.9786	0.9690	0.9462	0.9441	0.9195
L_1 FR	0.9774	0.9705	0.9524	0.9383	0.9301	0.8906
$L_1 + L_2^2$	0.9931	0.9907	0.9874	0.9794	0.9726	0.9686
L_0 [219]	**1**	0.8734	0.7687	0.6745	0.5945	0.4307
L_0 [193]	0.9939	0.9904	0.9823	0.9762	0.9543	0.9266
R_{MS}	0.9853	0.9801	0.9676	0.9444	0.9116	0.8225
Random-valued (%)	0	10	20	30	40	50
$L_1 - L_2$ CV	**1**	**0.9987**	**0.9966**	**0.9932**	**0.9887**	**0.9826**
$L_1 - 0.75L_2$ CV	0.9994	0.9983	0.9960	0.9915	0.9877	0.9759
$L_1 - 0.5L_2$ CV	0.9992	0.9975	0.9916	0.9899	0.9815	0.9535
$L_1 - 0.25L_2$ CV	0.9982	0.9928	0.9913	0.9784	0.9748	0.9344
L_1 CV	0.9938	0.9920	0.9798	0.9773	0.9493	0.9145
$L_1 - L_2$ FR	0.9977	0.9965	0.9943	0.9902	0.9071	0.7154
$L_1 - 0.75L_2$ FR	0.9979	0.9960	0.9921	0.9879	0.9815	0.9520
$L_1 - 0.5L_2$ FR	0.993	0.9907	0.9797	0.9742	0.9644	0.9526
$L_1 - 0.25L_2$ FR	0.9818	0.9781	0.9702	0.9620	0.9534	0.9161
L_1 FR	0.9774	0.9656	0.9533	0.9519	0.9316	0.8770
$L_1 + L_2^2$	0.9931	0.9912	0.9877	0.9812	0.9755	0.9726
L_0 [219]	**1**	0.9032	0.7991	0.6972	0.6089	0.5312
L_0 [193]	0.9939	0.9852	0.9846	0.9786	0.9573	0.9298
R_{MS}	0.9853	0.9797	0.9782	0.9465	0.9074	0.8260

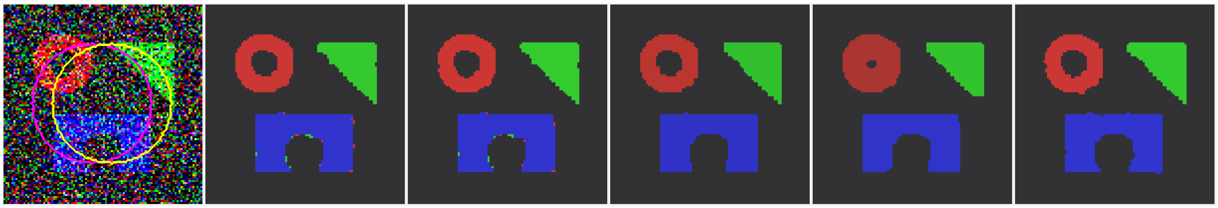
except for L_0 [219] in which $\lambda = 50$. For the AIFR models, we set $\nu = 2.5$. The maximum number of inner/outer iterations are the same as the case for Figure 7.1a. The DICE indices of the segmentation results are reported in Table 7.2, which shows that $L_1 - L_2$ CV generally yields the best results and AIFR is slightly worse than its AICV counterpart but better than L_1 FR. Figure 7.4 presents the comparison results of AICV (with optimal α), L_1 CV, AIFR

Table 7.3: DICE indices of various segmentation models applied to Figure 7.1c corrupted with different levels of impulsive noise.

Salt & Pepper (%)	0	10	20	30	40
$L_1 - L_2$ CV	0.9990	0.9762	0.9524	0.9245	0.8548
$L_1 - 0.75L_2$ CV	0.9992	0.9763	0.9649	0.9288	0.8978
$L_1 - 0.5L_2$ CV	0.9992	0.9789	0.9704	0.9509	0.9292
$L_1 - 0.25L_2$ CV	0.9994	0.9852	0.9686	0.9608	0.9448
L_1 CV	0.9987	0.9832	0.9788	0.9597	0.9496
$L_1 - L_2$ FR	0.9994	0.7869	0.6566	0.5424	0.4552
$L_1 - 0.75L_2$ FR	0.9994	0.9328	0.8736	0.8058	0.6541
$L_1 - 0.5L_2$ FR	0.9980	0.9905	0.9847	0.9720	0.8976
$L_1 - 0.25L_2$ FR	0.9976	0.9921	0.9863	0.9801	**0.9753**
L_1 FR	0.9976	**0.9924**	**0.9869**	**0.9804**	0.9474
$L_1 + L_2^2$	0.9984	0.9904	0.9691	0.8984	0.7562
L_0 [219]	**1**	0.7611	0.6284	0.5134	0.4225
L_0 [193]	0.9997	0.9245	0.7977	0.6536	0.4884
R_{MS}	**1**	0.9900	0.9771	0.9649	0.9575
Random-valued (%)	0	10	20	30	40
$L_1 - L_2$ CV	0.9990	0.9895	0.9757	0.9594	0.9261
$L_1 - 0.75L_2$ CV	0.9992	0.9910	0.9831	0.9755	0.9664
$L_1 - 0.5L_2$ CV	0.9992	0.9934	0.9875	0.9797	0.9737
$L_1 - 0.25L_2$ CV	0.9994	0.9934	0.9876	0.9798	0.9771
L_1 CV	0.9987	0.9941	0.9884	0.9789	0.9761
$L_1 - L_2$ FR	0.9994	0.8841	0.7118	0.6604	0.5972
$L_1 - 0.75L_2$ FR	0.9994	0.9916	0.9875	0.9353	0.8790
$L_1 - 0.5L_2$ FR	0.998	0.9947	**0.9912**	**0.9851**	**0.9833**
$L_1 - 0.25L_2$ FR	0.9976	0.9942	**0.9912**	0.9849	0.9821
L_1 FR	0.9976	0.9921	0.9892	**0.9851**	0.9553
$L_1 + L_2^2$	0.9984	0.9949	0.9857	0.9803	0.9705
L_0 [219]	**1**	0.7744	0.6932	0.5302	0.4478
L_0 [193]	0.9997	0.9828	0.9614	0.9482	0.9311
R_{MS}	**1**	**0.9953**	0.9900	0.9849	0.9831



(a) 40% SPIN (b) $L_1 - 0.25L_2$ CV (c) L_1 CV (d) $L_1 - 0.25L_2$ FR (e) L_1 FR (f) R_{MS}



(g) 40% RVIN (h) $L_1 - 0.25L_2$ CV (i) L_1 CV (j) $L_1 - 0.5L_2$ FR (k) L_1 FR (l) R_{MS}

Figure 7.5: Reconstruction results on Figure 7.1c corrupted with 40% SPIN (top) and 40% RVIN (bottom).

(with optimal α), L_1 FR, and $L_1 + L_2^2$ for 40% SPIN and 40% RVIN, showing that AICV and AIFR segment more salient regions than their L_1 counterparts and $L_1 + L_1^2$.

Figure 7.1c is a color image for multiphase segmentation. We set $\lambda = 2.25$ for all methods, except for L_0 [219] in which $\lambda = 50$. For the AIFR models, we set $\nu = 5$. The maximum number of inner iterations for the AITV models is 1000, while the maximum number of outer iterations is 40 for AICV and 160 for AIFR. Table 7.3 presents the DICE indices of the segmentation results under 0% to 40% SPIN/RVIN contamination for each color channel. For SPIN, $L_1 - 0.25L_2$ FR is comparable to L_1 FR and outperforms it when the noise level is 40% . For RVIN, $L_1 - 0.5L_2$ and $L_1 - 0.25L_2$ FR give the best results in general. We also observe that the smaller α is, the more robust AICV/AIFR are with respect to impulsive noise. The visual results are presented in Figure 7.5 for 40% SPIN/RVIN, clearly showing that AIFR provides the best segmentation. AICV and L_1 CV contain noise along the edges of the blue region, L_1 FR oversegments the red region, and R_{MS} appears slightly worse than AIFR.

Overall, the proposed AICV/AIFR methods are robust against impulsive noise, unlike the

(a) (b) (c) (d) (e)

Figure 7.6: Real images for image segmentation. (a) Close-up of a target board in a video. Size: 89×121. (b) Image of a hawk. Size: 318×370. (c) Image of a butterfly. Size: 321×481. (d) Image of a flower. Size: 321×481. (e) Image of peppers. Size: 481×321.

Table 7.4: PSNR values of segmentation methods applied to real color images. NA stands for "not applicable."

	Figure 7.6b	Figure 7.6c	Figure 7.6d	Figure 7.6e
$L_1 - L_2$ CV	23.3949	21.9000	NA	NA
$L_1 - 0.75L_2$ CV	23.3933	21.9001	NA	NA
$L_1 - 0.5L_2$ CV	**23.4001**	21.8976	NA	NA
$L_1 - 0.25L_2$ CV	23.3913	21.8985	NA	NA
L_1 CV	23.3690	21.8977	NA	NA
$L_1 - L_2$ FR	23.4223	22.2574	21.8283	22.2597
$L_1 - 0.75L_2$ FR	23.4014	**22.2578**	21.8383	22.4880
$L_1 - 0.5L_2$ FR	23.3814	22.2576	**21.8418**	**22.4901**
$L_1 - 0.25L_2$ FR	23.3523	22.2575	**21.8418**	22.4672
L_1 FR	23.3173	22.2570	21.8409	21.9482
$L_1 + L_2^2$	23.2601	21.6077	21.1802	21.0277
L_0 [219]	23.2419	22.2570	21.7914	22.0361
L_0 [193]	23.1985	17.7573	21.8129	21.9703
R_{MS}	23.0865	17.7140	21.7832	22.0904

two-stage methods. For the three synthetic images, AICV and AIFR with appropriately chosen α outperform their L_1 counterparts under a high level of impulsive noise. Unfortunately, there is no optimal choice of α that works for all images, as demonstrated by our experiments. For example, $\alpha = 1.0$ yields the highest DICE indices for Figure 7.1b according to Table 7.2, but it does not perform as well for Figure 7.1a according to Table 7.1.

Table 7.5: Computational time (seconds) of segmentation methods applied to real color images. NA stands for "not applicable."

	Figure 7.6a	Figure 7.6b	Figure 7.6c	Figure 7.6d	Figure 7.6e
$L_1 - L_2$ CV	2.06	16.09	49.27	NA	NA
$L_1 - 0.75L_2$ CV	1.86	15.91	55.91	NA	NA
$L_1 - 0.5L_2$ CV	2.08	15.89	70.68	NA	NA
$L_1 - 0.25L_2$ CV	2.17	16.09	71.23	NA	NA
L_1 CV	1.78	16.23	54.94	NA	NA
$L_1 - L_2$ FR	2.51	43.65	66.27	191.30	212.28
$L_1 - 0.75L_2$ FR	1.91	46.26	64.98	185.26	233.79
$L_1 - 0.5L_2$ FR	1.23	15.29	68.3	175.67	263.52
$L_1 - 0.25L_2$ FR	0.92	13.18	69.49	182.08	227.62
L_1 FR	0.72	13.18	69.49	182.08	227.62
$L_1 + L_2^2$	0.24	1.8	**1.2**	1.75	2.48
L_0 [219]	**0.15**	**0.92**	1.71	**1.6**	**1.97**
L_0 [193]	0.17	2.96	3.06	3.05	4.26
R_{MS}	0.61	6.60	17.71	17.24	20.10

7.5.2 Real Images

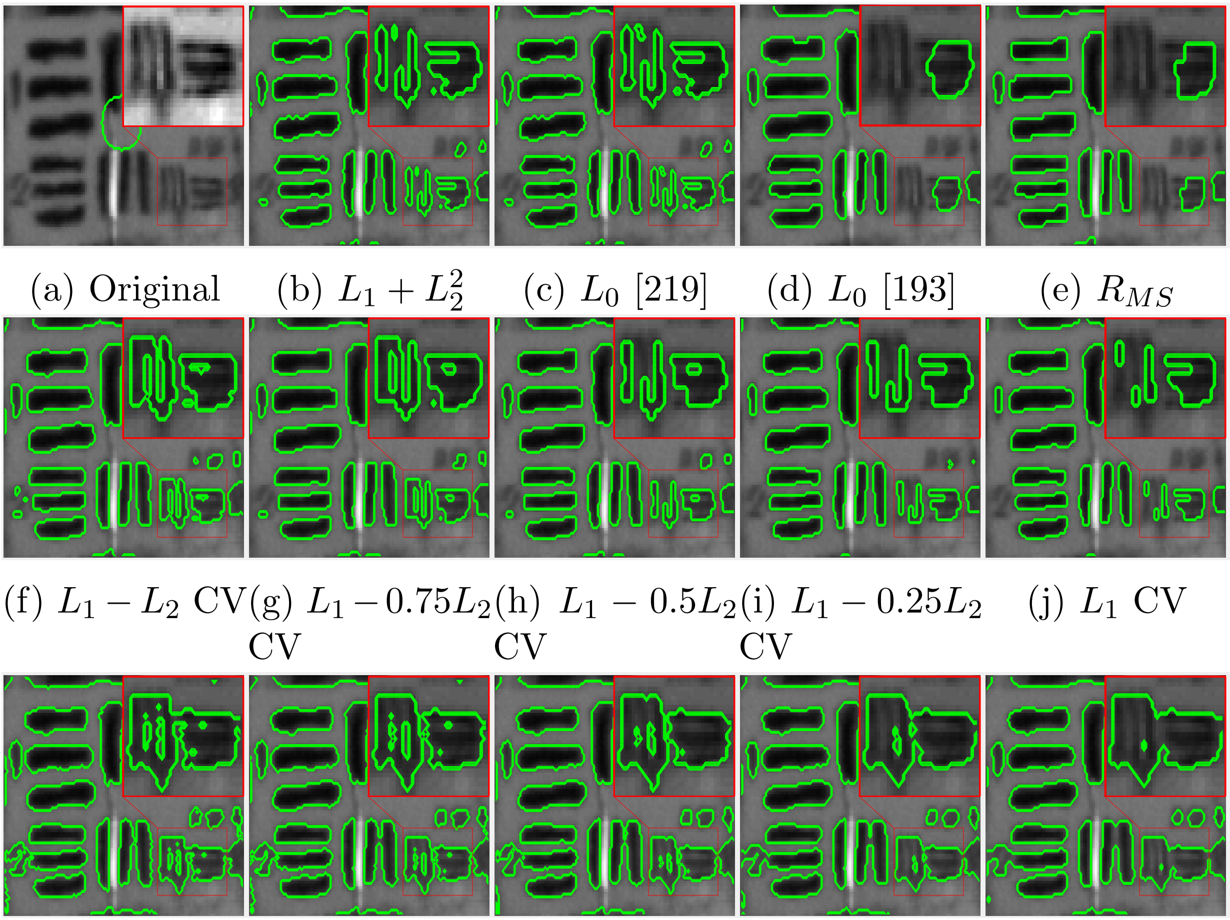
We apply the proposed methods and the two-stage methods on real images (all rescaled to $[0, 1]$ for the pixel values) shown in Figure 7.6 without additive noise. Figure 7.6a is provided in [136] while Figures 7.6b-7.6e are provided by the Berkeley Segmentation Dataset and Benchmark [157]. Specifically, Figures 7.6a and 7.6b are for two-phase segmentation, Figure 7.6c is for four-phase segmentation, and Figures 7.6d and 7.6e are for five-phase and seven-phase segmentation, respectively. We set the maximum number of inner iterations for CV/FR methods as 300, and the maximum number of outer iterations for CV as 20. The maximum outer iteration number of the FR methods depends on images, which is set to 40 for Figures 7.6a-7.6b, 80 for Figure 7.6c, and 160 for Figures 7.6d-7.6e. Following the work of [104], we compute the peak signal-to-noise ratio (PSNR) between the reconstructed image $\tilde{\mathbf{f}}$ derived by (7.56) and the original image $\mathbf{f}$. PSNR is defined by $10 \log_{10} \frac{3mn}{\sum_{\iota \in \{r,g,b\}} \|\tilde{f}_\iota - f_\iota\|_X^2}$, and it quantitatively measures the quality of the segmentation results for real color images without ground truth. The PSNR values are recorded in Table 7.4. As the CV methods are inapplicable to non-power-of-2 segmentation examples, we indicate by NA ("not applicable") their results on Figures 7.6d-7.6e in Table 7.4.

For Figure 7.6a, we set $\lambda = 100$ for all methods, except for L_0 [219] in which $\lambda = 10000$. For all FR methods, we set $\nu = 35$. The initialization for the CV and FR methods is a step function of a circle in the image center with radius 10. The segmentation results of these competing methods are displayed in Figure 7.7, each equipped with a zoomed-in region of the bottom right of the image. We observe that as α decreases, the CV methods segment lesser regions, while the FR methods identify lesser gaps. The results of the two-stage methods are not as detailed as the results provided by $L_1 - L_2$ CV and FR.

For Figure 7.6b, we set $\lambda = 50$ for L_0 [219], $\lambda = 10$ for the other methods, and $\nu = 10.0$ for the FR methods. The initialization for the CV and FR methods is the same as Figure 7.6a. Quantitative comparison of these methods is listed in Table 7.4, showing that the AICV and AIFR methods outperform their L_1 counterparts. The visual results in Figure 7.8 demonstrate that AICV and AIFR can segment finer details, especially on the branch on the left side of the image and on the hawk, than their L_1 counterparts, which thereby explains their higher PSNR values.

For Figure 7.6c, we set $\lambda = 1000$ for all methods and $\nu = 650$ for the FR methods. Initialization for the CV methods are two step functions of circles both with radius 10, one shifted 5 pixels to the left of the image center and the other shifted 5 pixels to the right. For the FR methods, the initialization of the membership functions are uniformly distributed in $[0, 1]$ and then normalized. Figure 7.9 compares the AIFR and AICV methods (using the optimal α value that corresponds to the highest PSNR in Table 7.4) with their L_1 counterparts. As PSNR values are all similar, we do not observe much visual differences between the images in Figure 7.9.

For Figure 7.6d, we set $\lambda = 650$ for all methods, except L_0 [219] in which $\lambda = 1000$. For the



(a) Original (b) $L_1 + L_2^2$ (c) L_0 [219] (d) L_0 [193] (e) R_{MS}

(f) $L_1 - L_2$ CV (g) $L_1 - 0.75L_2$ CV (h) $L_1 - 0.5L_2$ CV (i) $L_1 - 0.25L_2$ CV (j) L_1 CV

(k) $L_1 - L_2$ FR (l) $L_1 - 0.75L_2$ FR (m) $L_1 - 0.5L_2$ FR (n) $L_1 - 0.25L_2$ FR (o) L_1 FR

Figure 7.7: Segmentation results on Figure 7.6a. (The images may need to be zoomed in on a pdf reader to see the differences.)

FR methods, we set $\nu = 1050$. For Figure 7.6e, we set $\lambda = 500$ for all methods and $\nu = 400$ for the FR methods. Initialization of the membership functions for the FR methods is the same as for Figure 7.6c. The segmentation results of the FR methods and the two-stage methods are shown in Figures 7.10 and 7.11. In Figure 7.10, the results of the FR methods have better contrast than the result of $L_1+L_2^2$ and thus they look more similar to the original image. In Figure 7.11, $L_1 - L_2$ FR, L_1 FR, and L_0 are unable to identify the yellow/orange peppers behind the red peppers, which explains their lower PSNR values. Although the results of the AIFR methods for $\alpha = 0.25, 0.5, 0.75$ appear similar to $L_1 + L_2^2$ and R_{MS}, $L_1 - 0.5L_2$ attains the best segmentation based on its PSNR value.

(a) Original (b) $L_1 - 0.5L_2$ CV (c) L_1 CV (d) $L_1 - 0.75L_2$ FR (e) L_1 FR

Figure 7.8: Reconstruction results on Figure 7.6b.

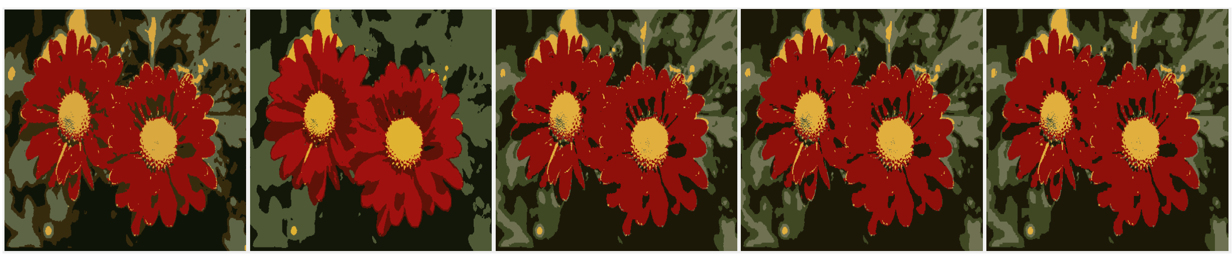
(a) Original (b) $L_1 - 0.75L_2$ CV (c) L_1 CV (d) $L_1 - 0.75L_2$ FR (e) L_1 FR

Figure 7.9: Reconstruction results on Figure 7.6c.

Last, we report the computational times of the segmentation methods in Table 7.5. Admittedly, the proposed methods are slower compared to other segmentation methods. Besides, our computational times largely depend on the image size, the number of channels, and the number of u_k's needed to segment. The acceleration of the proposed scheme will be left for future investigation.

In summary, given particular choices of α, the AITV models outperform their L_1 counterparts and the two-stage methods. For Figure 7.6a, larger values of α provide better segmentation results, but this may not be the case for other images. Thus, the optimal α value in an AITV model varies for an individual image. In addition, although the AITV methods tend to be slower than the two-stage methods, they are consistently more accurate based on their PSNR values. This observation is apparent in Figures 7.6c-7.6e, the most complex images tested in this section.

(a) Original (b) $L_1 - L_2$ FR (c) $L_1 - 0.75L_2$ FR (d) $L_1 - 0.5L_2$ FR (e) $L_1 - 0.25L_2$ FR



(f) L_1 FR (g) $L_1 + L_2^2$ (h) L_0 [219] (i) L_0 [193] (j) R_{MS}

Figure 7.10: Reconstruction results on Figure 7.6d.

(a) Original (b) $L_1 - L_2$ FR (c) $L_1 - 0.75L_2$ FR (d) $L_1 - 0.5L_2$ FR (e) $L_1 - 0.25L_2$ FR

(f) L_1 FR (g) $L_1 + L_2^2$ (h) L_0 [219] (i) L_0 [193] (j) R_{MS}

Figure 7.11: Reconstruction results on Figure 7.6e.

Chapter 8

An Efficient Smoothing and Thresholding Image Segmentation Framework with Weighted Anisotropic-Isotropic Total Variation

In this chapter, we propose an efficient ADMM framework to solve the AITV variant of (6.4) and demonstrate its efficiency and effectiveness in the SaT/SLaT framework through various numerical experiments. The efficiency lies in the closed-form solution [138] of the proximal operator for $\ell_1 - \alpha\ell_2$ to avoid nested loops in DCA as considered in [24, 217]. The main contributions of this paper are summarized as follows:

1. We provide model analysis such as coerciveness and the existence of global minimizers for the AITV-regularized variant of (6.4).
2. We develop an efficient ADMM algorithm for minimizing the AITV-based MS model based on the proximal operator of $\ell_1 - \alpha\ell_2$ with a convergence guarantee.

3. We conduct extensive numerical experiments to showcase that the SaT/SLaT framework with AITV regularization is a competitive segmentation method, especially using our proposed ADMM algorithm. The segmentation framework is robust to noise, blur, and intensity inhomogeneity.

4. We demonstrate experimentally that the proposed ADMM framework is significantly more efficient than DCA used in [24, 217] in producing segmentation results of comparable or even better quality.

8.1 Preliminaries

8.1.1 Notations

For simplicity, we adopt the discrete notations for images and mathematical models. Without loss of generality, an image is represented as an $M \times N$ matrix, so the image domain is $\Omega = \{1, 2, \ldots, M\} \times \{1, 2, \ldots, N\}$. Then we denote $X := \mathbb{R}^{M\times N}$. We adopt the linear index for 2D image, where for $u \in X$, we have $u_{i,j} \in \mathbb{R}$ be the $((i-1)M + j)$th component of u. The gradient operator $\nabla : X \to X \times X$ is denoted by $\nabla u = (\nabla_x u, \nabla_y u)$ with ∇_x and ∇_y being the horizontal and vertical forward difference operators, respectively, with the periodic boundary condition. Specifically, the (i, j)th entry of ∇u is defined by

$$(\nabla u)_{i,j} = \begin{bmatrix} (\nabla_x u)_{i,j} \\ (\nabla_y u)_{i,j} \end{bmatrix},$$

where

$$(\nabla_x u)_{i,j} = \begin{cases} u_{i,j} - u_{i,j-1} & \text{if } 2 \leq j \leq N, \\ u_{i,1} - u_{i,n} & \text{if } j = 1 \end{cases}$$

and

$$(\nabla_y u)_{i,j} = \begin{cases} u_{i,j} - u_{i-1,j} & \text{if } 2 \leq i \leq M, \\ u_{1,j} - u_{m,j} & \text{if } i = 1. \end{cases}$$

For $p = (p_x, p_y) \in X \times X$, its $((i-1)M + j)$th component is $p_{i,j} = \begin{bmatrix} (p_x)_{i,j} \\ (p_y)_{i,j} \end{bmatrix} \in \mathbb{R}^2$. We define the following norms on $X \times X$:

$$\|p\|_1 = \sum_{i=1}^{M} \sum_{j=1}^{N} |(p_x)_{i,j}| + |(p_y)_{i,j}|,$$
$$\|p\|_2 = \sqrt{\sum_{i=1}^{M} \sum_{j=1}^{N} |(p_x)_{i,j}|^2 + |(p_y)_{i,j}|^2},$$
$$\|p\|_{2,1} = \sum_{i=1}^{M} \sum_{j=1}^{N} \sqrt{(p_x)_{i,j}^2 + (p_y)_{i,j}^2}.$$

8.1.2 Review of SaT/SLaT

Both SaT and SLaT frameworks consist of two general steps: (1) smoothing to extract a piecewise-smooth approximation of a given image and (2) thresholding to segment the regions via k-means clustering. SLaT has an intermediate stage called lifting, which generates additional color channels as oppposed to the RGB color space for the smoothed image. More details for each stage are described below.

First Stage: Smoothing

Let $f = (f_1, \ldots, f_d) \in X^d$, where d represents the number of color channels of the image f. The discretized model of (6.4) for each color channel $\ell = 1, \ldots, d$ can be expressed as

$$\min_{u_\ell} \frac{\lambda}{2}\|f_\ell - Au_\ell\|_2^2 + \frac{\mu}{2}\|\nabla u_\ell\|_2^2 + \|\nabla u_\ell\|_{2,1}, \tag{8.1}$$

where $\lambda > 0, \mu \geq 0$ and $\|\nabla u\|_2^2$ is a smoothing term to reduce the staircase effects caused by the isotropic TV $\|\nabla u_\ell\|_{2,1}$. We assume the same pair of parameters (λ, μ) across color channels. In summary, we obtain a smooth approximation u_ℓ for each channel f_ℓ by solving (8.1).

Intermediate Stage: Lifting

For a color image $f = (f_1, f_2, f_3) \in X^3$, where f_1, f_2, and f_3 are the red, green, and blue channels, respectively, we can obtain (u_1, u_2, u_3) by applying the smoothing stage to each channel of f. Instead of using (u_1, u_2, u_3), SLaT transforms (u_1, u_2, u_3) into $(\bar{u}_1, \bar{u}_2, \bar{u}_3)$ in the Lab space (perceived lightness, red-green, and yellow-blue) [147] and operates on a new vector-valued image $(u_1, u_2, u_3, \bar{u}_1, \bar{u}_2, \bar{u}_3)$. The rationale is that RGB channels are highly correlated, while the Lab space relies on numerical color differences to approximate the color differences perceived by the human eye. As a result, $(u_1, u_2, u_3, \bar{u}_1, \bar{u}_2, \bar{u}_3)$ leads to better segmentation results compared to (u_1, u_2, u_3).

Final Stage: Thresholding

After rescaling the image obtained after smoothing and/or lifting, we denote the resultant image by $u^* \in [0, 1]^D$ (For SaT, $D = 1$; for SLaT, $D = 6$ if the original image is RGB.) Suppose the number of segmented regions is given and denoted by k. The thresholding stage

Algorithm 6: AITV SaT/SLaT

1 **Input:**

- image $f = (f_1, \dots, f_d)$
- blurring operator A
- fidelity parameter $\lambda > 0$
- smoothing parameter $\mu \geq 0$
- AITV parameter $\alpha \in [0, 1]$
- the number of regions in the image k

Output: Segmentation $\tilde{f}$ Stage one: Compute u_ℓ by solving (8.3) for $\ell = 1, \dots, d$.
Stage two: **if** f *is a grayscale image, i.e.,* $d = 1$ **then**
  Go to stage three.
else if f *is a color image, i.e,* $d = 3$ **then**
  Transfer $u = (u_1, u_2, u_3)$ into Lab space to obtain $(\bar{u}_1, \bar{u}_2, \bar{u}_3)$ and concatenate to form $(u_1, u_2, u_3, \bar{u}_1, \bar{u}_2, \bar{u}_3)$.
Stage three: Apply k-means to obtain $\{(c_l, \Omega_l)\}_{l=1}^k$ and compute $\tilde{f}$ by (8.2).

applies k-means clustering to the vector-valued image u^*, providing k centroids $c_1, c_2, \dots, c_k$. These centroids are used to form the regions

$$\Omega_l = \left\{ (i,j) \in \Omega : \|u^*_{i,j} - c_l\|_2 = \min_{1 \leq \kappa \leq k} \|u^*_{i,j} - c_\kappa\|_2 \right\},$$

for $l = 1, \dots, k$ such that Ω_l's are disjoint and $\bigcup_{l=1}^k \Omega_l = \Omega$. Using the centorids and regions, we can obtain a piecewise-constant approximation of f, denoted by

$$\tilde{f} = \sum_{l=1}^{k} c_l \mathbb{1}_{\Omega_l}, \quad \text{where } \mathbb{1}_{\Omega_l} = \begin{cases} 1 & \text{if } (i,j) \in \Omega_l, \\ 0 & \text{if } (i,j) \notin \Omega_l. \end{cases} \tag{8.2}$$

8.2 Smoothing with AITV Regularization

We replace the isotropic TV in (8.1) by a weighted difference of anisotropic and isotropic TV, i.e.,

$$\min_u F(u) := \frac{\lambda}{2}\|f - Au\|_2^2 + \frac{\mu}{2}\|\nabla u\|_2^2 + \|\nabla u\|_1 - \alpha\|\nabla u\|_{2,1}, \tag{8.3}$$

with $\lambda > 0, \mu \geq 0, \alpha \in [0, 1]$. AITV is a more suitable alternative to TV (no matter whether it is anisotropic or isotropic), since TV typically fails to recover oblique edges [19, 56], which can be preserved by AITV [24, 141]. To simplify notations, we omit the subscript ℓ in (8.1), as the smoothing model is applied channel by channel independently. We show that our model (8.3) admits a global solution in Section 8.2.1. To find a solution to (8.3), we describe in Section 8.2.2 the ADMM scheme with its convergence analysis conducted in Section 8.2.3. The overall AITV SaT/SLaT framework for segmentation is summarized in Algorithm 6.

8.2.1 Model Analysis

In Theorem 8.1 we establish the existence of a global solution to (8.3) by showing that its objective function F is coercive in Lemma 8.1.

Lemma 8.1. *If $\lambda > 0, \mu \geq 0, \alpha \in [0, 1)$, and $ker(A) \cap ker(\nabla) = \{0\}$, then F defined in* (8.3) *is coercive.*

Proof. We prove by contradiction. Suppose there exists a sequence $\{u_n\}_{n=1}^\infty$ and a constant $C > 0$ such that $\|u_n\|_2 \to \infty$ and $F(u_n) < C$ for all $n \in \mathbb{N}$. We define a sequence $\{v_n\}_{n=1}^\infty$ where $v_n = \dfrac{u_n}{\|u_n\|_2}$ and thereby satisfies $\|v_n\|_2 = 1$ for all $n \in \mathbb{N}$. Since $\{v_n\}_{n=1}^\infty$ is bounded, there exists a convergent subsequence $\{v_{n_k}\}_{k=1}^\infty$ such that $v_{n_k} \to v^*$ and $\|v^*\|_2 = 1$.

It follows from $\|\nabla u\|_{2,1} \leq \|\nabla u\|_1$ that

$$F(u) \geq \frac{\lambda}{2}\|Au - f\|_2^2 + (1-\alpha)\|\nabla u\|_1 \geq \frac{\lambda}{2}(\|Au\|_2 - \|f\|_2)^2 + (1-\alpha)\|\nabla u\|_1.$$

Since $F(u_n) < C$, we have $\|\nabla u_n\|_1 < \frac{C}{1-\alpha}$ and $\|Au_n\|_2 < \sqrt{\frac{2C}{\lambda}} + \|f\|_2$. As a result, we have

$$\begin{aligned}
\|Av_{n_k}\|_2 &= \frac{\|Au_{n_k}\|_2}{\|u_{n_k}\|_2} < \frac{\sqrt{\frac{2C}{\lambda}} + \|f\|_2}{\|u_{n_k}\|_2} \\
\|\nabla v_{n_k}\|_1 &= \frac{\|\nabla u_{n_k}\|_1}{\|u_{n_k}\|_2} < \frac{C}{(1-\alpha)\|u_{n_k}\|_2}.
\end{aligned}$$

After taking the limit $n_k \to \infty$, we get $\|Av^*\|_2 = 0$ and $\|\nabla v^*\|_1 = 0$, which implies that $v^* = 0$ due to the assumption that $\ker(A) \cap \ker(\nabla) = \{0\}$. However, it contradicts with $\|v^*\|_2 = 1$, and hence F is coercive. $\square$

Theorem 8.1. *If $\lambda > 0, \mu \geq 0, \alpha \in [0,1)$, and ker$(A) \cap$ ker$(\nabla) = \{0\}$, then F has a global minimizer.*

Proof. As F is lower bounded by 0, it has a minimizing sequence $\{u_n\}_{n=1}^{\infty}$. Without loss of generality, we assume $u_1 = 0$. Since F is coercive by Lemma 8.1, we have $F(u_n) \leq F(0) < \infty$, showing that $\{\|\nabla u_n\|_1\}_{n=1}^{\infty}$ and $\{\|Au_n\|_2\}_{n=1}^{\infty}$ are bounded. As $\ker(A) \cap \ker(\nabla) = \{0\}$, we have $\{u_n\}_{n=1}^{\infty}$ shall be bounded. Then there exists a convergent subsequence $\{u_{n_k}\}_{k=1}^{\infty}$ such that $u_{n_k} \to u^*$. Since A and ∇ are both bounded, linear operators, we have $Au_{n_k} \to Au^*$ and $\nabla u_{n_k} \to \nabla u^*$. Since norms are continuous and thereby lower semi-continuous, we have

$$\begin{aligned}
&\|\nabla u^*\|_1 - \alpha\|\nabla u^*\|_{2,1} \leq \liminf_{k\to\infty} \left(\|\nabla u_{n_k}\|_1 - \alpha\|\nabla u_{n_k}\|_{2,1}\right), \\
&\|\nabla u^*\|_2^2 \leq \liminf_{k\to\infty} \|\nabla u_{n_k}\|_2^2, \\
&\|Au^* - f\|_2^2 \leq \liminf_{k\to\infty} \|Au_{n_k} - f\|_2^2.
\end{aligned}$$

Altogether, we obtain $F(u^*) \leq \liminf\limits_{k\to\infty} F(u_{n_k})$, which implies that u^* minimizes $F(u)$. $\square$

8.2.2 Numerical Scheme

We describe an efficient algorithm to minimize (8.3) via ADMM. In particular, we introduce an auxiliary variable $w = (w_x, w_y) \in X \times X$ and rewrite (8.3) into an equivalent constrained optimization problem

$$\begin{aligned} \min_{u,w} \quad & \frac{\lambda}{2}\|f - Au\|_2^2 + \frac{\mu}{2}\|\nabla u\|_2^2 + \|w\|_1 - \alpha\|w\|_{2,1} \\ \text{s.t.} \quad & \nabla u = w, \end{aligned} \tag{8.4}$$

where $w_x = \nabla_x u$ and $w_y = \nabla_y u$. Then the corresponding augmented Lagrangian is expressed by

$$\begin{aligned} \mathcal{L}_\delta(u, w, z) :=& \frac{\lambda}{2}\|f - Au\|_2^2 + \frac{\mu}{2}\|\nabla u\|_2^2 + \|w\|_1 - \alpha\|w\|_{2,1} \\ &+ \langle z, \nabla u - w\rangle + \frac{\delta}{2}\|\nabla u - w\|_2^2 \\ =& \frac{\lambda}{2}\|f - Au\|_2^2 + \frac{\mu}{2}\|\nabla u\|_2^2 + \|w\|_1 - \alpha\|w\|_{2,1} \\ &+ \frac{\delta}{2}\left\|\nabla u - w + \frac{z}{\delta}\right\|_2^2 - \frac{1}{2\delta}\|z\|_2^2, \end{aligned}$$

where $\delta > 0$ is a penalty parameter and $z = (z_x, z_y) \in X \times X$ is a dual variable. The ADMM iterations proceed as follows:

$$u_{t+1} \in \arg\min_u \mathcal{L}_{\delta_t}(u, w_t, z_t) \tag{8.5a}$$

$$w_{t+1} \in \arg\min_w \mathcal{L}_{\delta_t}(u_{t+1}, w, z_t) \tag{8.5b}$$

$$z_{t+1} = z_t + \delta_t(\nabla u_{t+1} - w_{t+1}) \tag{8.5c}$$

$$\delta_{t+1} = \sigma\delta_t, \ \sigma \geq 1. \tag{8.5d}$$

Note that $\sigma = 1$ reduces to the original ADMM framework [21]. We consider an adaptive penalty parameter δ_t by choosing $\sigma > 1$. In fact, the parameter $\sigma > 1$ controls the numerical convergence speed of the algorithm in the sense that a larger σ leads to a less number of iterations the algorithm needs to run before satisfying a stopping criterion. However, if δ_t increases too quickly, the ADMM algorithm will numerically converge within a few iterations, which may yield a low-quality solution. Thus, a small σ is recommended and we discuss its choice in experiments (Section 8.3).

Next we elaborate on how to solve the two subproblems (8.5a) and (8.5b). The subproblem (8.5a) is written as

$$u_{t+1} \in \arg\min_u \frac{\lambda}{2}\|f - Au\|_2^2 + \frac{\mu}{2}\|\nabla u\|_2^2 + \langle z_t, \nabla u - w_t\rangle + \frac{\delta_t}{2}\|\nabla u - w_t\|_2^2.$$

The first-order optimality condition of (8.5a) is given by

$$\left[\lambda A^\top A - (\mu + \delta_t)\Delta\right] u_{t+1} = \lambda A^\top f + \delta_t \nabla^\top \left(w_t - \frac{z_t}{\delta_t}\right),$$

where $\Delta = -\nabla^\top\nabla$ is the Laplacian operator. If $\ker(A)\cap\ker(\nabla) = \{0\}$, then $\lambda A^\top A - (\mu+\delta_t)\Delta$ is positive definite. By assuming the periodic boundary condition, $A^\top A$ and Δ are block circulant, so we can solve for u_{t+1} via the fast Fourier transform $\mathcal{F}$ [41, 165, 211]. By the Convolution Theorem, the closed-form solution for u_{t+1} is

$$u_{t+1} = \mathcal{F}^{-1}\left(\frac{\lambda\mathcal{F}(A)^* \circ \mathcal{F}(f) + \delta_t\mathcal{F}(\nabla)^* \circ \mathcal{F}\left(w_t - \frac{z_t}{\delta_t}\right)}{\lambda\mathcal{F}(A)^* \circ \mathcal{F}(A) - (\mu + \delta_t)\mathcal{F}(\Delta)}\right),$$

where $\mathcal{F}^{-1}$ is the inverse Fourier transform, $*$ denotes complex conjugate, $\circ$ denotes componentwise multiplication, and division is also componentwise.

Denote $w_{i,j} = \begin{bmatrix}(w_x)_{i,j}\\(w_y)_{i,j}\end{bmatrix} \in \mathbb{R}^2$ as the (i,j)th entry of w. The subproblem (8.5b) can be expressed as

$$w_{t+1} \in \arg\min_w \|w\|_1 - \alpha\|w\|_{2,1} + \frac{\delta_t}{2}\left\|\nabla u_{t+1} + \frac{z_t}{\delta_t} - w\right\|_2^2.$$

Expanding (8.5b), we get

$$\arg\min_{w} \sum_{(i,j)\in\Omega} \left(\|w_{i,j}\|_1 - \alpha \|w_{i,j}\|_2 + \frac{\delta_t}{2} \left\| (\nabla u_{t+1})_{i,j} + \frac{(z_t)_{i,j}}{\delta_t} - w_{i,j} \right\|_2^2 \right), \tag{8.6}$$

which shows that $w_{i,j}$ can be solved elementwise. Specifically, the optimal solution of $w_{i,j} \in \mathbb{R}^2$ is related to the proximal operator for $\ell_1 - \alpha\ell_2$ defined by

$$\operatorname{prox}(y; \alpha, \beta) = \arg\min_{x} \|x\|_1 - \alpha\|x\|_2 + \frac{1}{2\beta}\|x - y\|_2^2. \tag{8.7}$$

The closed-form solution for (8.7) is given in Lemma 8.2 [138]. By comparing (8.6) and (8.7), the w-update is given by $\forall (i,j) \in \Omega$,

$$(w_{t+1})_{i,j} = \operatorname{prox}\left((\nabla u_{t+1})_{i,j} + \frac{(z_t)_{i,j}}{\delta_t}; \alpha, \frac{1}{\delta_t} \right).$$

Lemma 8.2 ([138]). *Given $y \in \mathbb{R}^n$, $\beta > 0$, and $\alpha \geq 0$, the optimal solution to (8.7) can be discussed separately into the following cases:*

1. *When $\|y\|_\infty > \beta$, we have*

   $$x^* = (\|\xi\|_2 + \alpha\beta)\frac{\xi}{\|\xi\|_2},$$

   where $\xi = \operatorname{sign}(y) \circ \max(|y| - \beta, 0)$.

2. *When $(1-\alpha)\beta < \|y\|_\infty \leq \beta$, then x^* is a 1-sparse vector such that one chooses $i \in \arg\max_j(|y_j|)$ and defines $x_i^* = (|y_i| + (\alpha - 1)\beta)\operatorname{sign}(y_i)$ and the rest of the elements equal to 0.*

3. *When $\|y\|_\infty \leq (1-\alpha)\beta$, then $x^* = 0$.*

In summary, the ADMM scheme that minimizes (8.3) is presented in Algorithm 7.

Algorithm 7: ADMM for minimizing the AITV-Regularized smoothing model

1 **Input:**

- image f
- blurring operator A
- fidelity parameter $\lambda > 0$
- smoothing parameter $\mu \geq 0$
- AITV parameter $\alpha \in [0,1]$
- penalty parameter $\delta_0 > 0$
- penalty multiplier $\sigma \geq 1$
- relative error $\epsilon > 0$

Output:u_t Initialize u_0, w_0, z_0.
Set $t = 0$.
while $\frac{\|u_t - u_{t-1}\|_2}{\|u_t\|_2} > \epsilon$ **do**

$$u_{t+1} = \mathcal{F}^{-1}\left(\frac{\lambda \mathcal{F}(A)^* \circ \mathcal{F}(f) + \delta_t \mathcal{F}(\nabla)^* \circ \mathcal{F}\left(w_t - \frac{z_t}{\delta_t}\right)}{\lambda \mathcal{F}(A)^* \circ \mathcal{F}(A) - (\mu + \delta_t)\mathcal{F}(\Delta)}\right)$$

$$(w_{t+1})_{i,j} = \text{prox}\left((\nabla u_{t+1})_{i,j} + \frac{(z_t)_{i,j}}{\delta_t}; \alpha, \frac{1}{\delta_t}\right) \quad \forall (i,j) \in \Omega$$

$$z_{t+1} = z_t + \delta_t(\nabla u_{t+1} - w_{t+1})$$

$$\delta_{t+1} = \sigma \delta_t$$

$$t := t+1$$

8.2.3 Convergence Analysis

We aim to analyze the convergence for Algorithm 7. It is true that global convergence of ADMM has been established in [58] for certain classes of nonconvex optimization problems, but unfortunately it cannot be applied to our problem (8.4) since the gradient operator ∇ is not surjective. Instead of global convergence, we manage to achieve weaker subsequential convergence for two cases: $\sigma = 1$ and $\sigma > 1$. The proof of $\sigma > 1$ is adapted from [81, 235].

Before providing convergence results for ADMM, we provide a definition of subdifferential for general functions. For a function $h : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, we denote the (limiting) subdifferential by $\partial h(x)$ [184, Definition 11.10], which is defined as a set

$$\partial h(x) = \{v \in \mathbb{R}^n : \exists \{(x_t, v_t)\}_{t=1}^{\infty} \text{ s.t. } x_t \to x,\ h(x_t) \to h(x),\ \hat{\partial} h(x_t) \ni v_t \to v\},$$

with

$$\hat{\partial} h(x) = \left\{ v \in \mathbb{R}^n : \liminf_{z \to x, z \neq x} \frac{h(z) - h(x) - \langle v, z - x \rangle}{\|z - x\|_2} \geq 0 \right\}.$$

Since $\hat{\partial} h(x) \subset \partial h(x)$ where h is finite on x, the graph $x \mapsto \partial h(x)$ is closed [54, 184] by definition:

$$v_t \in \partial h(x_t),\ x_t \to x,\ h(x_t) \to h(x), v_t \to v \implies v \in \partial h(x).$$

Lemma 8.3. *Suppose that ker*$(A) \cap$ *ker*$(\nabla) = \{0\}$ *and* $\alpha \in [0, 1)$. *Let* $\{(u_t, w_t, z_t)\}_{t=1}^{\infty}$ *be generated by* (8.5a)-(8.5d) *with* $\sigma \geq 1$. *The following inequality holds,*

$$\mathcal{L}_{\delta_{t+1}}(u_{t+1}, w_{t+1}, z_{t+1}) - \mathcal{L}_{\delta_t}(u_t, w_t, z_t) \leq \frac{\sigma + 1}{2\sigma^t \delta_0} \|z_{t+1} - z_t\|_2^2 - \frac{\zeta}{2} \|u_{t+1} - u_t\|_2^2, \tag{8.9}$$

where $\zeta > 0$ *is the smallest eigenvalue of* $\lambda A^\top A + (\mu + \delta_0) \nabla^\top \nabla$.

Proof. It is straightforward that $u^\top A^\top A u = \|Au\|_2^2 \geq 0$ and $u^\top \nabla^\top \nabla u = \|\nabla u\|_2^2 \geq 0$ for any $u \in X$, so $\zeta \geq 0$. If $\zeta = 0$, then there exists a nonzero vector $x \in X$ such that $\lambda \|Ax\|_2^2 + (\mu + \delta_0) \|\nabla x\|_2^2 = \lambda x^\top A^\top A x + (\mu + \delta_0) x^\top \nabla^\top \nabla x = 0$. Then we shall have $x \in \ker(A) \cap \ker(\nabla)$, contradicting that $\ker(A) \cap \ker(\nabla) = \{0\}$. Therefore, $\zeta > 0$ and hence we get

$$\lambda \|Au\|_2^2 + (\mu + \delta_0) \|\nabla u\|_2^2 \geq \zeta \|u\|_2^2 \quad \forall u \in X.$$

As $\delta_{t+1} \geq \delta_t$ ($\sigma \geq 1$), $\mathcal{L}_{\delta_t}(u, w_t, z_t)$ is a strongly convex function of u with parameter $\zeta > 0$. Fixing w_t, z_t, the minimizer u_{t+1} of $\mathcal{L}_{\delta_t}(u, w_t, z_t)$ in (8.5a) satisfies the following inequality [18, Theorem 5.25],

$$\mathcal{L}_{\delta_t}(u_{t+1}, w_t, z_t) - \mathcal{L}_{\delta_t}(u_t, w_t, z_t) \leq -\frac{\zeta}{2}\|u_{t+1} - u_t\|_2^2. \tag{8.10}$$

As w_{t+1} is the optimal solution to (8.5b), we have

$$\mathcal{L}_{\delta_t}(u_{t+1}, w_{t+1}, z_t) - \mathcal{L}_{\delta_t}(u_{t+1}, w_t, z_t) \leq 0. \tag{8.11}$$

It follows from the update (8.5c) that

$$\begin{aligned} \mathcal{L}_{\delta_t}(u_{t+1}, w_{t+1}, z_{t+1}) - \mathcal{L}_{\delta_t}(u_{t+1}, w_{t+1}, z_t) &= \langle z_{t+1} - z_t, \nabla u_{t+1} - w_{t+1}\rangle \\ &= \frac{1}{\delta_t}\|z_{t+1} - z_t\|_2^2. \end{aligned} \tag{8.12}$$

Similarly, we get

$$\begin{aligned} \mathcal{L}_{\delta_{t+1}}(u_{t+1}, w_{t+1}, z_{t+1}) - \mathcal{L}_{\delta_t}(u_{t+1}, w_{t+1}, z_{t+1}) =& \frac{\delta_{t+1} - \delta_t}{2}\|\nabla u_{t+1} - w_{t+1}\|_2^2 \\ =& \frac{\delta_{t+1} - \delta_t}{2\delta_t^2}\|z_{t+1} - z_t\|_2^2. \end{aligned} \tag{8.13}$$

Combining (8.10)-(8.13) leads to the desired inequality

$$\begin{aligned} \mathcal{L}_{\delta_{t+1}}(u_{t+1}, w_{t+1}, z_{t+1}) - \mathcal{L}_{\delta_t}(u_t, w_t, z_t) &\leq \frac{\delta_{t+1} - \delta_t}{2\delta_t^2}\|z_{t+1} - z_t\|_2^2 + \frac{1}{\delta_t}\|z_{t+1} - z_t\|_2^2 \\ &\quad - \frac{\zeta}{2}\|u_{t+1} - u_t\|_2^2 \\ &= \frac{\sigma + 1}{2\sigma^t\delta_0}\|z_{t+1} - z_t\|_2^2 - \frac{\zeta}{2}\|u_{t+1} - u_t\|_2^2. \end{aligned}$$

□

Proposition 8.1. *Suppose that ker*$(A) \cap$ *ker*$(\nabla) = \{0\}$ *and* $\alpha \in [0, 1)$*. Let* $\{(u_t, w_t, z_t)\}_{t=1}^{\infty}$

be generated by (8.5a)-(8.5d)*. Assume one of the conditions holds:*

- $\sigma = 1$ *and* $\sum_{i=0}^{\infty} \|z_{i+1} - z_i\|_2^2 < \infty$.
- $\sigma > 1$.

Then we have the following statements:

(a) The sequence $\{(u_t, w_t, z_t)\}_{t=1}^{\infty}$ *is bounded.*

(b) $u_{t+1} - u_t \to 0$*, as* $t \to \infty$.

Proof. (a) We start by proving the boundedness of $\{z_t\}_{t=1}^{\infty}$. The optimality condition of (8.5b) at iteration t is expressed by

$$0 \in \partial\left(\|w_{t+1}\|_1 - \alpha\|w_{t+1}\|_{2,1}\right) - \delta_t\left(\nabla u_{t+1} - w_{t+1}\right) - z_t. \tag{8.14}$$

Together with (8.5c), we have

$$z_{t+1} \in \partial\left(\|w_{t+1}\|_1 - \alpha\|w_{t+1}\|_{2,1}\right) \subset \partial\|w_{t+1}\|_1 - \alpha\partial\|w_{t+1}\|_{2,1}, \tag{8.15}$$

which implies that there exist two vectors $v_1 \in \partial\|w_{t+1}\|_1$ and $v_2 \in \partial\|w_{t+1}\|_{2,1}$ such that $z_{t+1} = v_1 - \alpha v_2$. For any $v \in \partial\|w\|_1$, we have

$$(v_x)_{i,j} = \operatorname{sign}((w_x)_{i,j}) \text{ and } (v_y)_{i,j} = \operatorname{sign}((w_y)_{i,j}), \tag{8.16}$$

which guarantees that $\|v\|_\infty \le 1$. If $z \in \partial\|w\|_{2,1}$, then

$$z_{i,j} = \begin{cases} \dfrac{w_{i,j}}{\|w_{i,j}\|_2} & \text{if } \|w_{i,j}\|_2 \neq 0, \\ \in \{z_{i,j} \in \mathbb{R}^2 : \|z_{i,j}\|_2 \le 1\} & \text{if } \|w_{i,j}\|_2 = 0. \end{cases} \tag{8.17}$$

By (8.17), we have $\|(v_2)_{i,j}\|_2 \leq 1$, which means that $\|v_2\|_\infty \leq 1$. As a result, $\|z_{t+1}\|_\infty \leq \|v_1\|_\infty + \alpha\|v_2\|_\infty \leq 2$. Altogether, we arrive at an upper bound, i.e.,

$$\|z_{t+1}\|_2 = \sqrt{\sum_{i,j} \left(|(z_{t+1,x})_{i,j}|^2 + |(z_{t+1,y})_{i,j}|^2\right)} \leq \sqrt{2^2(2MN)} = 2\sqrt{2MN}. \tag{8.18}$$

By telescoping summation of (8.9), we have for all t that

$$\begin{aligned}\mathcal{L}_{\delta_{t+1}}(u_{t+1}, w_{t+1}, z_{t+1}) \leq& \mathcal{L}_{\delta_0}(u_0, w_0, z_0) + \frac{(\sigma+1)}{2\delta_0}\sum_{i=0}^{t}\frac{1}{\sigma^i}\|z_{i+1} - z_i\|_2^2 \\ \leq& \mathcal{L}_{\delta_0}(u_0, w_0, z_0) + \frac{(\sigma+1)}{2\delta_0}\sum_{i=0}^{\infty}\frac{1}{\sigma^i}\|z_{i+1} - z_i\|_2^2.\end{aligned}$$

Now that $\{z_t\}_{t=1}^\infty$ is bounded, then $\{\|z_{t+1}-z_t\|_2^2\}_{t=1}^\infty$ is bounded. Denote $C := \sup_{t\in\mathbb{N}} \|z_{t+1}-z_t\|_2^2$. If $\sigma = 1$ and $\sum_{i=0}^{\infty} \|z_{i+1} - z_i\|_2^2 < \infty$, then $\{\mathcal{L}_{\delta_t}(u_t, w_t, z_t)\}_{t=1}^\infty$ is uniformly bounded above. On the other hand, if $\sigma > 1$, then we get

$$\mathcal{L}_{\delta_{t+1}}(u_{t+1}, w_{t+1}, z_{t+1}) \leq \mathcal{L}_{\delta_0}(u_0, w_0, z_0) + \frac{C(\sigma+1)}{2\delta_0}\sum_{i=0}^{\infty}\frac{1}{\sigma^i} < \infty,$$

where the infinite sum converges for $\sigma > 1$. In either case, we have that $\{\mathcal{L}_{\delta_t}(u_t, w_t, z_t)\}_{t=1}^\infty$ is uniformly bounded above, and hence there exists a constant $\tilde{C} > 0$ such that $\mathcal{L}_{\delta_t}(u_t, w_t, z_t) < \tilde{C}$.

Since $\|w\|_{2,1} \leq \|w\|_1$, we have

$$(1-\alpha)\|w_t\|_1 - \frac{1}{2\delta_t}\|z_t\|_2^2 \leq \|w_t\|_1 - \alpha\|w_t\|_{2,1} - \frac{1}{2\delta_t}\|z_t\|_2^2 \leq \mathcal{L}_{\delta_t}(u_t, w_t, z_t) \leq \tilde{C}.$$

This suggests an upper bound of $\|w_t\|_1$, i.e.,

$$\|w_t\|_1 \leq \frac{1}{1-\alpha}\left(\tilde{C} + \frac{1}{2\delta_t}\|z_t\|_2^2\right) \leq \frac{1}{1-\alpha}\left(\tilde{C} + \frac{4MN}{\delta_0}\right).$$

It further follows from (8.5c) that

$$\|\nabla u_t\|_2 = \left\| \frac{z_t - z_{t-1}}{\delta_{t-1}} + w_t \right\|_2 \le \frac{4\sqrt{2MN}}{\delta_0} + \bar{C},$$

where $\bar{C}$ is an upper bound of $\|w_t\|_2$ for all $t \in \mathbb{N}$. Lastly, we observe that

$$\frac{\lambda}{2}\|f - Au_t\|_2^2 - \frac{1}{2\delta_t}\|z_t\|_2^2 \le \mathcal{L}_{\delta_t}(u_t, w_t, z_t) \le \tilde{C}.$$

As $\{z_t\}_{t=1}^{\infty}$ is bounded, then $\{\|f - Au_t\|_2^2\}_{t=1}^{\infty}$ is bounded as well. Altogether $\{F(u_t)\}_{t=1}^{\infty}$ is a bounded sequence, and hence we conclude that $\{u_t\}_{t=1}^{\infty}$ is bounded by coercivity in Lemma 8.1.

(b) By Lemma 8.3, we can derive

$$\mathcal{L}_{\delta_{t+1}}(u_{t+1}, w_{t+1}, z_{t+1}) \le \mathcal{L}_0(u_0, w_0, z_0) + \frac{(\sigma+1)}{2\delta_0}\sum_{i=0}^{t}\frac{1}{\sigma^i}\|z_{i+1} - z_i\|_2^2 - \frac{\zeta}{2}\sum_{i=0}^{t}\|u_{i+1} - u_i\|_2^2.$$

By (8.18), we have

$$\mathcal{L}_{\delta_{t+1}}(u_{t+1}, w_{t+1}, z_{t+1}) \ge -\frac{1}{2\delta_{t+1}}\|z_{t+1}\|_2^2 \ge -\frac{4MN}{\delta_0}, \ \forall t \in \mathbb{N}. \tag{8.19}$$

Combining the two inequalities gives us

$$\begin{aligned} -\frac{4MN}{\delta_0} + \frac{\zeta}{2}\sum_{i=0}^{t}\|u_{i+1} - u_i\|_2^2 \le & \mathcal{L}_{\delta_{t+1}}(u_{t+1}, w_{t+1}, z_{t+1}) + \frac{\zeta}{2}\sum_{i=0}^{t}\|u_{i+1} - u_i\|_2^2 \\ \le & \mathcal{L}_0(u_0, w_0, z_0) + \frac{(\sigma+1)}{2\delta_0}\sum_{i=0}^{t}\frac{1}{\sigma^i}\|z_{i+1} - z_i\|_2^2. \end{aligned}$$

As $t \to \infty$, we obtain

$$0 \le \frac{\zeta}{2}\sum_{i=0}^{\infty}\|u_{i+1} - u_i\|_2^2 \le \mathcal{L}_{\delta_0}(u_0, w_0, z_0) + \frac{(\sigma+1)}{2\delta_0}\sum_{i=0}^{\infty}\frac{1}{\sigma^i}\|z_{i+1} - z_i\|_2^2 + \frac{4MN}{\delta_0}.$$

Earlier in proving the boundedness of $\{\mathcal{L}_{\delta_t}(u_t, w_t, z_t)\}_{t=1}^{\infty}$, we show that the summation $\sum_{i=0}^{\infty} \frac{1}{\sigma^i} \|z_{i+1} - z_i\|_2^2$ converges. As a result, the summation $\sum_{i=0}^{\infty} \|u_{i+1} - u_i\|_2^2$ converges, which implies that $u_{t+1} - u_t \to 0$. □

Proposition 8.1 reveals an advantage of using the adaptive penality parameter with $\sigma > 1$. For $\sigma = 1$, we require $\sum_{i=0}^{\infty} \|z_{i+1} - z_i\|_2^2 < \infty$ in order for the iterates $\{(u_t, w_t, z_t)\}_{t=1}^{\infty}$ of Algorithm 7 to be bounded and to satisfy the relative stopping criterion $\frac{\|u_t - u_{t-1}\|_2}{\|u_t\|_2} < \epsilon$. The requirement $\sum_{i=0}^{\infty} \|z_{i+1} - z_i\|_2^2 < \infty$ is no longer necessary if $\sigma > 1$.

Finally, we establish the subsequential convergence in Theorem 8.2 under stronger conditions compared to the ones in Proposition 8.1. These conditions are motivated by a series of works [46, 45, 105, 104, 120, 126] that proved the theoretical convergence of ADMM in solving TV-based inverse problems.

Theorem 8.2. *Let* $\{(u_t, w_t, z_t)\}_{t=1}^{\infty}$ *be generated by* (8.5a)-(8.5d). *Assume one set of the following conditions holds:*

- $\sigma = 1$ *and* $\sum_{i=0}^{\infty} \|z_{i+1} - z_i\|_2^2 < \infty$.
- $\sigma > 1$, $\delta_t(w_{t+1} - w_t) \to 0$, *and* $z_{t+1} - z_t \to 0$.

Then there exists a subsequence of $\{(u_t, w_t, z_t)\}_{t=1}^{\infty}$ *whose limit point* (u^*, w^*, z^*) *is a KKT point of* (8.4) *that satisfies*

$$0 = \lambda A^\top (Au^* - f) - \mu \Delta u^* + \nabla^\top z^* \tag{8.20a}$$

$$z^* \in \partial \left(\|w^*\|_1 - \alpha \|w^*\|_{2,1} \right) \tag{8.20b}$$

$$\nabla u^* = w^*. \tag{8.20c}$$

Proof. By Proposition 8.1, $\{(u_t, w_t, z_t)\}_{t=1}^{\infty}$ is bounded, and hence there exists a subsequence that converges to a point (u^*, w^*, z^*), denoted by $(u_{t_k}, w_{t_k}, z_{t_k}) \to (u^*, w^*, z^*)$. Proposition 8.1

also establishes $\lim\limits_{t\to\infty} u_{t+1} - u_t = 0$, which implies that $\lim\limits_{k\to\infty} u_{t_k+1} = \lim\limits_{k\to\infty} u_{t_k} = u^*$. Either set of assumptions establishes $\lim\limits_{k\to\infty} z_{t_k+1} = \lim\limits_{k\to\infty} z_{t_k} = z^*$. The optimality conditions at iteration t_k are

$$0 = \lambda A^\top(Au_{t_k+1} - f) - \mu\Delta u_{t_k+1} + \delta_{t_k}\nabla^\top(\nabla u_{t_k+1} - w_{t_k}) + \nabla^\top z_{t_k} \tag{8.21a}$$

$$0 \in \partial\left(\|w_{t_k+1}\|_1 - \alpha\|w_{t_k+1}\|_{2,1}\right) - \delta_{t_k}\left(\nabla u_{t_k+1} - w_{t_k+1}\right) - z_{t_k} \tag{8.21b}$$

$$z_{t_k+1} = z_{t_k} + \delta_{t_k}(\nabla u_{t_k+1} - w_{t_k+1}). \tag{8.21c}$$

Next we discuss two sets of assumptions individually.

If $\sigma = 1$, then $\delta_{t_k} = \delta_0$ for each iteration t_k. Together with $\lim\limits_{t\to\infty} z_{t+1} - z_t = 0$, we have $\lim\limits_{t\to\infty} \nabla u_t - w_t = 0$ by (8.5c) and

$$\nabla u^* = \lim_{k\to\infty} \nabla u_{t_k} = \lim_{k\to\infty}(\nabla u_{t_k} - w_{t_k}) + \lim_{k\to\infty} w_{t_k} = w^*,$$

leading to (8.20c). According to (8.21a), the point u_{t_k+1} satisfies

$$\begin{aligned}
0 =&\lambda A^\top(Au_{t_k+1} - f) - \mu\Delta u_{t_k+1} + \delta_0\nabla^\top(\nabla u_{t_k+1} - w_{t_k}) + \nabla^\top z_{t_k}\\
=&\lambda A^\top(Au_{t_k+1} - f) - \mu\Delta u_{t_k+1} + \delta_0\nabla^\top(\nabla u_{t_k+1} - \nabla u_{t_k}) + \delta_0\nabla^\top(\nabla u_{t_k} - w_{t_k})\\
&+ \nabla^\top z_{t_k}.
\end{aligned}$$

Then (8.20a) holds after taking $k \to \infty$. Finally, we have

$$\lim_{k\to\infty} w_{t_k+1} = \lim_{k\to\infty}(w_{t_k+1} - \nabla u_{t_k+1}) + \lim_{k\to\infty} \nabla u_{t_k+1} = \lim_{k\to\infty} \nabla u_{t_k} = w^*.$$

If $\sigma > 1$ and $\delta_t(w_{t+1} - w_t) \to 0$, we substitute (8.21c) into (8.21a) and simplify it to obtain

$$0 = \lim_{k\to\infty} \lambda A^\top(Au_{t_k+1} - f) - \mu\Delta u_{t_k} + \delta_{t_k}\nabla^\top(w_{t_k+1} - w_{t_k}) + \nabla^\top z_{t_k+1}$$

$$= \lambda A^\top (Au^* - f) - \mu \Delta u^* + \nabla^\top z^*.$$

We need to prove $\lim_{k\to\infty} w_{t_k+1} = w^*$. By (8.5c), we have

$$\begin{aligned}
\|w_{t+1} - w_t\|_2 &\leq \|w_{t+1} - \nabla u_{t+1}\|_2 + \|\nabla u_{t+1} - \nabla u_t\|_2 + \|\nabla u_t - w_t\|_2 \\
&= \left\| \frac{z_{t+1} - z_t}{\delta_t} \right\|_2 + \|\nabla u_{t+1} - \nabla u_t\|_2 + \left\| \frac{z_t - z_{t-1}}{\delta_{t-1}} \right\|_2 \\
&\leq \frac{4C}{\delta_{t-1}} + \|\nabla u_{t+1} - \nabla u_t\|_2.
\end{aligned}$$

Taking the limit $t \to \infty$, we obtain $\|w_{t+1} - w_t\|_2 \to 0$ and $w_{t+1} - w_t \to 0$. It follows that

$$\lim_{k\to\infty} w_{t_k+1} - w_{t_k} = 0 \implies \lim_{k\to\infty} w_{t_k+1} = \lim_{k\to\infty} w_{t_k} = w^*.$$

Since $\{z_t\}_{t=1}^\infty$ is bounded in this case, there exists $C > 0$ such that $\|z_t\|_2 \leq C$. Then (8.21c) implies

$$\|\nabla u^* - w^*\|_2 = \lim_{k\to\infty} \|\nabla u_{t_k+1} - w_{t_k+1}\|_2 = \lim_{k\to\infty} \frac{1}{\delta_{t_k}} \|z_{t_k+1} - z_{t_k}\|_2 \leq \lim_{k\to\infty} \frac{2C}{\delta_{t_k}} = 0.$$

As a result, we have $\nabla u^* = w^*$.

By substituting (8.21c) into (8.21b), we have

$$z_{t_k+1} \in \partial \left(\|w_{t_k+1}\|_1 - \alpha \|w_{t_k+1}\|_{2,1} \right) \quad \forall k \in \mathbb{N}.$$

By continuity, we have $\|w_{t_k+1}\|_1 - \alpha\|w_{t_k+1}\|_{2,1} \to \|w^*\|_1 - \alpha\|w^*\|_{2,1}$. Together with the fact that $(w_{t_k+1}, z_{t_k+1}) \to (w^*, z^*)$, we obtain $z^* \in \partial \left(\|w^*\|_1 - \alpha\|w^*\|_{2,1} \right)$.

Therefore, if either set of assumptions hold, then (u^*, w^*, z^*) is a KKT point of (8.4). $\square$

8.3 Experimental Results

We examine the SaT/SLaT framework by comparing the isotropic TV[1] [29, 28], $\text{TV}^p (0 < p < 1)$ [216], and the AITV. The experiment comparison also includes the AITV-regularized CV and fuzzy region (FR) model [24] together with the Potts model [180] solved by either a primal-dual algorithm[2] [176] or ADMM[3] [193]. In particular, the primal-dual algorithm solves a convex relaxation of the Potts model [176]:

$$U^* = \operatorname*{arg\,min}_{U \in S} \sum_{\ell=1}^{k} \left[\lambda \sum_{(i,j) \in \Omega} (u_\ell)_{i,j} |(u_\ell)_{i,j} - c_\ell|^2 + \|\nabla u_\ell\|_{2,1} \right], \tag{8.22}$$

where k is the number of regions specified in an image, $\{c_\ell\}_{\ell=1}^{k} \subset \mathbb{R}$ are constant values, and

$$S = \left\{ U = (u_1, u_2, \ldots, u_k) \in X^k : \forall\, (i,j) \in \Omega, \sum_{\ell=1}^{k} (u_\ell)_{i,j} = 1;\; (u_\ell)_{i,j} \in [0,1], \ell = 1, \ldots, k \right\}.$$

Once getting U^* from (8.22), the regions of an image can be approximated by

$$\Omega_\kappa = \left\{ (i,j) \in \Omega : \kappa = \operatorname*{arg\,max}_{1 \leq \ell \leq k} (u_\ell^*)_{i,j} \right\},$$

with $\kappa = 1, \ldots, k$. For short, we refer (8.22) as the convex Potts model. To apply ADMM, Storath and Weinmann [193] considered the following version of the Potts model:

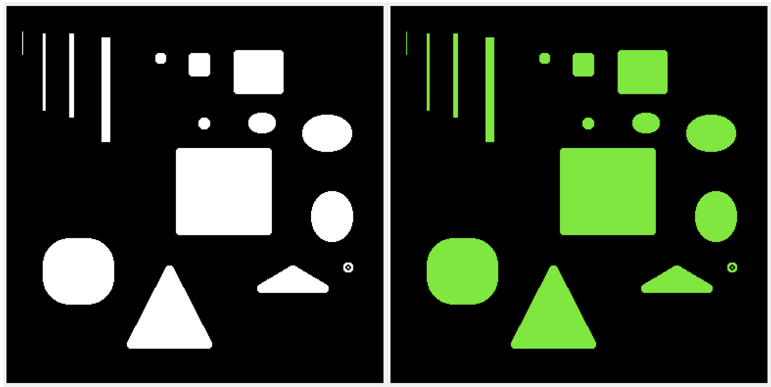
$$\min_u \lambda \|u - f\|_2^2 + \|\nabla u\|_0. \tag{8.23}$$

Since it does not admit a segmentation result with a chosen number of regions, we develop its SaT version called SaT-Potts that solves (8.23), followed by the k-means clustering for segmentation. Both (8.22) and (8.23) can deal with multichannel input; please refer to

[1] MATLAB code is available at `https://xiaohaocai.netlify.app/download/`.

[2] Python code is available at `https://github.com/VLOGroup/pgmo-lecture/blob/master/notebooks/tv-potts.ipynb` and a translated MATLAB code is available at `https://github.com/kbui1993/MATLAB_Potts`.

[3] Code is available at `https://github.com/mstorath/Pottslab`.



(a) (b)

Figure 8.1: Synthetic images for two-phase segmentation. (a) Grayscale image and (b) Color image. Size: 385×385.

[176, 193] for more details.

To ease the parameter tuning, we scale the pixel intensity of all the testing images in our experiments to $[0, 1]$. The fidelity parameter λ and the smoothing parameter μ are tuned for each image, which will be specified later. Stage 1 of the isotropic TV SaT/SLaT is solved using the authors' official code that is implemented by a similar ADMM algorithm to Algorithm 7 with $\sigma = 1$. Stage 1 of TV^p and AITV SaT/SLaT is solved by Algorithm 7 with $\sigma = 1.25$ using the appropriate proximal operators. We set the penalty parameter in Algorithm 7 to be $\delta_0 = 1.0, 2.0$ for grayscale and multichannel images, respectively. The stopping criterion for the ADMM algorithms are until $\frac{\|u_{t+1}-u_t\|_2}{\|u_{t+1}\|_2} < 10^{-4}$ with a maximum number of 300 iterations. We compare the proposed ADMM algorithm with our own DCA implementation for AITV SaT/SLaT as described in [217]. Note that its inner minimization subproblem is solved by semi-proximal ADMM [83], which has more parameters than ADMM. We use the default parameter setting as suggested in [217].

To quantitatively evaluate the segmentation performance, we use two metrics: DICE index [60] when the ground truth is available and PSNR when the ground truth is unavailable.

The DICE index is given by

$$\text{DICE} = 2\frac{\#\{R(i) \cap R'(i)\}}{\#\{R(i)\} + \#\{R'(i)\}},$$

where $R(i)$ is the set of pixels with label i in the ground-truth image f, $R'(i)$ is the set of pixels with label i in the segmented image $\tilde{f}$, and $\#\{R\}$ refers to the number of pixels in the set R. Following the work of [104], we use PSNR to determine how well the segmented image $\tilde{f}$ approximates the original image f. It is computed by $10\log_{10}(1/\text{MSE})$, where MSE is the mean square error between f and $\tilde{f}$.

We tune various parameters in the investigated algorithms to achieve the best DICE indices or PSNRs for synthetic or real images, respectively. For all methods, we tune the fidelity parameter $\lambda \in [1.0, 3.5]$. For TV^p SaT/SLaT, we only consider $p = 1/2, 2/3$ because they are the only values that have closed-form solutions [33, 222] for their proximal operators. For the AITV related algorithms, we tune $\alpha \in \{0.2, 0.4, 0.6, 0.8\}$. For the SaT-Potts model [193], we use a default setting for the other parameters. For the convex Potts model [176], we run the algorithm for up to 150 iterations with the same stopping criterion as AITV does. Lastly, we tune $\mu \in [0.01, 1.0]$ in all SaT/SLaT methods.

All experiments are performed in MATLAB R2019a on a Dell laptop with a 1.80 GHz Intel Core i7-8565U processor and 16.0 GB of RAM. In the general SaT/SLaT framework, we use some MATLAB built-in functions. In Stage 2, `makecform('srgb2lab')` is used to convert RGB to Lab. In Stage 3, `kmeans` is executed to perform k-means clustering ten times with different initialization and selects the best arrangement among the ten solutions. We also parallelize Stage 1 for color, or generally multichannel, images to speed up the computation. To compute DICE and PSNR, we use the MATLAB functions `dice` and `psnr`. The AITV SaT/SLaT codes are available at `https://github.com/kbui1993/Official_AITV_SaT_SLaT`.

8.3.1 Two-Phase Segmentation on Synthetic Images

We compare the proposed ADMM algorithm of AITV SaT/SLaT with the other SaT/SLaT methods, the Potts models, and the AITV CV model on the synthetic images presented in Figure 8.1. We corrupt the images with either random-valued (RV) or salt-and-pepper (SP) impulsive noises. Additionally, we consider blurring the image before adding impulsive noises. Specifically, we use an average blur `fspecial('average', 15)` for Figure 8.1a and a motion blur `fspecial('motion', 5, 45)` for Figure 8.1b. For the SaT methods applied to Figure 8.1a, we set the parameters $\lambda = 1.5$ and $\mu = 1.0$. For the SLaT methods applied to Figure 8.1b, we find the optimal parameters $\lambda = 2.5$ and $\mu = 1.0$.

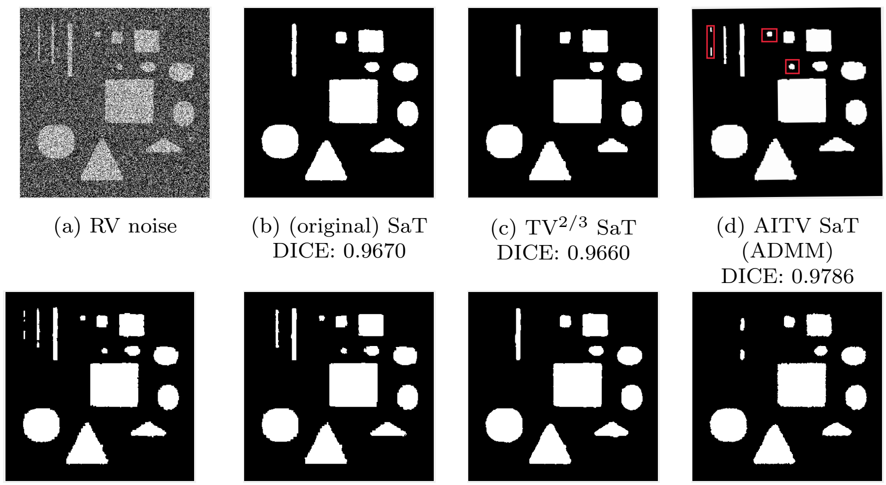
Synthetic Grayscale Images

We apply the competing segmentation methods on four types of input data based on Figure 8.1a, i.e., 65% RV noise, 65% SP noise, average blur followed by 50% RV, and average blur followed by 50% SP. The resulting DICE indices together with computational times are recorded in Table 8.1. For the noisy inputs, our proposed AITV SaT (ADMM) achieves the highest DICE indices with the least amount of time no matter which impulsive noise is added. For the blurry, noisy data, the original SaT yields the highest DICE indices. Although AITV SaT (ADMM) is the second best, it is two-three times faster than the original SaT. The AITV CV and the Potts models perform worse than the SaT methods on blurry images because, unlike the SaT methods, they do not account for blurring. Lastly, we point out that ADMM yields higher DICE than DCA by using significantly less time for the AITV SaT model.

Visual segmentation results are presented in Figures 8.2-8.3 under the RV noise with no blur and average blur, respectively, showing that the AITV SaT (ADMM) method yields binary segmentations closest to the ground truth. Specifically, AITV SaT (ADMM) identifies the

Table 8.1: Comparison of the DICE indices and computation times (seconds) between the segmentation methods applied to Figure 8.1a corrupted in four cases. Number in **bold** indicates either the highest DICE index or the fastest time among the segmentation methods for a given corrupted image.

	65% RV		65% SP		Blur and 50% RV		Blur and 50% SP	
	DICE	Time (s)	DICE	Time (s)	DICE	Time (s)	DICE	Time (s)
(Original) SaT	0.9670	3.95	0.9584	4.07	**0.9552**	4.87	**0.9497**	4.77
TV^p SaT	0.9660	1.71	0.9567	1.48	0.9488	2.00	0.9412	1.75
AITV SaT (ADMM)	**0.9786**	**1.43**	**0.9657**	**1.30**	0.9550	**1.66**	0.9470	**1.57**
AITV SaT (DCA)	0.9774	21.46	0.9655	23.01	0.9516	31.10	0.9424	31.97
AITV CV	0.9768	105.55	0.9655	152.65	0.9288	167.24	0.9164	110.14
Convex Potts	0.9665	7.07	0.9604	5.08	0.9101	5.97	0.9132	4.14
SaT-Potts	0.9480	3.49	0.9536	3.86	0.9101	5.97	0.9180	2.70



(a) RV noise

(b) (original) SaT DICE: 0.9670

(c) $\text{TV}^{2/3}$ SaT DICE: 0.9660

(d) AITV SaT (ADMM) DICE: 0.9786

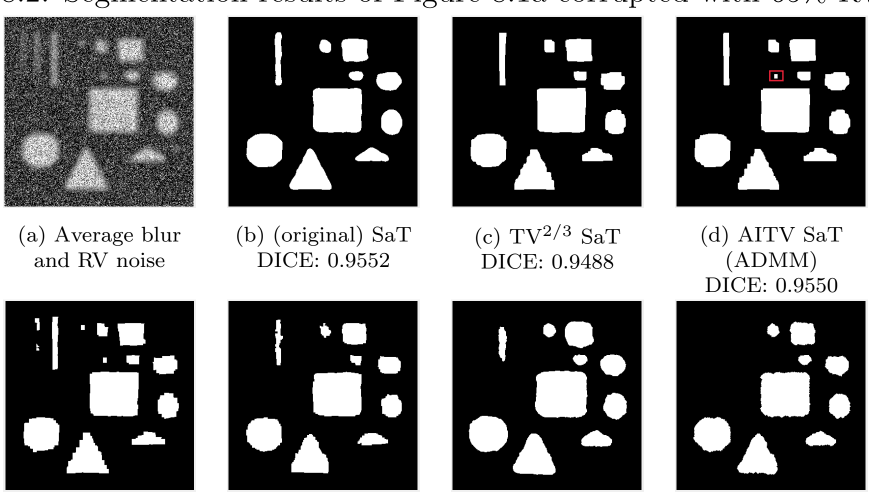
(e) AITV SaT (DCA) DICE: 0.9774

(f) AITV CV DICE: 0.9768

(g) Convex Potts DICE: 0.9665

(h) SaT-Potts DICE: 0.9480

Figure 8.2: Segmentation results of Figure 8.1a corrupted with 65% RV noise.



(a) Average blur and RV noise

(b) (original) SaT DICE: 0.9552

(c) $\text{TV}^{2/3}$ SaT DICE: 0.9488

(d) AITV SaT (ADMM) DICE: 0.9550

(e) AITV SaT (DCA) DICE: 0.9516

(f) AITV CV DICE: 0.9288

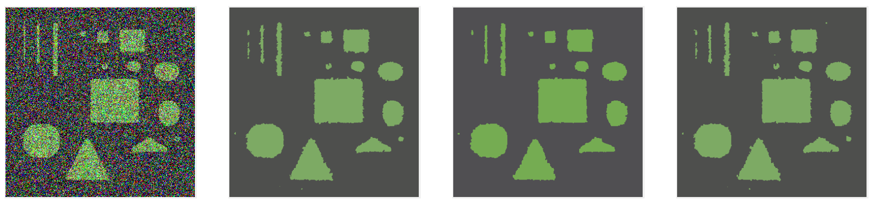
(g) Convex Potts DICE: 0.9101

(h) SaT-Potts DICE: 0.9234

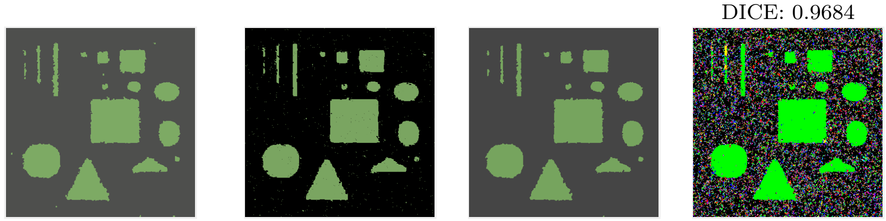
Figure 8.3: Segmentation results of Figure 8.1a corrupted with average blur followed by 50% RV noise.

Table 8.2: Comparison of the DICE indices and computation times (seconds) between the segmentation methods applied to Figure 8.1b corrupted in four cases. Number in **bold** indicates either the highest DICE index or the fastest time among the segmentation methods for a given corrupted image.

	60% RV		60% SP		Blur and 45% RV		Blur and 45% SP	
	DICE	Time (s)	DICE	Time (s)	DICE	Time (s)	DICE	Time (s)
(Original) SLaT	0.9809	5.66	0.9683	6.85	0.9815	9.91	0.9744	11.22
TV^p SLaT	0.9818	**2.77**	0.9709	3.42	0.9828	6.63	0.9760	5.38
AITV SLaT (ADMM)	0.9827	2.82	0.9684	**3.03**	**0.9867**	8.62	**0.9776**	5.91
AITV SLaT (DCA)	0.9831	26.44	0.9684	25.56	0.9853	65.57	0.9771	55.79
AITV CV	**0.9893**	58.49	**0.9807**	77.73	0.9790	77.54	0.9707	81.09
Convex Potts	0.9818	5.34	0.9735	4.98	0.9723	5.12	0.9678	**4.95**
SaT-Potts	0.9806	4.75	0.6997	4.93	0.9753	**4.66**	0.8003	7.36

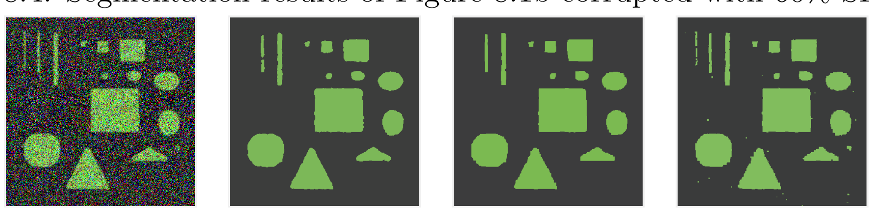


(a) SP noise
(b) (original) SLaT DICE: 0.9683
(c) TV$^{1/2}$ SLaT DICE: 0.9709
(d) AITV SLaT (ADMM) DICE: 0.9684



(e) AITV SLaT (DCA) DICE: 0.9684
(f) AITV CV DICE: 0.9807
(g) Convex Potts DICE: 0.9735
(h) SaT-Potts DICE: 0.6997

Figure 8.4: Segmentation results of Figure 8.1b corrupted with 60% SP noise.



(a) Motion blur with SP noise
(b) (original) SLaT DICE: 0.9744
(c) TV$^{2/3}$ SLaT DICE: 0.9760
(d) AITV SLaT (ADMM) DICE: 0.9776

(e) AITV SLaT (DCA) DICE: 0.9771
(f) AITV CV DICE: 0.9707
(g) Convex Potts DICE: 0.9678
(h) SaT-Potts DICE: 0.8003

Figure 8.5: Segmentation results of Figure 8.1b corrupted with motion blur followed by 45% SP noise.

(a) (b) (c) (d)

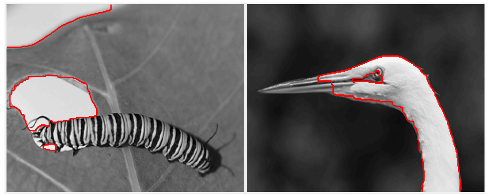
Figure 8.6: Real, grayscale images for image segmentation. (a) Caterpillar. Size: 200×300. (b) Egret. Size: 200×300. (c) Swan. Size: 225×300. (d) Leaf. Size: 203×300.

leftmost rectangle in the top left corner and the two smallest circles above the middle square of Figure 8.2, and the small circular region above the left side of the square in the middle of Figure 8.3. These regions are enclosed in red boxes.

Synthetic Color Images

The (original) color image, Figure 8.1b, is corrupted by either 60% impulsive noise or motion blur followed by 45% noise. Table 8.2 records the DICE indices and the computational times of various segmentation methods applied on all the four cases. For the noisy images of Figure 8.1b, AITV SLaT (ADMM) attains comparable DICE indices to the best AITV CV method but with much less computation time. For the blurry, noisy inputs, AITV SLaT (ADMM) attains the highest DICE indices. In general, AITV SLaT (ADMM) gives satisfactory segmentation results under a reasonable amount of time, compared to others; especially it is much faster than its DCA counterpart.

Figures 8.4-8.5 illustrate the visual results under the SP noise cases. In Figure 8.4, four methods (the original SLaT, the AITV SLaT (ADMM), the AITV SLaT (DCA), and the AITV CV) identify most of the three rectangles in the upper left corner, compared to the other competing methods. Although AITV CV has the highest DICE index, its segmentation result is slightly noisier upon closer inspection. In Figure 8.5, AITV SLaT (ADMM) is able to preserve the three rectangular bars, while the other methods can only segment two bars.



(a) (b)

Figure 8.7: AITV SaT results on real grayscale images.

8.3.2 Real Grayscale Images with Intensity Inhomogeneities

We examine real images with intensity inhomogeneities [5], as shown Figure 8.6. Intensity inhomogeneities can be problematic for image segmentation because of the dramatically varying pixel intensities in local regions of an image. For example, we apply AITV SaT (ADMM) to Figures 8.6a-8.6b to exemplify the challenges of segmenting the object of interest. In Figure 8.7a, no part of the caterpillar is segmented while in Figure 8.7b, most of the egret's beak is not segmented.

Following the work of [124], we incorporate an intensity inhomogeneity (IIH) image, appended as an additional channel of the original image to facilitate segmentation. To generate the IIH image, one calculates an IIH-indicator D

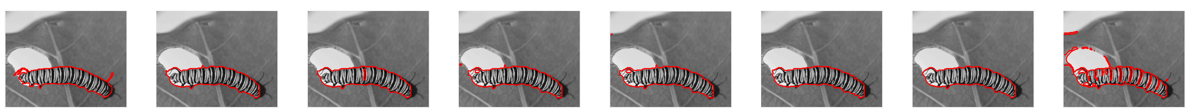
$$D = \frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} \left(\frac{1}{|\Omega_{(i,j)}|} \sum_{(i',j')\in\Omega_{(i,j)}} |u_{i',j'} - \bar{u}_{i,j}|^2 \right),$$

where $\Omega_{(i,j)}$ is a neighborhood centered at pixel (i,j) and $\bar{u}_{i,j}$ is the average pixel intensity in the neighborhood $\Omega_{(i,j)}$. Using the IIH-indicator D, the IIH-image is calculated by

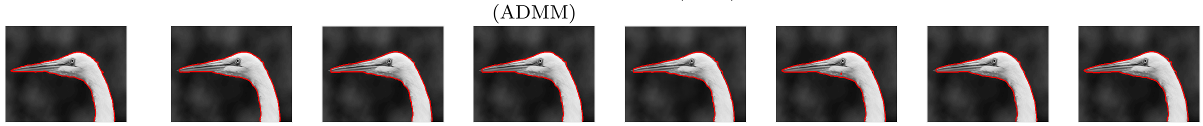
$$u^{\text{IIH}}_{i,j} = \frac{1}{|\Omega_{(i,j)}|} \sum_{(i',j')\in\Omega_{(i,j)}} \mathbb{1}_{\Omega_{(i,j)}}(i',j'),$$

Table 8.3: Comparison of the DICE indices and computation times (seconds) between the segmentation methods applied to Figure 8.6. Number in **bold** indicates either the highest DICE index or the fastest time among the segmentation methods for a given image.

	Figure 8.6a		Figure 8.6b		Figure 8.6c		Figure 8.6d	
	DICE	Time (s)	DICE	Time (s)	DICE	Time (s)	DICE	Time (s)
(Original) SaT	0.9268	1.97	0.9863	1.60	0.9768	2.63	0.9468	1.93
TV^p SaT	0.9222	0.64	0.9806	**0.62**	0.9794	1.01	0.9417	0.96
AITV SaT (ADMM)	0.9292	**0.51**	**0.9870**	**0.62**	0.9791	**0.75**	0.9426	**0.60**
AITV SaT (DCA)	0.9284	9.25	0.9836	9.21	0.9769	14.35	**0.9474**	12.44
AITV CV	**0.9307**	34.44	0.9790	22.83	**0.9828**	57.10	0.9397	20.79
Convex Potts	0.9302	1.28	0.9797	1.03	0.9784	2.54	0.4522	1.41
SaT-Potts	0.8463	1.18	0.9771	1.18	0.9805	1.46	0.9384	1.36

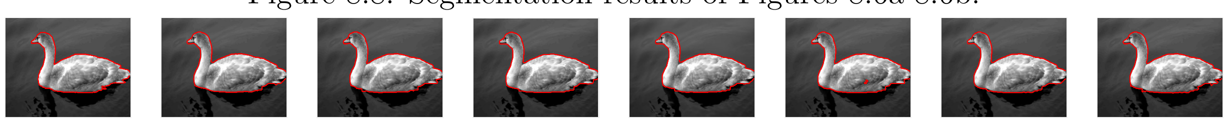


(a) Ground truth. (b) (original) SaT (c) TV$^{1/2}$ SaT (d) AITV SaT (ADMM) (e) AITV SaT (DCA) (f) AITV CV (g) Convex Potts (h) SaT-Potts

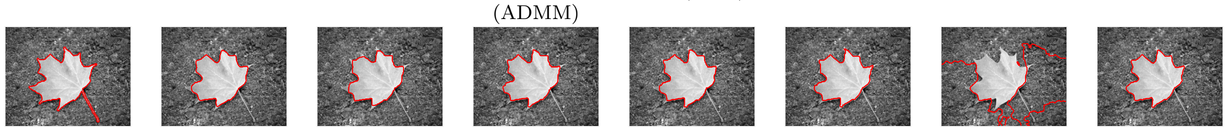


(i) Ground truth. (j) (original) SaT (k) TV$^{2/3}$ SaT (l) AITV SaT (ADMM) (m) AITV SaT (DCA) (n) AITV CV (o) Convex Potts (p) SaT-Potts

Figure 8.8: Segmentation results of Figures 8.6a-8.6b.



(a) Ground truth. (b) (original) SaT (c) TV$^{2/3}$ SaT (d) AITV SaT (ADMM) (e) AITV SaT (DCA) (f) AITV CV (g) Convex Pott (h) SaT-Potts



(i) Ground truth. (j) (original) SaT (k) TV$^{2/3}$ SaT (l) AITV SaT (ADMM) (m) AITV SaT (DCA) (n) AITV CV (o) Convex Potts (p) SaT-Pott

Figure 8.9: Segmentation results of Figures 8.6c-8.6d.

where

$$\mathbb{1}_{\Omega_{(i,j)}}(i',j') = \begin{cases} 1 & \text{if } |\bar{u}_{i,j} - u_{i',j'}|^2 \geq D, \\ 0 & \text{if } |\bar{u}_{i,j} - u_{i',j'}|^2 < D. \end{cases}$$

For our experiments, $\Omega_{(i,j)}$ is a 3×3 patch centered at pixel (i, j).

When the IIH image is added as a channel to the grayscale image, we smooth each channel, followed by the k-means clustering, for the SaT methods. For the other segmentation methods, we consider their multichannel extensions to process the two channels that are composed of grayscale and IIH. We set the parameters $\lambda = 1.75, 1.9, 1.5, 1.25$ and $\mu = 0.45, 0.01, 0.1, 0.1$ for Figures 8.6a-8.6d, respectively, for the SaT methods.

The segmentation results and their ground truths are presented in Figures 8.8-8.9. For each image, the ground truth is determined from the segmentation results by three human subjects. A pixel is declared an object of interest in the ground truth if at least two subjects agree [5]. The DICE indices and computational times of the segmentation algorithms are recorded in Table 8.3. For all the four images, SaT and AITV CV methods can successfully segment the objects of interest. As the fastest method, AITV SaT (ADMM) achieves the highest DICE index for Figure 8.6b, and it is the second best for Figure 8.6a and Figure 8.6c.

8.3.3 Real Color Images

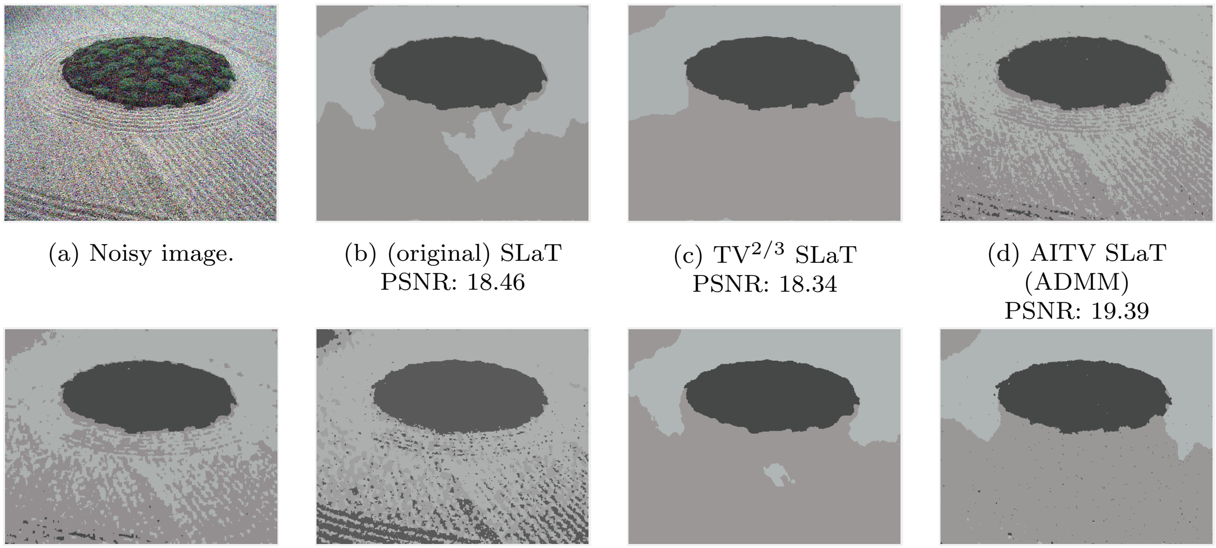
We examine 4 real color images that are provided in [157] for segmentation. We manually add the Gaussian noise of mean zero and variance 0.025 to the clean images as shown in Figure 8.10, aiming to segment Figure 8.10a with $k = 3$ regions, Figure 8.10b with $k = 5$ regions, and Figures 8.10c-8.10d with $k = 6$ regions. Because ground truth is unavailable, we use PSNR to evaluate the segmentation result as a piecewise-constant approximation of the original image. For the SLaT methods, we set the parameters $\lambda = 3.5$ and $\mu = 1.0$ for all the images. In addition, we find that $p = 2/3$ for TV^p and $\alpha = 0.8$ for AITV SLaT and FR give the best PSNR values.

(a) (b) (c) (d)

Figure 8.10: Real color images for image segmentation. (a) Garden. Size: 321×481. (b) Man. Size: 321×481. (c) House. Size: 321×481. (d) Building. Size: 481×321.

Visual segmentation results together with PSNR values are presented in Figures 8.11-8.14. Overall, the AITV SLaT method yields the highest PSNRs and preserves the most details compared to other methods. Specifically in Figure 8.11, AITV SLaT (ADMM), AITV SLaT (DCA), and AITV FR are able to segment the sand lines in fine details, but AITV FR mistakenly identifies the top left corner to be the same group as the middle circular garden. In Figure 8.12, AITV SLaT ADMM and DCA are the best at preserving the man's eyes and palm trees' foliage. In Figure 8.13, the wheel on the right and the windows on the left house are best captured by AITV SLaT ADMM and DCA. For the other methods, the wheel is merged with the grass and the many windowpanes in the left house are absent. Lastly, in Figure 8.14, AITV SLaT ADMM and DCA have a clear advantage in segmenting windows and flowers.

The computational times are recorded in Table 8.4, showing that AITV SLaT (ADMM) is comparable to the original SLaT and nearly 10 times faster than the DCA implementation. It is true that TV^p SaT and SaT-Potts are the fastest methods, but their segmentation results are less satisfactory.



(a) Noisy image.

(b) (original) SLaT
PSNR: 18.46

(c) $\text{TV}^{2/3}$ SLaT
PSNR: 18.34

(d) AITV SLaT (ADMM)
PSNR: 19.39

(e) AITV SLaT (DCA)
PSNR: 19.01

(f) AITV FR
PSNR: 18.51

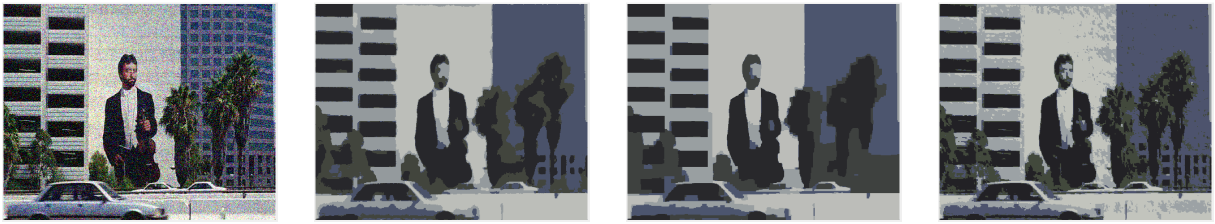
(g) Convex Potts
PSNR: 18.40

(h) SaT-Potts
PSNR: 18.46

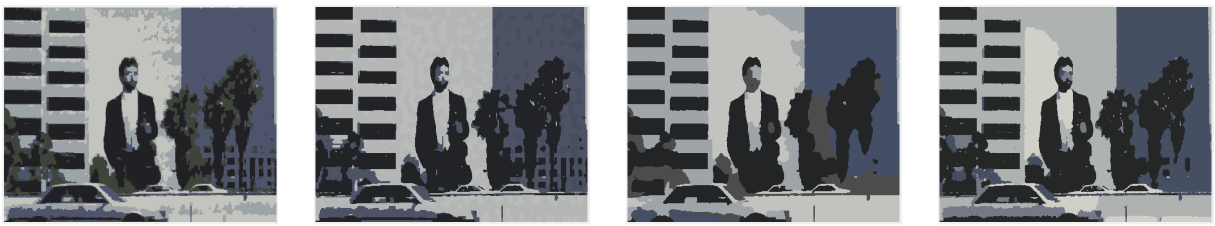
Figure 8.11: Segmentation results into $k = 3$ regions.

Table 8.4: Comparisons of computational times in seconds among the segmentation methods applied to the images in Figure 8.10 corrupted with Gaussian noise with mean zero and variance 0.025.

	garden (Figure 8.10a) $k = 3$	man (Figure 8.10b) $k = 5$	house (Figure 8.10c) $k = 6$	man (Figure 8.10d) $k = 6$
(original) SLaT	10.64	10.63	12.40	14.35
$\text{TV}^{2/3}$ SLaT	6.77	6.81	8.71	10.08
AITV ($\alpha = 0.8$) SLaT (ADMM)	7.10	16.17	12.64	11.95
AITV ($\alpha = 0.8$) SLaT (DCA)	61.90	77.43	74.87	70.08
AITV ($\alpha = 0.8$) FR	124.79	230.32	347.74	395.11
Convex Potts	7.14	47.03	72.64	92.57
SaT-Potts	6.77	6.81	8.71	10.08

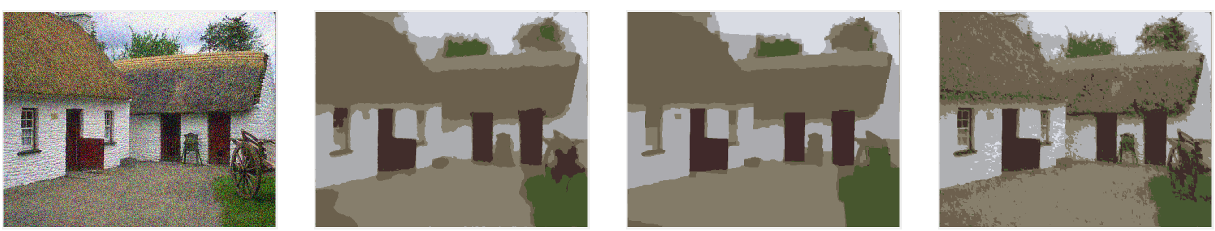


(a) Noisy image. | (b) (original) SLaT PSNR: 19.28 | (c) $\mathrm{TV}^{2/3}$ SLaT PSNR: 19.36 | (d) AITV SLaT (ADMM) PSNR: 20.74

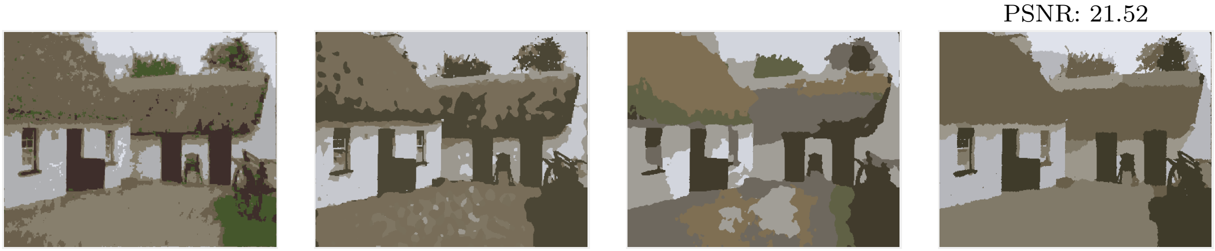


(e) AITV SLaT (DCA) PSNR: 20.65 | (f) AITV FR PSNR: 19.92 | (g) Convex Potts PSNR: 19.50 | (h) SaT-Potts PSNR: 20.12

Figure 8.12: Segmentation results into $k = 5$ regions.

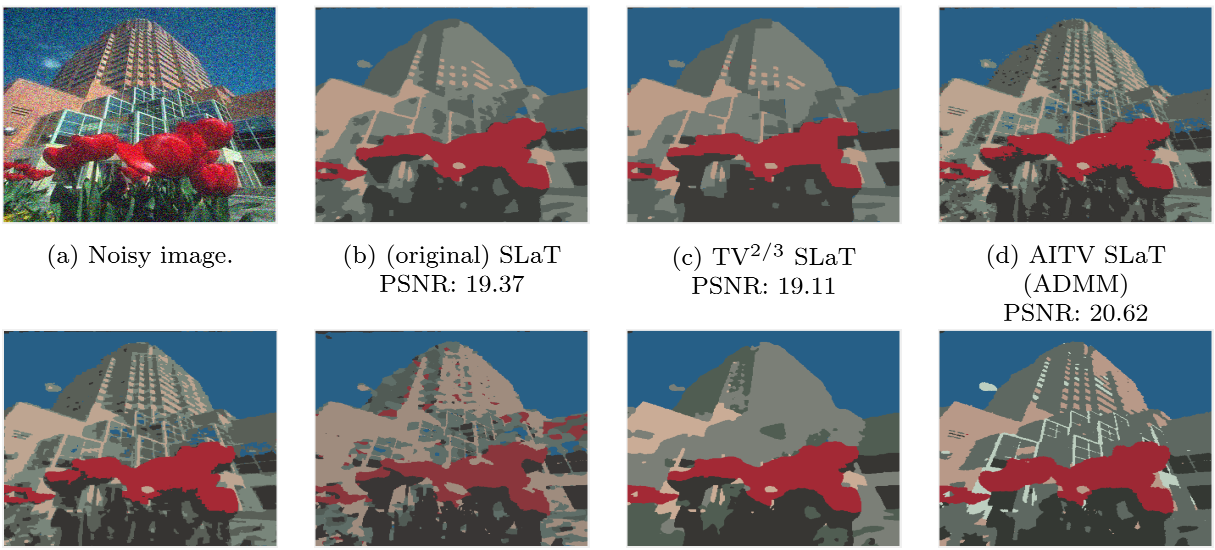


(a) Noisy image. | (b) (original) SLaT PSNR: 20.53 | (c) $\mathrm{TV}^{2/3}$ SLaT PSNR: 20.34 | (d) AITV SLaT (ADMM) PSNR: 21.52



(e) AITV SLaT (DCA) PSNR: 21.25 | (f) AITV FR PSNR: 19.75 | (g) Convex Potts PSNR: 19.48 | (h) SaT-Potts PSNR: 21.00

Figure 8.13: Segmentation results into $k = 6$ regions.



(a) Noisy image.

(b) (original) SLaT
PSNR: 19.37

(c) TV$^{2/3}$ SLaT
PSNR: 19.11

(d) AITV SLaT (ADMM)
PSNR: 20.62

(e) AITV SLaT (DCA)
PSNR: 20.40

(f) AITV FR
PSNR: 18.58

(g) Convex Potts
PSNR: 18.78

(h) SaT-Potts
PSNR: 20.02

Figure 8.14: Segmentation results into $k = 6$ regions.

Chapter 9

Conclusion

In Chapter 7, we proposed AICV and AIFR models for piecewise-constant segmentation that can deal with both grayscale and color images. We developed alternating minimization algorithms utilizing DCA and PDHGLS to efficiently solve the models. Convergence analyses were provided to demonstrate that the objective functions were monotonically decreasing and to validate the efficacy of the algorithms. Numerical results illustrated that the AICV/AIFR models outperform their anisotropic counterparts on various images in a robust manner. The segmentation results are comparable and sometimes better than those of the two-stage segmentation methods.

In Chapter 8, we proposed an efficient ADMM algorithm for the SaT/SLaT framework that utilizes AITV regularization. When designing the ADMM algorithm, we incorporated the proximal operator for the $\ell_1 - \alpha\ell_2$ regularization [138]. We provided convergence analysis of ADMM to demonstrate that the algorithm subsequentially converges to an KKT point under certain conditions. In our numerical experiments, the AITV SaT/SLaT using our ADMM algorithm produces high-quality segmentation results within a few seconds.

The aforementioned chapters demonstrate the effectiveness of using nonconvex regularizations in image processing. As for future works, we will explore other nonconvex regularizations, such as transformed ℓ_1 [246, 247] and ℓ_1/ℓ_2 [182, 208, 207, 206], as alternative options to AITV to other types of segmentation approaches, such as piecewise-smooth formulations [104, 114], the Potts models [179, 194, 212], the fuzzy region model [120], and deep learning techniques [99, 100, 106]. Moreover, the numerical experiments demonstrated that there is no optimal, universal α for all images, which motivates us to develop an automatic method to select α for any given image in the future.

Bibliography

[1] C. C. Aggarwal. *Neural networks and deep learning*. Springer, 2018.

[2] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems*, pages 3177–3186, 2017.

[3] A. Aghasi, A. Abdi, and J. Romberg. Fast convex pruning of deep neural networks. *SIAM Journal on Mathematics of Data Science*, 2(1):158–188, 2020.

[4] M. Ahn, J.-S. Pang, and J. Xin. Difference-of-convex learning: directional stationarity, optimality, and sparsity. *SIAM Journal on Optimization*, 27(3):1637–1665, 2017.

[5] S. Alpert, M. Galun, R. Basri, and A. Brandt. Image segmentation by probabilistic bottom-up aggregation and cue integration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2007.

[6] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[7] L. Ambrosio and V. M. Tortorelli. Approximation of functional depending on jumps by elliptic functional via t-convergence. *Communications on Pure and Applied Mathematics*, 43(8):999–1036, 1990.

[8] A. Antoniadis and J. Fan. Regularization of wavelet approximations. *Journal of the American Statistical Association*, 96(455):939–967, 2001.

[9] D. Arthur and S. Vassilvitskii. K-means++: the advantages of careful seeding. In *2007 ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, pages 1027–1035, 2007.

[10] H. Attouch, J. Bolte, and B. F. Svaiter. Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized gauss–seidel methods. *Mathematical Programming*, 137(1):91–129, 2013.

[11] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pages 2654–2662, 2014.

[12] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[13] F. R. Bach. Consistency of the group lasso and multiple kernel learning. *Journal of Machine Learning Research*, 9(Jun):1179–1225, 2008.

[14] E. Bae, J. Yuan, and X.-C. Tai. Global minimization for continuous multiphase partitioning problems using a dual approach. *International Journal of Computer Vision*, 92(1):112–129, 2011.

[15] Y. Bai, Y.-X. Wang, and E. Liberty. Proxquant: Quantized neural networks via proximal operators. *arXiv preprint arXiv:1810.00861*, 2018.

[16] C. Bao, B. Dong, L. Hou, Z. Shen, X. Zhang, and X. Zhang. Image restoration by minimizing zero norm of wavelet frame coefficients. *Inverse Problems*, 32(11):115004, 2016.

[17] L. Bar, T. F. Chan, G. Chung, M. Jung, N. Kiryati, R. Mohieddine, N. Sochen, and L. A. Vese. Mumford and Shah model and its applications to image segmentation and image restoration. In *Handbook of Mathematical Methods in Imaging*. Springer, 2011.

[18] A. Beck. *First-Order Methods in Optimization*. SIAM, 2017.

[19] H. Birkholz. A unifying approach to isotropic and anisotropic total variation denoising models. *Journal of Computational and Applied Mathematics*, 235(8):2502–2514, 2011.

[20] J. Bolte, S. Sabach, and M. Teboulle. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, 146(1):459–494, 2014.

[21] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[22] P. Breheny and J. Huang. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *The Annals of Applied Statistics*, 5(1):232, 2011.

[23] E. S. Brown, T. F. Chan, and X. Bresson. Completely convex formulation of the Chan–Vese image segmentation model. *International Journal of Computer Vision*, 98(1):103–121, 2012.

[24] K. Bui, F. Park, Y. Lou, and J. Xin. A weighted difference of anisotropic and isotropic total variation for relaxed Mumford–Shah color and multiphase image segmentation. *SIAM Journal on Imaging Sciences*, 14(3):1078–1113, 2021.

[25] K. Bui, F. Park, S. Zhang, Y. Qi, and J. Xin. Nonconvex regularization for network slimming: Compressing CNNs even more. In *International Symposium on Visual Computing*, pages 39–53. Springer, 2020.

[26] K. Bui, F. Park, S. Zhang, Y. Qi, and J. Xin. Improving network slimming with nonconvex regularization. *IEEE Access*, 9:115292–115314, 2021.

[27] K. Bui, F. Park, S. Zhang, Y. Qi, and J. Xin. Structured sparsity of convolutional neural networks via nonconvex sparse group regularization. *Frontiers in Applied Mathematics and Statistics*, 2021.

[28] X. Cai, R. Chan, M. Nikolova, and T. Zeng. A three-stage approach for segmenting degraded color images: Smoothing, lifting and thresholding (SLaT). *Journal of Scientific Computing*, 72(3):1313–1332, 2017.

[29] X. Cai, R. Chan, and T. Zeng. A two-stage image segmentation method using a convex variant of the Mumford–Shah model and thresholding. *SIAM Journal on Imaging Sciences*, 6(1):368–390, 2013.

[30] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37, 2011.

[31] E. J. Candès, 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.

[32] E. J. Candès, 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.

[33] W. Cao, J. Sun, and Z. Xu. Fast image deconvolution using closed-form thresholding formulas of $L_q(q = 1/2, 2/3)$ regularization. *Journal of Visual Communication and Image Representation*, 24(1):31–41, 2013.

[34] A. Chambolle. Finite-differences discretizations of the Mumford-Shah functional. *ESAIM: Mathematical Modelling and Numerical Analysis*, 33(2):261–288, 1999.

[35] A. Chambolle, V. Caselles, D. Cremers, M. Novaga, and T. Pock. An introduction to total variation for image analysis. *Theoretical foundations and numerical methods for sparse recovery*, 9(263-340):227, 2010.

[36] A. Chambolle, D. Cremers, and T. Pock. A convex approach to minimal partitions. *SIAM Journal on Imaging Sciences*, 5(4):1113–1158, 2012.

[37] A. Chambolle and G. Dal Maso. Discrete approximation of the Mumford-Shah functional in dimension two. *ESAIM: Mathematical Modelling and Numerical Analysis*, 33(4):651–672, 1999.

[38] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

[39] R. Chan, H. Yang, and T. Zeng. A two-stage image segmentation method for blurry images with Poisson or multiplicative Gamma noise. *SIAM Journal on Imaging Sciences*, 7(1):98–127, 2014.

[40] R. H. Chan, T. F. Chan, L. Shen, and Z. Shen. Wavelet algorithms for high-resolution image reconstruction. *SIAM Journal on Scientific Computing*, 24(4):1408–1432, 2003.

[41] R. H. Chan and M. K. Ng. Conjugate gradient methods for Toeplitz systems. *SIAM Review*, 38(3):427–482, 1996.

[42] T. F. Chan, S. Esedoglu, and M. Nikolova. Algorithms for finding global minimizers of image segmentation and denoising models. *SIAM Journal on Applied Mathematics*, 66(5):1632–1648, 2006.

[43] T. F. Chan, B. Y. Sandberg, and L. A. Vese. Active contours without edges for vector-valued images. *Journal of Visual Communication and Image Representation*, 11(2):130–141, 2000.

[44] T. F. Chan and L. A. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, 10(2):266–277, 2001.

[45] H. Chang, Y. Lou, Y. Duan, and S. Marchesini. Total variation–based phase retrieval for Poisson noise removal. *SIAM Journal on Imaging Sciences*, 11(1):24–55, 2018.

[46] H. Chang, Y. Lou, M. K. Ng, and T. Zeng. Phase retrieval from incomplete magnitude information via total variation regularization. *SIAM Journal on Scientific Computing*, 38(6):A3672–A3695, 2016.

[47] S. Changpinyo, M. Sandler, and A. Zhmoginov. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.

[48] R. Chartrand. Exact reconstruction of sparse signals via nonconvex minimization. *IEEE Signal Processing Letters*, 14(10):707–710, 2007.

[49] R. Chartrand and V. Staneva. Restricted isometry properties and nonconvex compressive sensing. *Inverse Problems*, 24(3):035020, 2008.

[50] R. Chartrand and W. Yin. Iteratively reweighted algorithms for compressive sensing. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3869–3872. IEEE, 2008.

[51] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2017.

[52] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[53] R. Choksi, Y. G. Gennip, and A. Oberman. Anisotropic total variation regularized L^1 approximation and denoising/deblurring of 2D bar codes. *Inverse Problems & Imaging*, 5:591–617, 2011.

[54] F. Clarke. *Functional analysis, calculus of variations and optimal control*, volume 264. Springer Science & Business Media, Heidelberg, 2013.

[55] A. Cohen, W. Dahmen, and R. DeVore. Compressed sensing and best k-term approximation. *Journal of the American mathematical society*, 22(1):211–231, 2009.

[56] L. Condat. Discrete total variation: New definition and minimization. *SIAM Journal on Imaging Sciences*, 10(3):1258–1290, 2017.

[57] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.

[58] W. Deng and W. Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, 66(3):889–916, 2016.

[59] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[60] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[61] L. Ding and W. Han. $\alpha\ell_1 - \beta\ell_2$ regularization for sparse recovery. *Inverse Problems*, 35(12):125009, 2019.

[62] T. Dinh and J. Xin. Convergence of a relaxed variable splitting method for learning sparse neural networks via ℓ_1,ℓ_0, and transformed-ℓ_1 penalties. In *Proceedings of SAI Intelligent Systems Conference*, pages 360–374. Springer, 2020.

[63] B. Dong and Y. Zhang. An efficient algorithm for ℓ_0 minimization in wavelet frame based image restoration. *Journal of Scientific Computing*, 54(2-3):350–368, 2013.

[64] 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.

[65] S. Esedoglu and S. J. Osher. Decomposition of images by the anisotropic Rudin–Osher–Fatemi model. *Communications on Pure and Applied Mathematics*, 57(12):1609–1626, 2004.

[66] S. Esedoglu and Y.-H. R. Tsai. Threshold dynamics for the piecewise constant Mumford–Shah functional. *Journal of Computational Physics*, 211(1):367–384, 2006.

[67] E. Esser, Y. Lou, and J. Xin. A method for finding structured sparse solutions to nonnegative least squares problems with applications. *SIAM Journal on Imaging Sciences*, 6(4):2010–2046, 2013.

[68] E. Esser, X. Zhang, and T. F. Chan. A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. *SIAM Journal on Imaging Sciences*, 3(4):1015–1046, 2010.

[69] J. Fan and R. Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360, 2001.

[70] J. Fan, H. Peng, et al. Nonconcave penalized likelihood with a diverging number of parameters. *The Annals of Statistics*, 32(3):928–961, 2004.

[71] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Springer, 2013.

[72] Y. Fu, C. Liu, D. Li, Z. Zhong, X. Sun, J. Zeng, and Y. Yao. Exploring structural sparsity of deep networks via inverse scale spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[73] D. Gabay. Chapter ix applications of the method of multipliers to variational inequalities. In *Studies in Mathematics and its Applications*, volume 15, pages 299–331. Elsevier, 1983.

[74] P. Getreuer. Chan–Vese segmentation. *Image Processing On Line*, 2:214–224, 2012.

[75] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[76] M. Gobbino. Finite difference approximation of the Mumford-Shah functional. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 51(2):197–228, 1998.

[77] T. Goldstein, X. Bresson, and S. Osher. Geometric applications of the split Bregman method: segmentation and surface reconstruction. *Journal of Scientific Computing*, 45(1-3):272–293, 2010.

[78] T. Goldstein and S. Osher. The split Bregman method for L1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.

[79] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *International Conference on Machine Learning*, pages 1319–1327. PMLR, 2013.

[80] G. Gu, S. Jiang, and J. Yang. A TVSCAD approach for image deblurring with impulsive noise. *Inverse Problems*, 33(12):125008, 2017.

[81] S. Gu, Q. Xie, D. Meng, W. Zuo, X. Feng, and L. Zhang. Weighted nuclear norm minimization and its applications to low level vision. *International Journal of Computer Vision*, 121(2):183–208, 2017.

[82] R. Gupta and T. Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017.

[83] D. Han, D. Sun, and L. Zhang. Linear rate convergence of the alternating direction method of multipliers for convex composite programming. *Mathematics of Operations Research*, 43(2):622–637, 2018.

[84] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[85] A. Hantoute, M. A. López, and C. Zălinescu. Subdifferential calculus rules in convex analysis: a unifying approach via pointwise supremum functions. *SIAM Journal on Optimization*, 19(2):863–882, 2008.

[86] J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 28(1):100–108, 1979.

[87] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media, 2009.

[88] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1026–1034, 2015.

[89] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[90] W. He, M. Wu, M. Liang, and S.-K. Lam. CAP: Context-aware pruning for semantic segmentation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 960–969, 2021.

[91] Z. Hou. A review on MR image intensity inhomogeneity correction. *International Journal of Biomedical Imaging*, 2006, 2006.

[92] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[93] G. Huang, Z. Liu, G. Pleiss, L. Van Der Maaten, and K. Weinberger. Convolutional networks with dense connectivity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[94] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4700–4708, 2017.

[95] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016.

[96] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[97] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[98] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[99] F. Jia, J. Liu, and X.-C. Tai. A regularized convolutional neural network for semantic image segmentation. *Analysis and Applications*, 19(01):147–165, 2021.

[100] F. Jia, X.-C. Tai, and J. Liu. Nonlocal regularized CNN for image segmentation. *Inverse Problems & Imaging*, 14(5):891, 2020.

[101] 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 *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[102] Z.-F. Jin, Z. Wan, Y. Jiao, and X. Lu. An alternating direction method with continuation for nonconvex low rank minimization. *Journal of Scientific Computing*, 66(2):849–869, 2016.

[103] H. Jung, J. C. Ye, and E. Y. Kim. Improved k–t blast and k–t sense using focuss. *Physics in Medicine & Biology*, 52(11):3201, 2007.

[104] M. Jung. Piecewise-smooth image segmentation models with L^1 data-fidelity terms. *Journal of Scientific Computing*, 70(3):1229–1261, 2017.

[105] M. Jung, M. Kang, and M. Kang. Variational image segmentation models involving non-smooth data-fidelity terms. *Journal of scientific computing*, 59(2):277–308, 2014.

[106] B. Kim and J. C. Ye. Mumford–Shah loss functional for image segmentation with deep learning. *IEEE Transactions on Image Processing*, 29:1856–1866, 2019.

[107] C. Kim and D. Klabjan. A simple and fast algorithm for L1-norm kernel PCA. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

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

[109] D. Krishnan and R. Fergus. Fast image deconvolution using hyper-laplacian priors. In *Advances in Neural Information Processing Systems*, pages 1033–1041, 2009.

[110] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[111] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[112] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[113] M.-J. Lai, Y. Xu, and W. Yin. Improved iteratively reweighted least squares for unconstrained smoothed ℓ_q minimization. *SIAM Journal on Numerical Analysis*, 51(2):927–957, 2013.

[114] T. M. Le and L. A. Vese. Additive & mutiplicative piecewise-smooth segmentation models in a functional minimization approach. *Contemporary Mathematics*, 445:207–224, 2007.

[115] H. A. Le Thi and T. P. Dinh. DC programming and DCA: thirty years of developments. *Mathematical Programming*, 169(1):5–68, 2018.

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

[117] J. Lellmann, J. Kappes, J. Yuan, F. Becker, and C. Schnörr. Convex multi-class image labeling by simplex-constrained total variation. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 150–162. Springer, 2009.

[118] C. Li, C.-Y. Kao, J. C. Gore, and Z. Ding. Minimization of region-scalable fitting energy for image segmentation. *IEEE Transactions on Image Processing*, 17(10):1940–1949, 2008.

[119] F. Li, M. K. Ng, T. Y. Zeng, and C. Shen. A multiphase image segmentation method based on fuzzy region competition. *SIAM Journal on Imaging Sciences*, 3(3):277–299, 2010.

[120] F. Li, S. Osher, J. Qin, and M. Yan. A multiphase image segmentation based on fuzzy membership functions and L1-norm fidelity. *Journal of Scientific Computing*, 69(1):82–106, 2016.

[121] F. Li, B. Zhang, and B. Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[122] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[123] P. Li, W. Chen, H. Ge, and M. K. Ng. $\ell_1 - \alpha\ell_2$ minimization methods for signal and image reconstruction with impulsive noise removal. *Inverse Problems*, 36(5):055009, 2020.

[124] X. Li, X. Yang, and T. Zeng. A three-stage variational image segmentation framework incorporating intensity inhomogeneity information. *SIAM Journal on Imaging Sciences*, 13(3):1692–1715, 2020.

[125] Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[126] Y. Li, C. Wu, and Y. Duan. The TVp regularized Mumford-Shah model for image labeling and segmentation. *IEEE Transactions on Image Processing*, 29:7061–7075, 2020.

[127] Z. Li, X. Luo, B. Wang, A. L. Bertozzi, and J. Xin. A study on graph-structured recurrent neural networks and sparsification with application to epidemic forecasting. In *World Congress on Global Optimization*, pages 730–739. Springer, 2019.

[128] M. Lim, J. M. Ales, B. R. Cottereau, T. Hastie, and A. M. Norcia. Sparse EEG/MEG source estimation via a group lasso. *PloS one*, 12(6):e0176835, 2017.

[129] D. Lin, V. D. Calhoun, and Y.-P. Wang. Correspondence between fMRI and SNP data by group sparse canonical correlation analysis. *Medical image analysis*, 18(6):891–902, 2014.

[130] D. Lin, J. Zhang, J. Li, V. D. Calhoun, H.-W. Deng, and Y.-P. Wang. Group sparse canonical correlation analysis for genomic data integration. *BMC bioinformatics*, 14(1):1–16, 2013.

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

[132] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1529–1538, 2020.

[133] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann. Towards optimal structured CNN pruning via generative adversarial learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2019.

[134] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2736–2744, 2017.

[135] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[136] Y. Lou, S. H. Kang, S. Soatto, and A. L. Bertozzi. Video stabilization of atmospheric turbulence distortion. *Inverse Problems and Imaging*, 7(3):839–861, 2013.

[137] Y. Lou, S. Osher, and J. Xin. Computational aspects of constrained $L_1 - L_2$ minimization for compressive sensing. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 169–180. Springer, 2015.

[138] Y. Lou and M. Yan. Fast $L_1 - L_2$ minimization via a proximal operator. *Journal of Scientific Computing*, 74(2):767–785, 2018.

[139] Y. Lou, P. Yin, Q. He, and J. Xin. Computing sparse representation in a highly coherent dictionary based on difference of L_1 and L_2. *Journal of Scientific Computing*, 64(1):178–196, 2015.

[140] Y. Lou, P. Yin, and J. Xin. Point source super-resolution via non-convex L_1 based methods. *Journal of Scientific Computing*, 68(3):1082–1100, 2016.

[141] Y. Lou, T. Zeng, S. Osher, and J. Xin. A weighted difference of anisotropic and isotropic total variation model for image processing. *SIAM Journal on Imaging Sciences*, 8(3):1798–1823, 2015.

[142] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. *International Conference on Learning Representations, 2018; CoRR*, abs/1712.01312, 2017.

[143] J. Lu, K. Qiao, X. Li, Z. Lu, and Y. Zou. ℓ_0-minimization methods for image restoration problems based on wavelet frames. *Inverse Problems*, 35(6):064001, 2019.

[144] Z. Lu and Y. Zhang. Sparse approximation via penalty decomposition methods. *SIAM Journal on Optimization*, 23(4):2448–2478, 2013.

[145] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5058–5066, 2017.

[146] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, and W. Lin. Thinet: pruning CNN filters for a thinner net. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(10):2525–2538, 2018.

[147] Q.-T. Luong. Color in computer vision. In *Handbook of Pattern Recognition and Computer Vision*, pages 311–368. World Scientific, 1993.

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

[149] J. Lv, Y. Fan, et al. A unified approach to model selection and sparse recovery using regularized least squares. *The Annals of Statistics*, 37(6A):3498–3528, 2009.

[150] J. Lyu, S. Zhang, Y. Qi, and J. Xin. Autoshufflenet: Learning permutation matrices via an exact lipschitz continuous penalty in deep convolutional neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 608–616, New York, NY, USA, 2020. Association for Computing Machinery.

[151] H. Ma, T. Celik, and H.-C. Li. Lightweight attention convolutional neural network through network slimming for robust facial expression recognition. *Signal, Image and Video Processing*, pages 1–9, 2021.

[152] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, pages 122–138, Cham, 2018. Springer International Publishing.

[153] R. Ma, J. Miao, L. Niu, and P. Zhang. Transformed ℓ_1 regularization for learning sparse deep neural networks. *Neural Networks*, 119:286–298, 2019.

[154] S. Ma, X. Song, and J. Huang. Supervised group lasso with applications to microarray data analysis. *BMC bioinformatics*, 8(1):60, 2007.

[155] T.-H. Ma, Y. Lou, T.-Z. Huang, and X.-L. Zhao. Group-based truncated L_{1-2} model for image inpainting. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2079–2083. IEEE, 2017.

[156] Y. Malitsky and T. Pock. A first-order primal-dual algorithm with linesearch. *SIAM Journal on Optimization*, 28(1):411–432, 2018.

[157] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 416–423. IEEE, 2001.

[158] A. Mehranian, H. S. Rad, A. Rahmim, M. R. Ay, and H. Zaidi. Smoothly Clipped Absolute Deviation (SCAD) regularization for compressed sensing MRI using an augmented Lagrangian scheme. *Magnetic Resonance Imaging*, 31(8):1399–1411, 2013.

[159] L. Meier, S. Van De Geer, and P. Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

[160] F. Meng, H. Cheng, K. Li, H. Luo, X. Guo, G. Lu, and X. Sun. Pruning filter in filter. *Advances in Neural Information Processing Systems*, 33:17629–17640, 2020.

[161] B. Merriman, J. K. Bence, and S. J. Osher. Motion of multiple junctions: A level set approach. *Journal of Computational Physics*, 112(2):334–363, 1994.

[162] C. Miao and H. Yu. A general-thresholding solution for $l_p(0 < p < 1)$ regularized CT reconstruction. *IEEE Transactions on Image Processing*, 24(12):5455–5468, 2015.

[163] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42(5):577–685, 1989.

[164] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[165] M. K. Ng, R. H. Chan, and W.-C. Tang. A fast algorithm for deblurring models with Neumann boundary conditions. *SIAM Journal on Scientific Computing*, 21(3):851–866, 1999.

[166] F. Nie, H. Wang, H. Huang, and C. Ding. Unsupervised and semi-supervised learning via ℓ_1-norm graph. In *2011 International Conference on Computer Vision*. IEEE, nov 2011.

[167] M. Nikolova. Local strong homogeneity of a regularized estimator. *SIAM Journal on Applied Mathematics*, 61(2):633–658, 2000.

[168] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Science & Business Media, 2006.

[169] D. Noll. Convergence of non-smooth descent methods using the Kurdyka–Łojasiewicz inequality. *Journal of Optimization Theory and Applications*, 160(2):553–572, 2014.

[170] M. K. Pandit, R. Naaz, and M. A. Chishti. Learning sparse neural networks using non-convex regularization. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2021.

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

[172] F. Park, Y. Lou, and J. Xin. A weighted difference of anisotropic and isotropic total variation for relaxed Mumford–Shah image segmentation. In *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, sep 2016.

[173] J.-P. Penot. *Calculus Without Derivatives*, volume 266. Springer Science & Business Media, 2012.

[174] T. Pham-Dinh and H. A. Le-Thi. Convex analysis approach to DC programming: Theory, algorithms and applications. *Acta Mathematica Vietnamica*, 22(1):289–355, 1997.

[175] T. Pham-Dinh and H. A. Le-Thi. A DC optimization algorithm for solving the trust-region subproblem. *SIAM Journal on Optimization*, 8(2):476–505, 1998.

[176] T. Pock, A. Chambolle, D. Cremers, and H. Bischof. A convex relaxation approach for computing minimal partitions. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 810–817. IEEE, 2009.

[177] T. Pock, D. Cremers, H. Bischof, and A. Chambolle. An algorithm for minimizing the Mumford-Shah functional. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1133–1140. IEEE, 2009.

[178] T. Pock and S. Sabach. Inertial proximal alternating linearized minimization (iPALM) for nonconvex and nonsmooth problems. *SIAM Journal on Imaging Sciences*, 9(4):1756–1787, 2016.

[179] T. Pock, T. Schoenemann, G. Graber, H. Bischof, and D. Cremers. A convex formulation of continuous multi-label problems. In *European Conference on Computer Vision*, pages 792–805. Springer, 2008.

[180] R. B. Potts. Some generalized order-disorder transformations. In *Mathematical proceedings of the cambridge philosophical society*, volume 48, pages 106–109. Cambridge Univ Press, 1952.

[181] Y. Qian, S. Jia, J. Zhou, and A. Robles-Kelly. Hyperspectral unmixing via $L_{1/2}$ sparsity-constrained nonnegative matrix factorization. *IEEE Transactions on Geoscience and Remote Sensing*, 49(11):4282–4297, 2011.

[182] Y. Rahimi, C. Wang, H. Dong, and Y. Lou. A scale-invariant approach for sparse signal recovery. *SIAM Journal on Scientific Computing*, 41(6):A3649–A3672, 2019.

[183] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[184] R. T. Rockafellar and R. J.-B. Wets. *Variational Analysis*, volume 317. Springer Science & Business Media, 2009.

[185] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[186] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[187] C. Samson, L. Blanc-Féraud, G. Aubert, and J. Zerubia. A level set model for image classification. *International Journal of Computer Vision*, 40(3):187–197, 2000.

[188] F. Santosa and W. W. Symes. Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing*, 7(4):1307–1330, 1986.

[189] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

[190] N. Z. Shor. *Minimization Methods for Non-Differentiable Functions*, volume 3. Springer Science & Business Media, 2012.

[191] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. A sparse-group lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245, 2013.

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

[193] M. Storath and A. Weinmann. Fast partitioning of vector-valued images. *SIAM Journal on Imaging Sciences*, 7(3):1826–1852, 2014.

[194] M. Storath, A. Weinmann, J. Frikel, and M. Unser. Joint image reconstruction and segmentation using the Potts model. *Inverse Problems*, 31(2):025003, 2015.

[195] E. Strekalovskiy and D. Cremers. Real-time minimization of the piecewise smooth Mumford-Shah functional. In *European Conference on Computer Vision*, pages 127–141. Springer, 2014.

[196] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

[197] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[198] C. Tai, T. Xiao, Y. Zhang, X. Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

[199] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[200] H. Tran and C. Webster. A class of null space conditions for sparse recovery via nonconvex, non-separable minimizations. *Results in Applied Mathematics*, 3:100011, 2019.

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

[202] J. Trzasko, A. Manduca, and E. Borisch. Sparse MRI reconstruction via multiscale L_0-continuation. In *2007 IEEE/SP 14th Workshop on Statistical Signal Processing*, pages 176–180. IEEE, August 2007.

[203] R. Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*, volume 47. Cambridge University Press, 2018.

[204] L. A. Vese and T. F. Chan. A multiphase level set framework for image segmentation using the Mumford and Shah model. *International Journal of Computer Vision*, 50(3):271–293, 2002.

[205] M. Vincent and N. R. Hansen. Sparse group lasso and high dimensional multinomial classification. *Computational Statistics & Data Analysis*, 71:771–786, 2014.

[206] C. Wang, M. Tao, C.-N. Chuah, J. Nagy, and Y. Lou. Minimizing L1 over L2 norms on the gradient. *Inverse Problems*, 38(6):065011, 2022.

[207] C. Wang, M. Tao, J. G. Nagy, and Y. Lou. Limited-angle CT reconstruction via the L_1/L_2 minimization. *SIAM Journal on Imaging Sciences*, 14(2):749–777, 2021.

[208] C. Wang, M. Yan, Y. Rahimi, and Y. Lou. Accelerated schemes for the L_1/L_2 minimization. *IEEE Transactions on Signal Processing*, 68:2660–2669, 2020.

[209] L. Wang, G. Chen, and H. Li. Group SCAD regression analysis for microarray time course gene expression data. *Bioinformatics*, 23(12):1486–1494, 2007.

[210] X.-F. Wang, D.-S. Huang, and H. Xu. An efficient local Chan–Vese model for image segmentation. *Pattern Recognition*, 43(3):603–618, 2010.

[211] Y. Wang, J. Yang, W. Yin, and Y. Zhang. A new alternating minimization algorithm for total variation image reconstruction. *SIAM Journal on Imaging Sciences*, 1(3):248–272, 2008.

[212] K. Wei, K. Yin, X.-C. Tai, and T. F. Chan. New region force for variational models in image segmentation and high dimensional data clustering. *Annals of Mathematical Sciences and Applications*, 3(1):255–286, 2018.

[213] F. Wen, L. Chu, P. Liu, and R. C. Qiu. A survey on nonconvex regularization-based sparse and low-rank recovery in signal processing, statistics, and machine learning. *IEEE Access*, 6:69883–69906, 2018.

[214] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, NIPS'16, pages 2074–2082, Red Hook, NY, USA, 2016. Curran Associates Inc.

[215] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li. Coordinating filters for faster deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2017.

[216] T. Wu, J. Shao, X. Gu, M. K. Ng, and T. Zeng. Two-stage image segmentation based on nonconvex $\ell_2 - \ell_p$ approximation and thresholding. *Applied Mathematics and Computation*, 403:126168, 2021.

[217] T. Wu, Y. Zhao, Z. Mao, L. Shi, Z. Li, and Y. Zeng. Image segmentation via Fischer-Burmeister total variation and thresholding. *Advances in Applied Mathematics and Mechanics*, 14(4):960–988, 2022.

[218] Y. Wu and K. He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[219] L. Xu, C. Lu, Y. Xu, and J. Jia. Image smoothing via L_0 gradient minimization. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, pages 1–12, 2011.

[220] Y. Xu, Y. Li, S. Zhang, W. Wen, B. Wang, Y. Qi, Y. Chen, W. Lin, and H. Xiong. Trained rank pruning for efficient deep neural networks. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 14–17. IEEE, 2019.

[221] Y. Xu, Y. Li, S. Zhang, W. Wen, B. Wang, Y. Qi, Y. Chen, W. Lin, and H. Xiong. TRP: Trained rank pruning for efficient deep neural networks. *International Joint Conference on Artificial Intelligence*, 2020.

[222] Z. Xu, X. Chang, F. Xu, and H. Zhang. $L_{1/2}$ regularization: A thresholding representation theory and a fast solver. *IEEE Transactions on Neural Networks and Learning Systems*, 23(7):1013–1027, 2012.

[223] Z. Xu, H. Guo, Y. Wang, and Z. Hai. Representative of $L_{1/2}$ regularization among $L_q(0 \leq q \leq 1)$ regularizations: an experimental study based on phase diagram. *Acta Automatica Sinica*, 38(7):1225–1228, 2012.

[224] F. Xue, Y. Qi, and J. Xin. RARTS: An efficient first-order relaxed architecture search method. *IEEE Access*, 10:65901–65912, 2022.

[225] F. Xue and J. Xin. Learning sparse neural networks via ℓ_0 and Tℓ_1 by a relaxed variable splitting method with application to multi-scale curve classification. In *World Congress on Global Optimization*, pages 800–809. Springer, 2019.

[226] F. Xue and J. Xin. Network compression via cooperative architecture search and distillation. In *IEEE International Conference on AI for Industries*, pages 42–43, 2021.

[227] C. Yang, Y. Akimoto, D. W. Kim, and M. Udell. Oboe: Collaborative filtering for automl model selection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1173–1183. ACM, July 2019.

[228] Q. Ye, H. Zhao, Z. Li, X. Yang, S. Gao, T. Yin, and N. Ye. L1-norm distance minimization-based fast robust twin support vector k-plane clustering. *IEEE Transactions on Neural Networks and Learning Systems*, 29(9):4494–4503, 2017.

[229] P. Yin, E. Esser, and J. Xin. Ratio and difference of ℓ_1 and ℓ_2 norms and sparse representation with coherent dictionaries. *Communications in Information and Systems*, 14(2):87–109, 2014.

[230] P. Yin, Y. Lou, Q. He, and J. Xin. Minimization of ℓ_{1-2} for compressed sensing. *SIAM Journal on Scientific Computing*, 37(1):A536–A563, 2015.

[231] P. Yin, Z. Sun, W.-L. Jin, and J. Xin. ℓ_1-minimization method for link flow correction. *Transportation Research Part B: Methodological*, 104:398–408, 2017.

[232] P. Yin, S. Zhang, J. Lyu, S. Osher, Y. Qi, and J. Xin. Binaryrelax: A relaxation approach for training deep neural networks with quantized weights. *SIAM Journal on Imaging Sciences*, 11(4):2205–2223, 2018.

[233] 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.

[234] J. Yoon and S. J. Hwang. Combined group and exclusive sparsity for deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, volume 70 of *Proceedings of Machine Learning Research*, pages 3958–3966, International Convention Centre, Sydney, Australia, August 2017. PMLR.

[235] J. You, Y. Jiao, X. Lu, and T. Zeng. A nonconvex model with minimax concave penalty for image restoration. *Journal of Scientific Computing*, 78(2):1063–1086, 2019.

[236] J. Yuan, E. Bae, X.-C. Tai, and Y. Boykov. A continuous max-flow approach to Potts model. In *European Conference on Computer Vision*, pages 379–392. Springer, 2010.

[237] J. Yuan, K. Yin, Y.-G. Bai, X.-C. Feng, and X.-C. Tai. Bregman-proximal augmented Lagrangian approach to multiphase image segmentation. In *International Conference on Scale Space and Variational Methods in Computer Vision*, pages 524–534. Springer, 2017.

[238] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[239] X.-T. Yuan, P. Li, and T. Zhang. Gradient hard thresholding pursuit. *Journal of Machine Learning Research*, 18:166–1, 2017.

[240] C. Zach, D. Gallup, J.-M. Frahm, and M. Niethammer. Fast global labeling for real-time stereo using multiple plane sweeps. In *VMV*, pages 243–252, 2008.

[241] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[242] J. Zeng, T. T.-K. Lau, S. Lin, and Y. Yao. Global convergence of block coordinate descent in deep learning. In *International Conference on Machine Learning*, pages 7313–7323. PMLR, 2019.

[243] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[244] C.-H. Zhang. Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, 38(2):894–942, 2010.

[245] J. Zhang, W. Wang, C. Lu, J. Wang, and A. K. Sangaiah. Lightweight deep network for traffic sign classification. *Annals of Telecommunications*, 75(7):369–379, 2020.

[246] S. Zhang and J. Xin. Minimization of transformed l_1 penalty: Closed form representation and iterative thresholding algorithms. *Communications in Mathematical Sciences*, 15(2):511 – 537, 2017.

[247] S. Zhang and J. Xin. Minimization of transformed l_1 penalty: theory, difference of convex function algorithm, and robust application in compressed sensing. *Mathematical Programming*, 169(1):307–336, 2018.

[248] S. Zhang, P. Yin, and J. Xin. Transformed Schatten-1 iterative thresholding algorithms for low rank matrix completion. *Communications in Mathematical Sciences*, 15(3):839 – 862, 2017.

[249] X. Zhang, Y. Lu, and T. Chan. A novel sparsity reconstruction method from Poisson data for 3D bioluminescence tomography. *Journal of Scientific Computing*, 50(3):519–535, 2012.

[250] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6848–6856. IEEE, 2018.

[251] Y. Zhang, B. Dong, and Z. Lu. ℓ_0 minimization for wavelet frame based image restoration. *Mathematics of Computation*, 82(282):995–1015, 2013.

[252] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian. Variational convolutional neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2780–2789, 2019.

[253] H. Zhou, M. E. Sehl, J. S. Sinsheimer, and K. Lange. Association screening of common and rare genetic variants by penalized regression. *Bioinformatics*, 26(19):2375, 2010.

[254] Y. Zhou, R. Jin, and S. C.-H. Hoi. Exclusive lasso for multi-task feature selection. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 988–995, Chia Laguna Resort, Sardinia, Italy, May 2010. JMLR Workshop and Conference Proceedings.

[255] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[256] M. Zhu and T. Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. *UCLA CAM Report*, 34, 2008.