Streamlining Convolutional Neural Networks: Channel-Level Sparsity for Efficient Compression

Project-II (CS47006) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Nikhil Saraswat

(20CS10039)

Under the supervision of

Professor Pabitra Mitra

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Spring Semester, 2023-24

May 3, 2024

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: May 3, 2024
Place: Kharagpur

(Nikhil Saraswat)
(20CS10039)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Streamlining Convolutional Neural Networks: Channel-Level Sparsity for Efficient Compression**" submitted by **Nikhil Saraswat** (Roll No. 20CS10039) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering is a record of bona fide work carried out by him under my supervision and guidance during Spring Semester, 2023-24.

Pabitra Mitra

Professor Pabitra Mitra
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Date: May 3, 2024

Place: Kharagpur

Abstract

Name of the student: **Nikhil Saraswat** Roll No: **20CS10039**
Degree for which submitted: **Bachelor of Technology**
Department: **Department of Computer Science and Engineering**
Thesis title: **Streamlining Convolutional Neural Networks: Channel-Level Sparsity for Efficient Compression**
Thesis supervisor: **Professor Pabitra Mitra**
Month and year of thesis submission: **May 3, 2024**

Streamlining Convolutional Neural Networks: Channel-Level Sparsity for Efficient Compression

Deep convolutional neural networks (CNNs) have revolutionized various computer vision tasks but are often limited in deployment due to their high computational demands. Addressing this challenge, we present a good learning scheme aimed at reducing computing operations, model size, and runtime memory footprint while maintaining accuracy. Our approach introduces channel-level sparsity into modern CNN architectures in a simple yet highly effective manner, significantly enhancing efficiency without compromising performance.

Unlike many existing methods, our proposed approach seamlessly integrates into popular CNN architectures, incurring minimal training overhead and requiring no specialized hardware or software accelerators. By identifying and pruning insignificant channels during training, our method transforms wide and large networks into thin and compact models with comparable accuracy. Empirical evaluations across several state-of-the-art CNN models, including VGGNet on the CIFAR dataset, showcase remarkable reductions in model size and computing operations—up to 10x and 3x, respectively.

Moreover, we advance the state-of-the-art in sparsity regularization by replacing the conventional L1 penalty with nonconvex penalties such as Lp and transformed L1 (TL1). Through numerical experiments on VGGNet and Densenet trained on CIFAR-10, we demonstrate the superior compression capabilities of nonconvex penalties, with TL1 exhibiting exceptional accuracy preservation post-channel pruning. Notably, L1/2,3/4 penalties yield compressed models with accuracy levels comparable to L1 after retraining, underscoring the versatility and efficacy of our proposed approach in CNN compression.

Keywords: Convolutional neural networks, Sparse optimization, L1 regularization, Lp regularization, TL1 regularization, Batch normalization, Channel pruning, Non-convex optimization, Model compression

Acknowledgements

I express my sincere gratitude to my supervisor, Professor Pabitra Mitra, for granting me the opportunity to work under his mentorship. Throughout my B.Tech project, he provided insightful guidance and constructive feedback. He was consistently supportive and understanding.

I am grateful for the chance to have worked on this project and to have learned from the people I have met along the way. This project has been challenging but rewarding, and I am proud of what I have achieved. I would also like to extend my warmest gratitude to my family and friends at IIT Kharagpur for their unwavering support and motivation throughout my journey.

Contents

List of Figures

List of Tables

Abbreviations

CNN	**C**onvolutional **N**eural **N**etwork
BN	**B**atch **N**ormalization

Symbols

$\oplus$	Tensor Product
λ	Scaling factor
ψ	Eigen Vector
μ	Mean
σ	Standard Deviation
Σ	Summation

Chapter 1

Introduction

1.1 Introduction

In the ever-evolving landscape of computer vision, Convolutional Neural Networks (CNNs) have emerged as the cornerstone for a multitude of tasks, ranging from image classification to semantic segmentation. The trajectory of CNN development has witnessed exponential growth in model complexity, transitioning from seminal architectures like AlexNet (6) and VGGNet (4) to the more intricate ResNets (7) and beyond. These advancements have undeniably bolstered the representation power of CNNs, leading to unprecedented levels of accuracy in various vision tasks.

However, this surge in model sophistication comes at a cost. As CNNs burgeon in size and depth, they become increasingly resource-intensive, posing significant challenges for deployment in resource-constrained environments such as mobile devices, wearables, and Internet of Things (IoT) devices. The crux of the issue lies in the resource burden imposed by large CNN models, both in terms of storage and computational requirements during inference.

1.2 Problem Formulation

The deployment of CNNs in real-world applications is encumbered by several intrinsic challenges:

- **Model Size:** The proliferation of parameters in modern CNN architectures necessitates significant storage capacity, posing a barrier to deployment on devices with limited memory resources.
- **Run-time Memory:** During inference, the memory requirements of CNNs can exceed the storage demands of model parameters, particularly when processing high-resolution images. This presents a bottleneck for applications operating under stringent memory constraints.
- **Computational Complexity:** The computational overhead incurred by convolutional operations, exacerbated by the scale and depth of contemporary CNN architectures, impedes real-time inference on low-power devices, thereby limiting their practical utility.

1.3 Motivation

In response to these challenges, a plethora of techniques have been proposed to alleviate the resource constraints associated with large CNN deployment. These range from model compression methods like low-rank approximation (8) and network quantization (9) to sparsity-inducing techniques.

By imposing L_1 regularization on the scaling factors within batch normalization layers, which facilitates the identification and pruning of insignificant channels, thereby reducing the model size and computational overhead. However, despite its efficacy, the reliance on L_1 regularization presents limitations in achieving optimal sparsity and preserving model accuracy.

Motivated by the imperative to further enhance the efficiency of large CNNs under resource constraints, we can try innovative refinement by leveraging nonconvex regularization techniques, specifically L_p and transformed L_1 (TL_1) regularizers, we aim to achieve superior sparsity induction while preserving model fidelity. By exploring the intersection of convex and nonconvex optimization principles, we endeavor to push the boundaries of efficiency in CNN deployment, unlocking new avenues for practical applications in diverse real-world scenarios.

Chapter 2

Background

In this chapter, we will cover essential ideas in Convolutional Neural Networks with various types of pruning and regularizations. Only the fundamental principles are required to comprehend the content being delivered.

2.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) have emerged as the cornerstone of modern computer vision (10; 11; 12), owing to their ability to automatically learn hierarchical representations from raw input data. CNNs are composed of multiple layers, including convolutional layers, pooling layers, and fully connected layers. Convolutional layers apply filters to input data, extracting features at different spatial locations, while pooling layers down-sample feature maps to reduce computational complexity. Fully connected layers combine extracted features to make predictions.

2.1.1 Evolution of CNN Architectures

The evolution of CNN architectures has been marked by a continuous quest for improved performance and efficiency. From the pioneering AlexNet, which introduced deep convolutional layers to win the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, to subsequent architectures like VGGNet, GoogLeNet, and ResNet, each iteration has pushed the boundaries of model complexity and accuracy. Notable advancements include the introduction of skip connections in ResNet to alleviate the vanishing gradient problem and the inception modules in GoogLeNet for efficient feature extraction.

2.1.2 VGGNet

The acronym VGG originates from "Visual Geometry Group," denoting a renowned deep Convolutional Neural Network (CNN) architecture distinguished by its multitude of layers. The term "deep" underscores the extensive layering, exemplified by variants like VGG-16 and VGG-19, which boast 16 and 19 convolutional layers respectively. Initially conceived for object recognition tasks, VGGNet (4) has emerged as a cornerstone in the field, exceeding benchmarks not only in the ImageNet dataset but also across various other tasks and datasets. Its enduring popularity stems from its efficacy and versatility, rendering it one of the most sought-after architectures for image recognition tasks to this day.

2.1.2.1 VGG16

VGG16, also known as the VGG model or VGGNet, is a convolutional neural network architecture proposed by A. Zisserman and K. Simonyan from the University of Oxford, outlined in their seminal paper "Very Deep Convolutional Networks for Large-Scale Image Recognition." Renowned for its 16-layer depth, VGG16 achieved remarkable success, boasting an impressive top-5 test accuracy of nearly 92.7% on the ImageNet dataset, which comprises over 14 million images across nearly 1000 classes. Notably, VGG16 emerged as one of the most prominent models in the ILSVRC-2014 competition. Distinguished by its utilization of multiple 3×3 kernel-sized filters instead of larger ones, VGG16 marked significant advancements over its predecessor, AlexNet. Training the VGG16 model demanded extensive computational resources, with researchers utilizing Nvidia Titan Black GPUs over multiple weeks. Capable of categorizing images into 1000 object classes, including but not limited to keyboards, animals, pencils, and mice, VGG16 operates with an input size of 224-by-224 pixels, cementing its status as a cornerstone in the realm of image recognition.

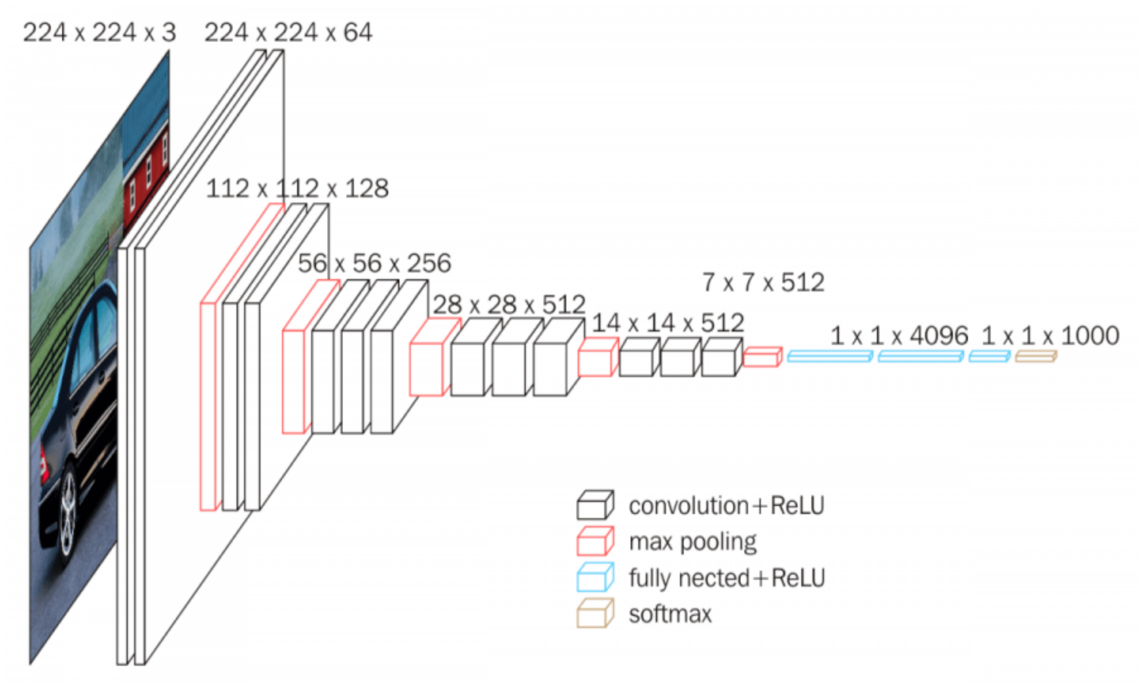

Figure 2.1: Visualization of VGG-16 architecture (2)

2.1.2.2 VGG19

The VGG19 (13) model, also referred to as VGGNet-19, shares the foundational concept of its predecessor, VGG16, with the primary distinction lying in its increased depth, boasting a total of 19 layers. The numerical suffixes "16" and "19" correspond to the respective number of weight layers, namely convolutional layers, within each model. Consequently, VGG19 encompasses three additional convolutional layers compared to VGG16, augmenting its capacity for feature extraction and representation. Further exploration of the distinguishing characteristics and performance nuances between VGG16 and VGG19 will be delved into later in this article.

2.1.2.3 VGG Architecture

The VGG architecture 2.1, rooted in the fundamental principles of convolutional neural networks (CNNs), stands as a pivotal model renowned for its simplicity and effectiveness. Comprising 13 convolutional layers and three fully connected layers, VGG-16 serves as a prime example of this architecture. Operating on image inputs of size 224×224, the VGGNet ensures consistency by cropping out the central 224×224 patch for each image in the ImageNet competition. Notably, VGG's convolutional layers employ compact 3×3 filters, capturing essential spatial features while minimizing computational overhead. Additionally, 1×1 convolution filters facilitate linear transformations of the input, further enriching the network's representational capacity.

The integration of rectified linear unit (ReLU) activations marks a significant departure from previous architectures like AlexNet, enhancing training efficiency by mitigating the vanishing gradient problem. ReLUs introduce non-linearity, crucial for modeling complex relationships within the data. Moreover, VGG prioritizes the preservation of spatial resolution by maintaining a fixed convolution stride of 1 pixel, ensuring the fidelity of spatial information throughout the network's layers. While ReLU activation is pervasive across all hidden layers, VGG eschews Local Response Normalization (LRN), citing its negligible impact on accuracy alongside increased memory consumption and training duration.

Fully connected layers constitute the latter segment of the VGG architecture, culminating in three densely connected layers. The first two fully connected layers boast 4096 channels each, while the final layer comprises 1000 channels, one for each class in the ImageNet dataset. The numerical designation ”16” in VGG16 denotes the network’s depth, underscoring its extensive nature with approximately 138 million parameters. Despite its formidable size, VGGNet’s allure lies in its uniform and straightforward architecture, characterized by alternating convolution and pooling layers. The versatility of VGG is evident in the flexibility afforded by the varying numbers of filters across layers, facilitating feature extraction at multiple scales and complexities, ultimately contributing to its enduring appeal in the realm of image recognition.

2.2 Batch Normalization

Batch normalization (BatchNorm) (14) is a pivotal technique in deep learning, originally introduced to stabilize and accelerate training processes by normalizing the activations of each layer within mini-batches. It operates by computing the mean and variance of activations across the mini-batch dimension and normalizing the activations using these statistics. Additionally, BatchNorm introduces learnable parameters, typically a scale parameter (γ) and a shift parameter (β), to adapt the normalized activations to the desired scale and location.

While BatchNorm’s primary objective is to stabilize training by mitigating issues stemming from internal covariate-shift, it offers several ancillary benefits. Firstly, it acts as a regularizer, reducing the reliance on techniques like dropout by introducing noise during training, which aids in generalization. Moreover, BatchNorm mitigates the sensitivity of deep networks to initialization and hyperparameters, making them more robust and easier to train.

At its core, BatchNorm operates by normalizing the activations of each layer across the mini-batch dimension. Given a mini-batch of size m, containing activations $\{x_1, x_2, ..., x_m\}$ for a particular layer, BatchNorm computes the mean μ and variance σ^2 of these activations:

$$\mu = \frac{1}{m} \sum_{i=1}^{m} x_i$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu)^2$$

Next, BatchNorm normalizes the activations using these statistics:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Here, ϵ is a small constant added for numerical stability. Following normalization, BatchNorm introduces two learnable parameters, typically a scale parameter γ and a shift parameter β, which allow the network to adapt the normalized activations to the desired scale and location:

$$y_i = \gamma \hat{x}_i + \beta$$

These parameters are learned during training through backpropagation, enabling the network to determine the optimal scaling and shifting for each layer. figure 2.2

However, BatchNorm does have limitations, particularly during inference, as its behavior may differ between training and inference due to the reliance on mini-batch statistics for normalization. To address this issue, techniques like running batch normalization statistics during inference or utilizing alternative normalization techniques have been proposed.

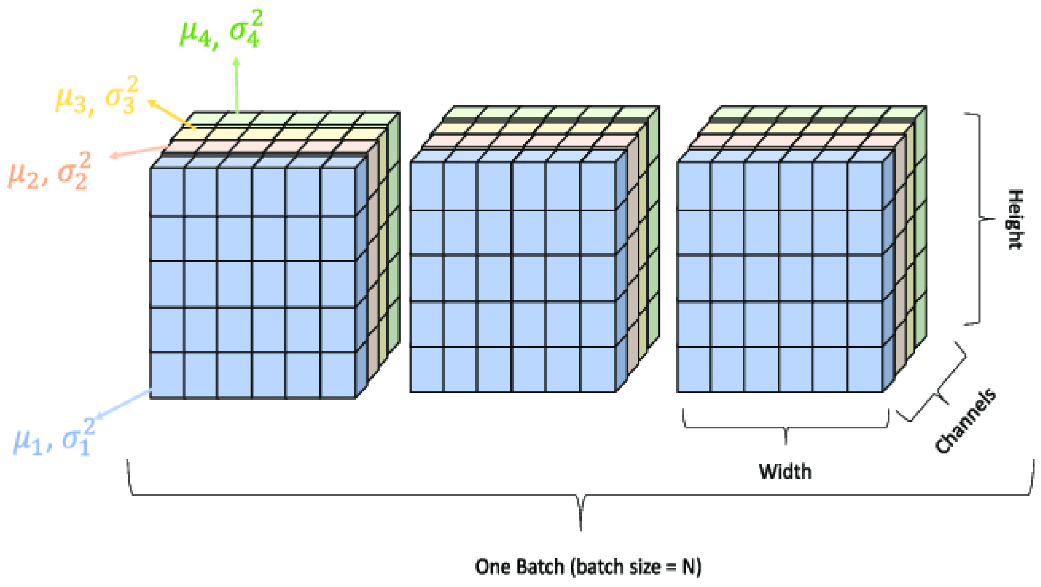

FIGURE 2.2: Visualization of batch normalization on a feature map. The mean and variance of the values of the pixels of the same colors corresponding to the channels are computed and are used to normalize these pixels (1)

One innovative application of BatchNorm is as a channel-wise sparsity regularization layer. By modifying the formulation of BatchNorm to include a sparsity-inducing term, the network can be encouraged to preferentially activate a subset of channels within each layer, leading to increased sparsity and efficiency in representation. This approach promotes the selective activation of channels, enhancing the network's representational capacity while reducing redundancy and computational overhead.

2.3 Pruning Techniques

Pruning techniques are instrumental in reducing the size and computational complexity of neural networks by eliminating redundant connections or parameters. These techniques can be broadly classified into structured and unstructured pruning methodologies.

2.3.1 Structured Pruning

Structured Pruning (15; 16; 17) entails the removal of entire filters, channels, or layers based on predefined criteria. For instance, magnitude-based pruning involves removing filters or channels with small weights, thus eliminating less impactful components. Another approach within structured pruning is sensitivity analysis, where the consequences of removing specific components on network performance are evaluated. By identifying and discarding redundant structures at a higher level, structured pruning offers efficiency gains while preserving the network's overall architecture.

2.3.2 Unstructured Pruning

Unstructured Pruning operates at a more granular level by targeting individual weights or connections based on their importance scores. Magnitude pruning and weight thresholding are common techniques utilized within unstructured pruning methodologies. In magnitude pruning, weights below a certain threshold are pruned, effectively removing connections deemed less significant. Weight thresholding similarly identifies and eliminates less crucial connections, thereby reducing the network's computational burden.

Both structured and unstructured pruning techniques contribute to model optimization by eliminating redundant parameters, ultimately enhancing efficiency without compromising performance. While structured pruning offers the advantage of simplifying the network's architecture, unstructured pruning provides finer granularity in parameter reduction. By leveraging pruning techniques judiciously, practitioners can tailor neural network architectures to strike an optimal balance between model size and computational complexity, facilitating deployment in resource-constrained environments while maintaining high-performance levels.

2.4 Channel Pruning

Channel pruning, often referred to as filter pruning, represents a targeted approach to optimizing convolutional neural networks (CNNs) by selectively removing entire channels (18) or filters from convolutional layers based on their importance. This technique aims to identify and discard channels with low activation or minimal contribution to feature extraction, thereby reducing model size and computational complexity while maintaining performance.

The process of channel pruning leverages the inherent redundancy present in CNNs, exploiting the fact that not all channels contribute equally to the network's representational capacity. By removing redundant channels, channel pruning enables the creation of more compact and efficient models without sacrificing accuracy.

Pruning channels involves evaluating each channel's importance based on various criteria, such as activation levels, gradient magnitudes, or sensitivity analysis. Channels with low activation or gradients during training may indicate reduced importance and can be candidates for pruning. Similarly, sensitivity analysis techniques assess the impact of removing specific channels on network performance to determine their importance.

Once redundant channels are identified, they are pruned from the network, resulting in a streamlined architecture with fewer parameters and reduced computational overhead. Channel pruning can lead to significant compression gains, making it particularly valuable for deploying CNNs in resource-constrained environments such as mobile devices or embedded systems.

Overall, channel pruning offers a targeted and effective approach to model optimization, enabling the creation of more efficient CNNs without compromising performance. By selectively removing redundant channels (19), this technique contributes to the development of compact and deployable deep learning models capable of meeting the demands of real-world applications.

2.5 Regularization techniques

Regularization techniques are vital tools in the training of machine learning models, aimed at preventing over-fitting and improving generalization performance. Among these techniques, L_1, L_p, and TL_1 regularization methods are widely used to penalize the complexity of models by adding regularization terms to the loss function. Let's delve into each of these techniques:

- L_1 **Regularization (Lasso Regression):** L_1 regularization, also known as Lasso Regression, adds a penalty term to the loss function that is proportional to the absolute values of the model's weights. This encourages sparsity in the weight vector by driving some weights to zero, effectively performing feature selection.

  The L_1 regularization term is defined as:

  $$L_1 \text{ regularization term} = \lambda \sum_{i=1}^{n} |w_i|$$

  where λ is the regularization parameter controlling the strength of regularization, w_i are the model weights, and n is the total number of weights.

- L_p **Regularization:** L_p regularization generalizes L_1 regularization by using the L_p norm of the weight vector instead of the absolute values. This allows for different degrees of penalization based on the chosen value of p. When $p = 1$, it reduces to L_1 regularization, and when $p = 2$, it corresponds to L_2 regularization.

  The L_p regularization term is defined as:

  $$L_p \text{ regularization term} = \lambda \left(\sum_{i=1}^{n} |w_i|^p \right)^{\frac{1}{p}}$$

where λ is the regularization parameter, w_i are the model weights, and n is the total number of weights.

- TL_1 **Regularization (Tikhonov** L_1 **Regularization):** TL_1 regularization is a combination of L_1 and L_2 regularization, aiming to provide the benefits of both techniques. It adds a penalty term to the loss function that consists of both the L_1 and L_2 norms of the weight vector.

  The TL_1 regularization term is defined as:

$$TL_1 \text{ regularization term} = \sum_{i=1}^{n} \frac{(a+1)|w_i|}{a+|w_i|}$$

  where a is the regularization parameter controlling the strengths of the penalty, w_i are the model weights, and n is the total number of weights.

  These regularization techniques serve to control the complexity of models, preventing them from overfitting the training data and improving their ability to generalize to unseen data. By penalizing large weight values, regularization encourages simpler and more robust models that are less sensitive to noise in the training data. The choice of regularization technique and the regularization parameter(s) play a crucial role in achieving optimal model performance.

2.6 Related Work

- **Weight Pruning:** Weight pruning, also known as sparsifying, is a technique aimed at reducing the storage requirements and computational complexity of neural networks by removing unimportant connections with small weights. This approach, proposed by (20), results in networks where the majority of weights are zero, allowing for storage in a sparse format and reducing the overall memory footprint. However, achieving speedup with such methods typically requires specialized sparse matrix operation libraries and/or hardware, and runtime memory savings are often limited as most memory space is occupied by dense activation maps rather than weights. To address this limitation, (21) introduces an approach that imposes a sparse constraint on each weight by introducing additional gate variables. By explicitly pruning connections with zero gate values, this method achieves high compression rates while maintaining model performance, offering improved efficiency in terms of both storage and computational resources.

- **Low-rank Decomposition:** Various approaches have been proposed to achieve model compression through low-rank decomposition in convolutional neural networks (CNNs). (8) utilized singular value decomposition to compress weight tensors of convolutional layers, while Jaderberg et al. exploited redundancy among feature channels and filters to approximate a full-rank filter bank with combinations of rank-one filters. Wen et al. introduced force regularization to guide CNN training towards a low-rank representation, while Xu et al. proposed trained rank pruning, an optimization scheme that integrates low-rank decomposition into the training process, further enhanced by nuclear norm regularization. These methods collectively focus on decomposing pre-trained weight tensors or incorporating low-rank decomposition directly into the training process to achieve model compression and efficiency gains in CNNs.

- **Weight quantization:** Weight quantization techniques have emerged as effective strategies for reducing the storage requirements of neural networks while maintaining computational efficiency. HashNet (9) proposes to hash network weights into different groups before training, enabling weight sharing within each group. By storing only shared weights and hash indices, significant storage space can be saved. However, this approach does not save runtime memory or inference time, as shared weights need to be restored during inference. Improved quantization techniques, as demonstrated by recent studies, achieve remarkable compression rates, such as 35x to 49x on AlexNet and VGGNet, by quantizing real-valued weights. These techniques involve quantizing weights (22) into binary or ternary values, restricting weight values to -1, 1 or -1, 0, 1, respectively. While such aggressive low-bit approximation methods yield substantial model-size savings and speedup, they often incur a moderate loss in accuracy due to the discretization of weight values. Overall, weight quantization techniques offer promising avenues for reducing the storage footprint of neural networks and improving computational efficiency, albeit with trade-offs in accuracy.

- **Neural architecture learning:** Neural architecture learning represents a frontier in the field of deep learning, aiming to automate the design process of convolutional neural networks (CNNs) instead of relying solely on expert intuition. While traditional CNNs are often meticulously crafted by experts, recent research has explored methods for automatically learning network architectures. One such approach, introduced by (25), leverages sub-modular and supermodular optimization techniques to search for network architectures within a given resource budget. Additionally, some works (23; 24) propose the use of reinforcement learning to automatically learn neural architectures, although the immense search space necessitates training hundreds of models to distinguish superior architectures.

Chapter 3

Unveiling Channel-Level Sparsity: Leveraging Scaling Layers for Efficient Channel Pruning in CNNs

In our pursuit of achieving channel-level sparsity in deep convolutional neural networks (CNNs), we embark on a quest to develop a straightforward yet effective scheme. This section delves into the merits and obstacles associated with channel-level sparsity, elucidating our strategy for leveraging scaling layers within batch normalization to discern and prune redundant channels in the network.

Channel-level sparsity presents a promising avenue for enhancing the efficiency of CNNs by selectively pruning unimportant channels. However, its implementation poses notable challenges, including the need for an effective mechanism to identify and eliminate redundant channels while preserving network performance and interpretability.

To overcome these challenges, we are focusing to capitalize on the scaling layers within batch normalization. Batch normalization, a pivotal technique in deep learning, not only normalizes activations but also introduces learnable scaling parameters. By analyzing the magnitudes of these scaling parameters across channels, we can effectively discern the importance of each channel in the network. Channels with negligible scaling parameters are deemed less crucial and thus ripe for pruning.

By integrating this mechanism into our pruning scheme, we aim to achieve channel-level sparsity in a streamlined and efficient manner, thereby enhancing the computational efficiency and interpretability of deep CNNs. Through our innovative approach, we endeavor to unlock the potential of channel-level sparsity as a means of optimizing deep neural network architectures for diverse applications.

3.1 Advantages of Channel-level Sparsity

- **Flexibility and Generality:** Sparsity (17; 15) can be achieved at various levels, including weight-level, kernel-level, channel-level, or layer-level. Fine-grained sparsity, such as weight-level sparsity, offers the highest flexibility and generality, resulting in higher compression rates. However, it often necessitates specialized software or hardware accelerators for fast inference on sparsified models.

- **Inference Speedup:** Coarsest layer-level sparsity does not require special packages for inference speedup, making it more accessible. However, it's less flexible as entire layers need to be pruned. Removing layers is most effective when the network depth is substantial (17; 26), typically exceeding 50 layers.

- **Tradeoff between Flexibility and Implementation Ease:** Channel-level sparsity strikes a balance between flexibility and ease of implementation. It can be applied to various CNN architectures or fully-connected networks, treating each neuron as a channel. The resulting network is essentially a "thinned" version of the unpruned network, facilitating efficient inference on conventional CNN platforms.

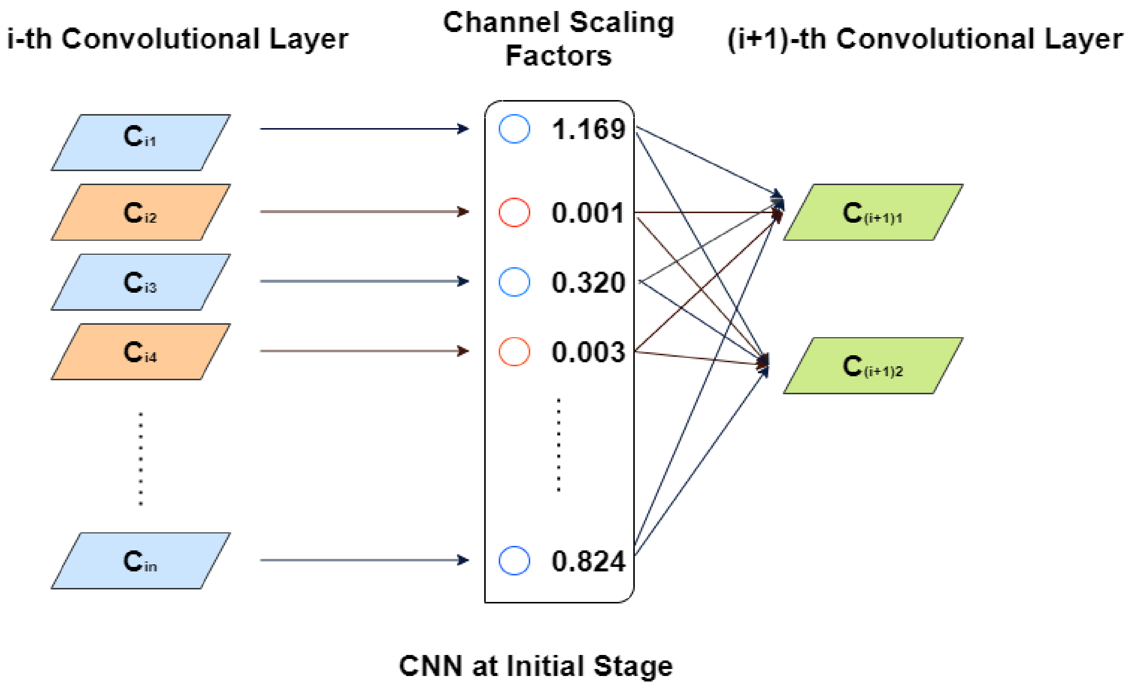

FIGURE 3.1: CNN before Pruning

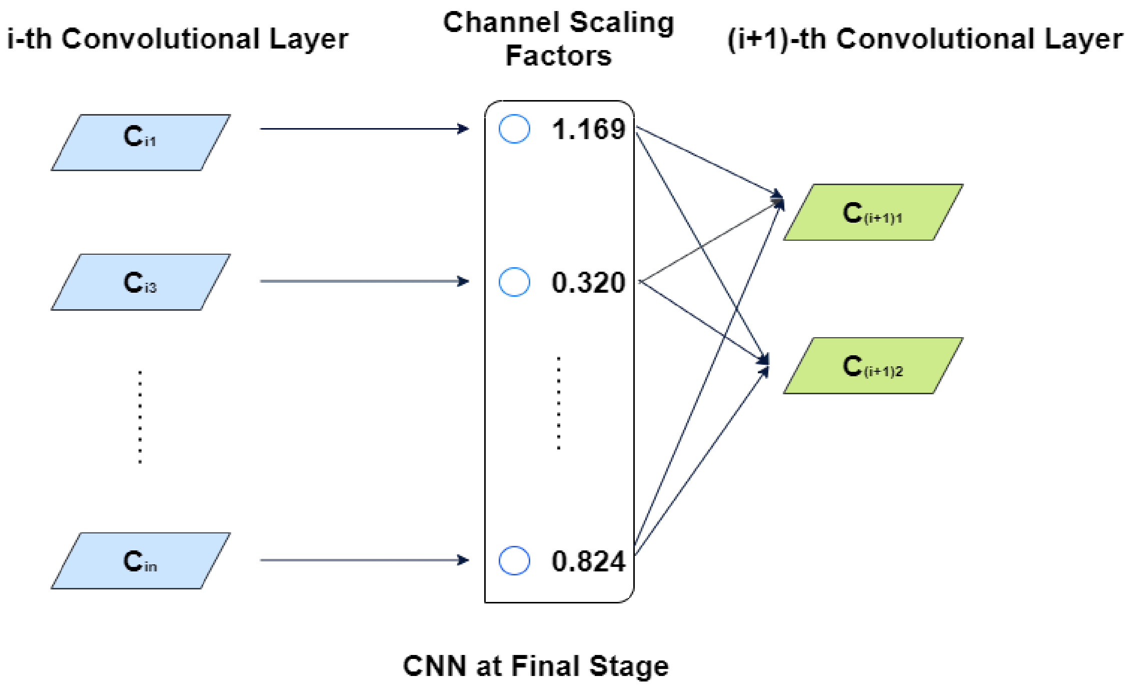

FIGURE 3.2: CNN after Pruning

3.2 Sparsity-induced Penalty and Scaling Factors

Our approach revolves around introducing a scaling factor for each channel in the network, which is multiplied to the output of that specific channel. Through joint training of the network weights and these scaling factors, with sparsity regularization imposed on the latter, we aim to identify and prune channels with small factors. This process is followed by fine-tuning the pruned network to ensure optimal performance.

The training objective $L = \sum_{x,y} l(f(x, w), y) + \lambda \sum_{\gamma} g(\gamma)$,

where (x, y) represents the input and target during training, W denotes the trainable weights, the first sum term corresponds to the normal training loss of a CNN, $g(\cdot)$ represents a sparsity-induced penalty on the scaling factors, and λ balances the two terms. In our experiments, we employ $g(s) = |s|$, which is the L_1-norm and commonly used to induce sparsity. We utilize sub-gradient descent as the optimization method for the non-smooth L_1 penalty term. Alternatively, the L_1 penalty can be replaced with the smooth-L_1 penalty to avoid using sub-gradient at non-smooth points.

Pruning a channel entails removing all the incoming and outgoing connections associated with that channel, resulting in a narrower network without the need for special sparse computation packages. The scaling factors serve as agents for channel selection, as they are jointly optimized with the network weights. This enables the network to automatically identify insignificant channels, which can be safely removed without significantly impacting generalization performance.

3.3 Utilizing Batch Normalization Scaling Factors for Channel-Wise Pruning

Leveraging the scaling factors within Batch Normalization (BN) (14) layers presents a straightforward and efficient method for incorporating channel-wise scaling factors into convolutional neural networks (CNNs). BN, a standard technique adopted by modern CNNs, normalizes internal activations using mini-batch statistics. Specifically, given the input z_{in} and output z_{out} of a BN layer, along with the current mini-batch B, BN performs the transformation $\hat{z} = \frac{z_{\text{in}} - \mu_B}{\sigma_B} + \epsilon$ and $z_{\text{out}} = \hat{z} \cdot \gamma + \beta$, where μ_B and σ_B are the mean and standard deviation of input activations over B, and γ and β are trainable affine transformation parameters (scale and shift) providing linear transformation capabilities.

Typically, a BN layer is inserted after a convolutional layer, with channel-wise scaling and shifting parameters. Thus, we can directly utilize these parameters as the scaling factors required for channel-level pruning, without introducing any overhead to the network. This approach is particularly effective for learning meaningful scaling factors for channel pruning. If we were to add scaling layers to a CNN without BN layers, the scaling factor values would not convey meaningful channel importance, as both convolutional layers and scaling layers are linear transformations. Additionally, inserting a scaling layer before a BN layer would result in the scaling effect being canceled out by the normalization process in BN. Conversely, inserting a scaling layer after a BN layer would lead to consecutive scaling factors for each channel. Therefore, leveraging the scaling factors within BN layers offers a practical and efficient means of incorporating channel-wise scaling factors into CNN architectures.

3.4 Utilizing Non-convex Sparse Regularization for updating scaling factors

Utilizing nonconvex sparse regularization techniques such as L_p and TL_1 for updating scaling factors in this context offers an alternative approach to channel pruning. These techniques introduce additional regularization terms to the training objective, promoting sparsity in the scaling factors and facilitating the identification and removal of redundant channels.

L_p regularization, with various values of p such as 0.25, 0.5, and 0.75, imposes penalties on the scaling factors based on their L_p norms. By selecting different values of p, we can control the degree of sparsity induced in the scaling factors. For instance, smaller values of p lead to more aggressive sparsity-inducing behavior, whereas larger values of p result in more moderate sparsity.

TL_1 regularization, with parameters a set to 0.5 or 1, introduces a combination of L_1 and L_2 penalties on the scaling factors. This regularization scheme encourages sparsity while also ensuring smoothness in the resulting solution. The choice of parameter a influences the trade-off between sparsity and smoothness, with higher values of a favoring sparser solutions.

By incorporating non-convex sparse regularization techniques like L_p and TL_1 into the optimization process for updating scaling factors, we can effectively encourage the emergence of sparse representations in the network. This promotes the identification and pruning of redundant channels, leading to more compact and efficient network architectures. Additionally, by experimenting with different values of p for L_p regularization and a for TL_1 regularization, we can fine-tune the degree of sparsity induced in the scaling factors to strike an optimal balance between model compression and performance. Overall, leveraging non-convex sparse regularization techniques offers a versatile and effective strategy for updating scaling factors in this context.

3.5 Fine-tuning

After training with channel-level sparsity-induced regularization, we obtain a model in which many scaling factors approach zero, indicating potentially insignificant channels (see Figure 3.1,3.2). Subsequently, we initiate channel pruning by removing channels with near-zero scaling factors, thereby eliminating all associated incoming and outgoing connections along with their corresponding weights. To determine which channels to prune, we employ a global threshold across all layers, defined as a certain percentile of all scaling factor values. This results in a more compact network with fewer parameters, reduced runtime memory requirements, and decreased computational operations. Although pruning may initially cause some loss in accuracy, especially with high pruning ratios, this can be mitigated through fine-tuning on the pruned network. In our experiments, we observed that the fine-tuned narrow network often achieves equal or even higher accuracy than the original unpruned network in many cases. This underscores the effectiveness of channel pruning followed by fine-tuning in optimizing network performance while reducing model complexity.

Chapter 4

Methodology

4.1 Experimental Setup

We conducted experiments to evaluate our proposed approach on the VGGNet architecture using the CIFAR-10 dataset. link

4.2 CIFAR Datasets

The CIFAR datasets (5) are widely used benchmarks in the field of computer vision. CIFAR-10 consists of natural images with a resolution of 32x32 pixels, categorized into 10 classes. The CIFAR-10 dataset contains 50,000 training images and 10,000 test images. Additionally, a validation set comprising 5,000 images is separated from the training set for hyperparameter tuning, specifically for searching for the optimal value of lambda in our regularization equation. Standard data augmentation techniques such as shifting and mirroring are applied to augment (26; 27; 28) the training data. Furthermore, the input data is normalized using channel means and standard deviations. In our experiments, we compare the performance of our method with that of a baseline approach (15) on the CIFAR datasets.

4.3 Network Model

For our experiments, we train VGGNet (4) with 19 layers. The VGGNet architecture, originally designed for ImageNet classification, has been adapted for CIFAR datasets. For our experiments, we utilized a variation of the original VGGNet architecture specifically tailored for CIFAR datasets, as proposed by (29). This variation of VGGNet is optimized for the smaller image size of CIFAR-10 and CIFAR-100 datasets, resulting in improved performance compared to the original architecture.

4.3.1 Variations for CIFAR-10

The VGGNet architecture used for CIFAR-10 experiments involves adjustments to accommodate the characteristics of the CIFAR dataset. These adjustments may include changes in the number of layers, filter sizes, and pooling operations to better suit the lower resolution and fewer classes of CIFAR-10 compared to ImageNet. Additionally, modifications in the input size and output classes are made to align with the specifications of CIFAR-10. These variations ensure that the VGGNet architecture is well-suited for the task of image classification on the CIFAR-10 dataset, facilitating effective experimentation and evaluation of our proposed approach.

4.4 Training, Pruning and Finetuning

4.4.1 Normal Training

In our baseline experiments, we trained all networks from scratch using stochastic gradient descent (SGD) optimization. Specifically, for training on the CIFAR datasets, we employed a minibatch size of 64 samples for 20 epochs. To facilitate effective training, we initialized the learning rate to 0.1 and applied a learning rate schedule where it was divided by 10 at 50% and 75% of the total number of training epochs. Additionally, we incorporated a weight decay of 10^{-4} to prevent overfitting and Nesterov momentum (30) with a value of 0.9 without dampening to accelerate convergence. The weight initialization method proposed by (31) was adopted, which is known to promote stable training and improved performance. These optimization settings closely align with the original implementation described in (32), ensuring consistency and comparability with existing research. Notably, in all our experiments, we initialized all channel scaling factors to 0.5, as this initialization strategy has been shown to yield higher accuracy for the baseline models compared to the default setting of initializing all scaling factors to 1 as suggested in (32). This initialization choice is crucial for achieving optimal performance and stability during training.

4.4.2 Training with Sparsity

When training with channel sparse regularization on the CIFAR dataset, the hyperparameter λ is crucial in balancing the tradeoff between empirical loss and sparsity induction. To determine an appropriate value for lambda, which controls this tradeoff, we conduct a grid search over values of 10^{-3}, 10^{-4}, and 10^{-5} on the CIFAR-10 validation set. For VGGNet architectures, we empirically set lambda to 10^{-4} based on preliminary experiments. All other training settings remain consistent with normal training procedures.

With the chosen regularization parameter λ set to 10^{-4}, we train the regularized models using a variety of sparse regularization penalties applied to the scaling factors. Specifically, we explore the efficacy of L_1 regularization, L_p regularization with p values of 0.25, 0.5, and 0.75, as well as TL_1 regularization with a values of 0.5 and 1. These penalties are applied to the scaling factors to encourage sparsity and facilitate the identification and removal of redundant channels during training. By systematically exploring different penalty functions, we aim to determine the most effective regularization approach for inducing sparsity in the network while maintaining or improving performance on the classification task.

4.4.3 Pruning

When pruning the channels of models trained with sparsity, it is essential to determine a suitable pruning threshold on the scaling factors. We adopt a simpler strategy by using a global pruning threshold. This threshold is determined by selecting a percentile among all scaling factors, such as pruning 40% or 60% of channels.

The pruning process is implemented by constructing a new narrower model and copying the corresponding weights from the model trained with sparsity. This ensures that the pruned model retains the overall structure and learned features of the original model while removing redundant channels.

We explore the effects of pruning across various pruning percentages, ranging from 10% to 70%. By systematically varying the pruning percentage, we aim to investigate the impact of channel pruning on model size, computational efficiency, and performance on the classification task. This comprehensive analysis allows us to determine the optimal pruning strategy for achieving efficient and effective network compression while maintaining satisfactory performance levels.

4.4.4 Retraining After Pruning / Fine-tuning

After pruning, we obtain a narrower and more compact model, which undergoes a fine-tuning process to refine its performance. For fine-tuning on the CIFAR dataset, we utilize the same optimization settings as those used during the initial training phase. Specifically, we maintain the same learning rate and number of epochs, which are set to 15 epochs in our experiments.

During fine-tuning, the pruned model is further trained on the CIFAR dataset to adapt its parameters to the new network architecture resulting from channel pruning. By fine-tuning the pruned model, we aim to restore and potentially enhance its performance to levels comparable to the original, unpruned model. This process allows us to optimize the pruned model for improved accuracy while leveraging the benefits of network compression achieved through channel pruning.

Through fine-tuning, we ensure that the pruned model retains its ability to accurately classify images on the CIFAR dataset, thereby validating the effectiveness of the pruning and fine-tuning process in achieving efficient network compression without sacrificing classification performance.

Chapter 5

Results and Observations

5.1 Quantitative Evaluation

5.1.1 Training and validation of model with and without sparsity regularization

In our experimentation, we began by loading the CIFAR-10 dataset using the function `torch.utils.data.DataLoader()`, a commonly used tool for data loading and preprocessing in PyTorch. Our focus then shifted to the VGG16 model, a popular convolutional neural network architecture known for its effectiveness in image classification tasks. We initialized the model parameters with a specific configuration denoted by `cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512]` (3), which defines the number of convolutional filters and max-pooling layers in each stage of the network.

Having established our model architecture, we proceeded to evaluate the effectiveness of sparsity regularization within batch normalization layers for scaling the channels. To establish a baseline for comparison, we conducted initial training and testing without incorporating any sparsity regularization techniques. This baseline training was conducted for 20 epochs, during which the model was trained on the CIFAR-10 dataset. Following training, we assessed the model's performance by evaluating its accuracy on a separate test set. The resulting accuracy achieved was approximately 90%, with an average test loss of 0.30.

Table 5.1: Performance Metrics including Test Accuracy and Test Loss across Epochs without Sparsity Regularization

Epoch	**Accuracy %**	**Average loss**
0	53.8	1.3197
1	66.2	0.9983
2	71.9	0.8373
3	74.8	0.7528
4	77.0	0.7152
5	77.9	0.6753
6	76.6	0.7060
7	80.9	0.5831
8	80.6	0.5915
9	84.4	0.4726
10	88.9	0.3304
11	89.2	0.3232
12	89.4	0.3163
13	89.4	0.3197
14	89.8	0.3176
15	90.2	0.3108
16	90.8	0.3070
17	91.5	0.3103
18	92.4	0.3076
19	92.6	0.3064

This initial phase of experimentation provided us with a benchmark against which to compare the performance of the model after incorporating sparsity regularization techniques. By establishing a baseline accuracy and loss, we could effectively evaluate the impact of sparsity regularization on model performance and determine its effectiveness in improving the efficiency and compactness of the model.

Following the baseline training phase, we proceeded to incorporate sparsity regularization with L1 regularization into the training process within the batch normalization layers of the VGG16 model. This regularization technique introduced a mechanism for scaling the channels based on their importance, thereby enabling the identification and prioritization of less significant channels within the network architecture.

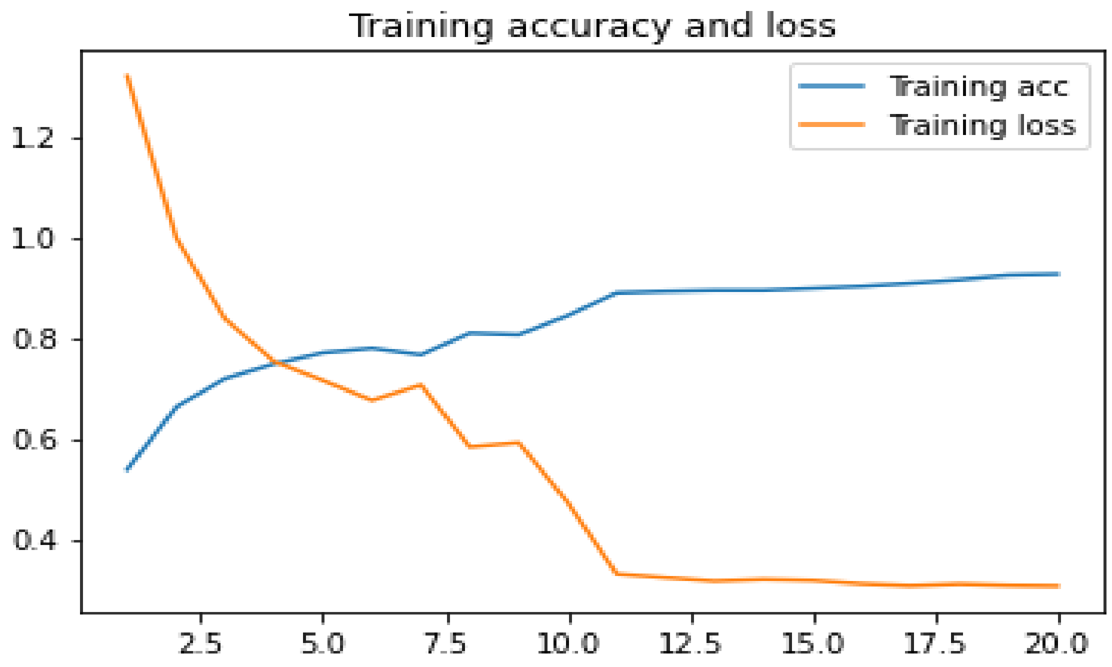

Figure 5.1: Performance Plot of Test Accuracy (Scaled 0 to 1) and Test Loss across Epochs without Sparsity Regularization

By leveraging sparsity regularization within the batch normalization layers, we aimed to promote the identification and removal of redundant channels, thereby improving the efficiency and compactness of the model. This regularization technique allowed us to assign lower scaling factors to less significant channels, effectively reducing their influence on the network's output. As a result, the model could focus its resources more effectively on the most informative channels, leading to potential improvements in performance and efficiency.

Here in Table 5.2, we can notice that accuracy has become almost stagnant i.e. not changing much, we are just updating scaling factors for pruning out channels.

5.1.2 Pruning and Re-training/Fine-tuning

After completing the training with sparsity regularization, our next step involved sorting all the channels based on their corresponding scaling factor values. This sorting process enabled us to rank the channels in descending order of importance,

Table 5.2: Performance Metrics including Test Accuracy and Test Loss Across Epochs with Sparsity Regularization

Epoch	Accuracy %	Average loss
0	90.1	1.3197
1	89.9	0.9983
2	90.0	0.8373
3	90.1	0.7528
4	90.0	0.7152
5	90.1	0.6753
6	90.2	0.7060
7	90.2	0.5831
8	90.2	0.5915
9	90.1	0.4726
10	90.2	0.3304
15	90.3	0.3108
19	90.2	0.3064

with channels possessing higher scaling factor values considered more critical for model performance.

Subsequently, we proceeded with the implementation of channel pruning, a process aimed at reducing the model's complexity by removing less important channels. The pruning percentage (x%) determined the proportion of channels to be pruned from the model. For instance, if the pruning percentage was set to 50%, then the top 50% of channels with the lowest scaling factor values were selected for pruning. These channels, characterized by scaling factor values closer to 0, were deemed less essential for maintaining optimal model performance.

Table 5.3: Performance Comparison of Pruned and Fine-Tuned Models at Various Pruning Levels

Pruning %	Pruned Acc. %	Finetuned Acc. %	Loss	No. of Params
No Pruning	90.08	91.75	0.2824	20030408
10	85.01	91.97	0.2766	16058016
20	30.12	91.75	0.2767	12151178
30	12.00	91.91	0.2786	8838650
40	10.00	92.01	0.2816	6177094
50	10.00	91.96	0.2733	4136018
60	10.00	91.87	0.2858	2721725
70	10.00	91.90	0.2886	1778829

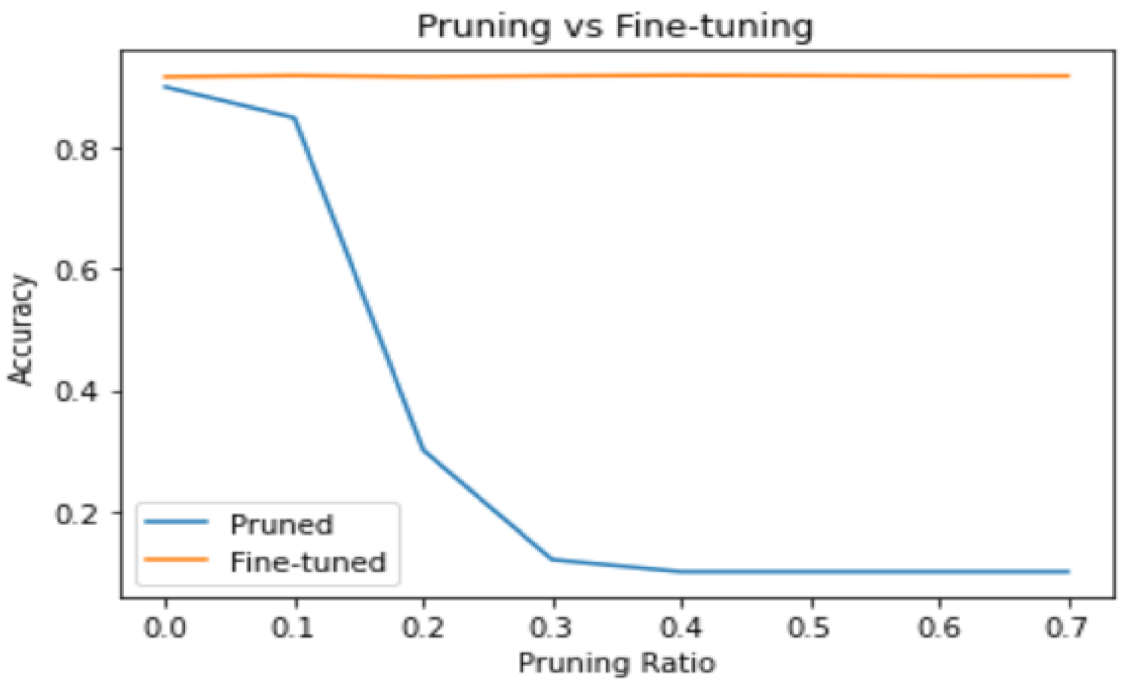

FIGURE 5.2: Comparative Analysis of Accuracy Progression: Pruned vs. Fine-Tuned Models Across Varying Pruning Percentages (Scaled 0 to 1)

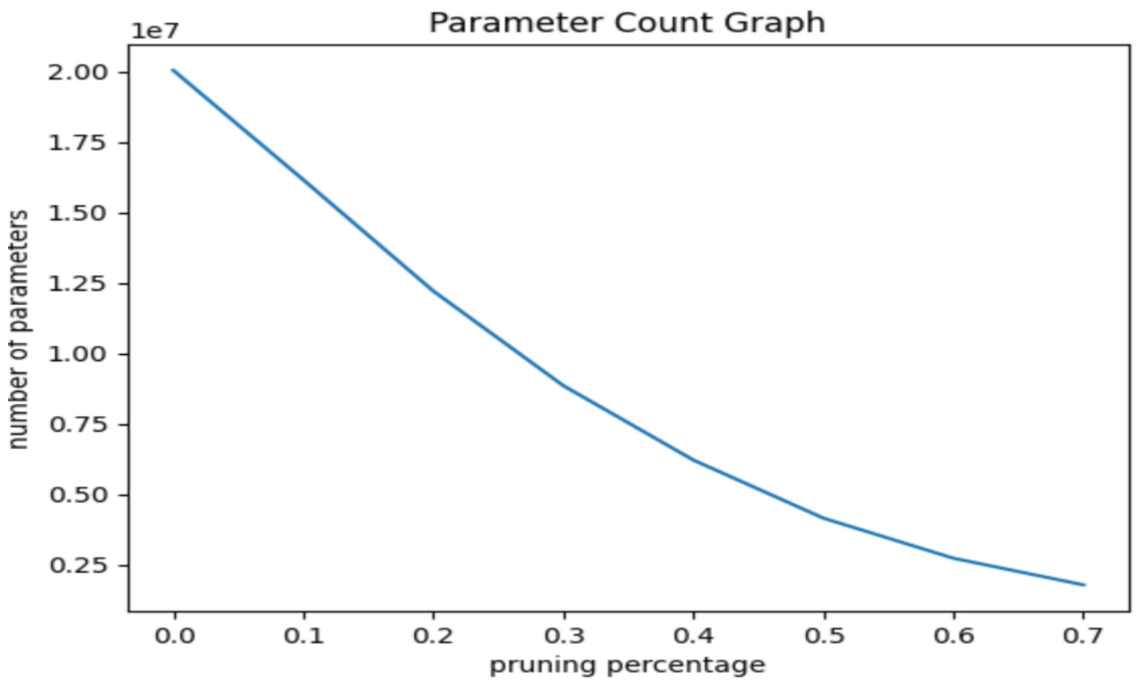

FIGURE 5.3: Effect of Pruning Ratio (Scaled 0 to 1) on Model Complexity: A Comparison of Parameter Count

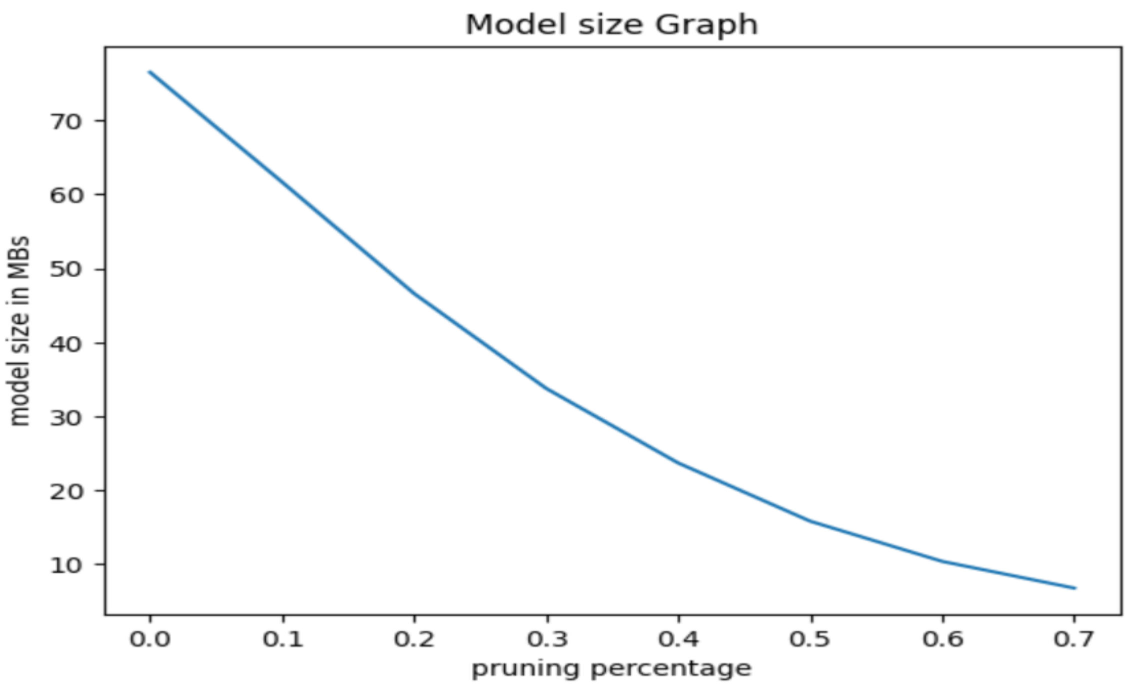

Figure 5.4: Model Size Variation with Pruning Ratio (Scaled 0 to 1): A Comparative Study in MBs

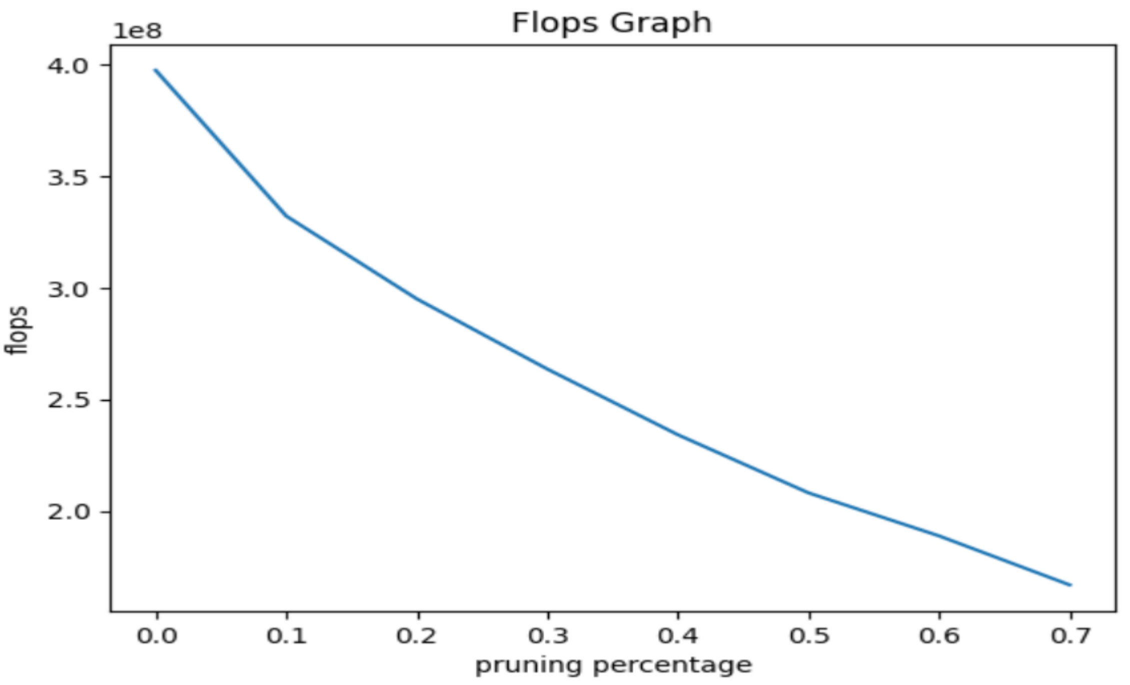

Figure 5.5: Floating Point Operations vs. Pruning Ratio (Scaled 0 to 1): Analyzing Computational Efficiency

5.1.2.1 Pruning with L_p regularization

- **For p = 0.25**

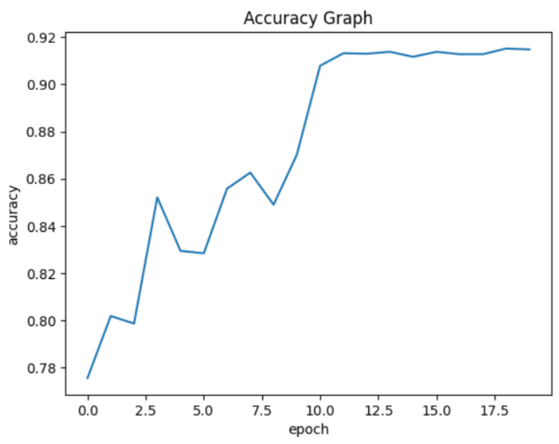

FIGURE 5.6: Accuracy Evolution over Epochs: Tracking Model Performance

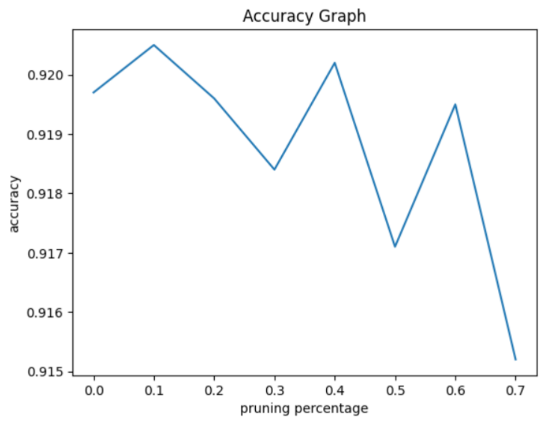

FIGURE 5.7: Accuracy vs. Pruning Ratio: Analyzing Model Performance under Pruning

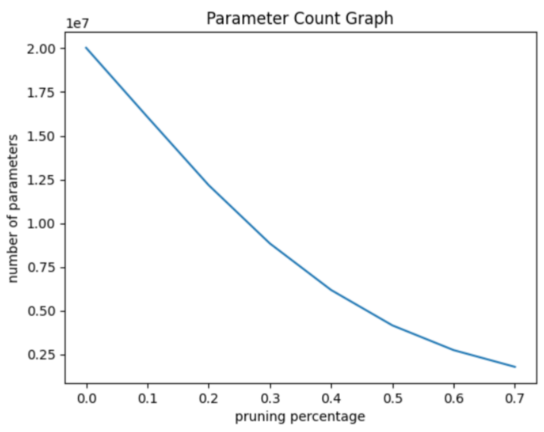

FIGURE 5.8: Parameter Count vs. Pruning Ratio: Understanding Model Complexity

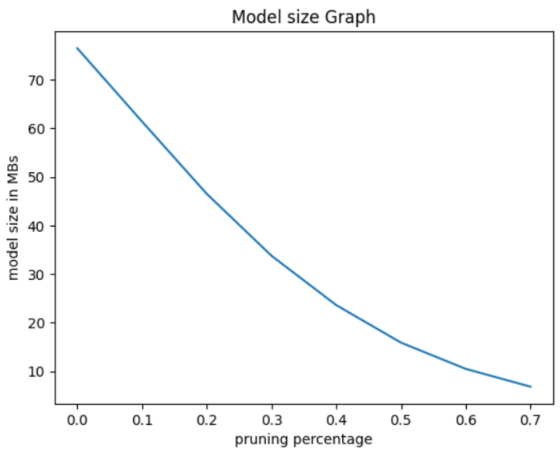

FIGURE 5.9: Model Size vs. Pruning Ratio: Analyzing Compression Efficiency

- **For p = 0.50**

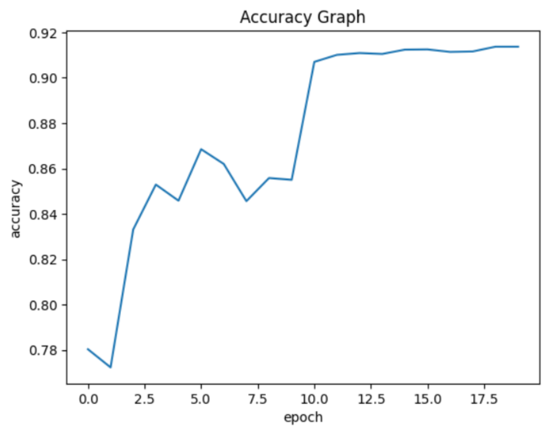

FIGURE 5.10: Accuracy Evolution over Epochs: Tracking Model Performance

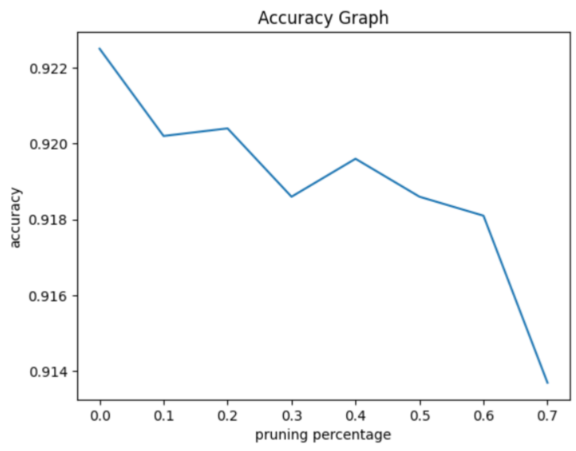

FIGURE 5.11: Accuracy vs. Pruning Ratio: Analyzing Model Performance under Pruning

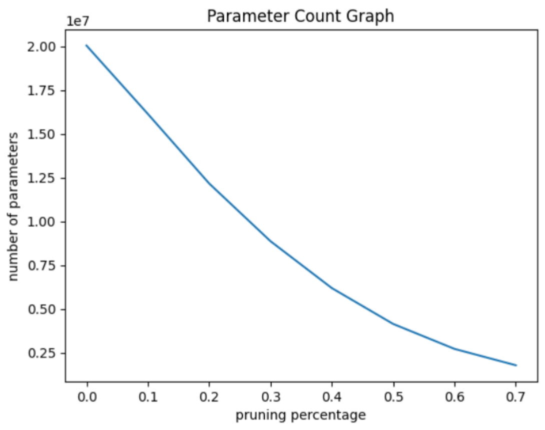

FIGURE 5.12: Parameter Count vs. Pruning Ratio: Understanding Model Complexity

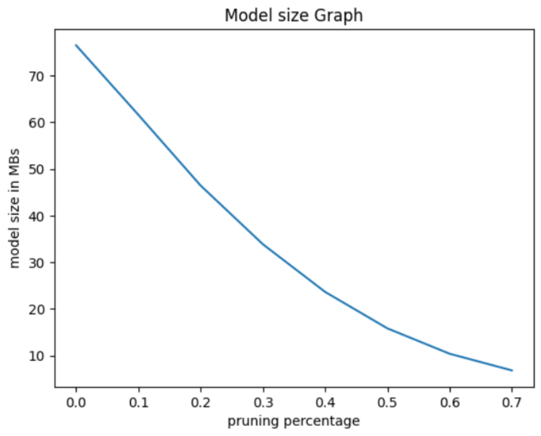

FIGURE 5.13: Model Size vs. Pruning Ratio: Analyzing Compression Efficiency

- **For p = 0.75**

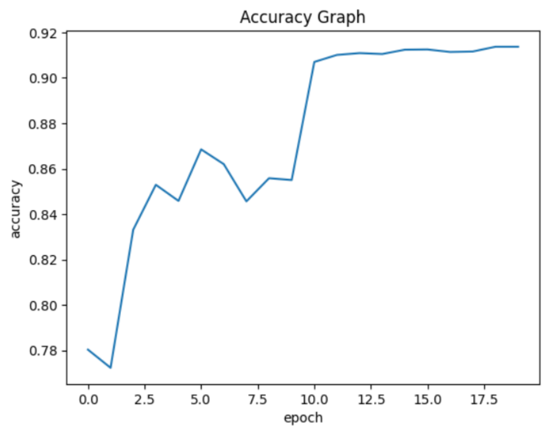

FIGURE 5.14: Accuracy Evolution over Epochs: Tracking Model Performance

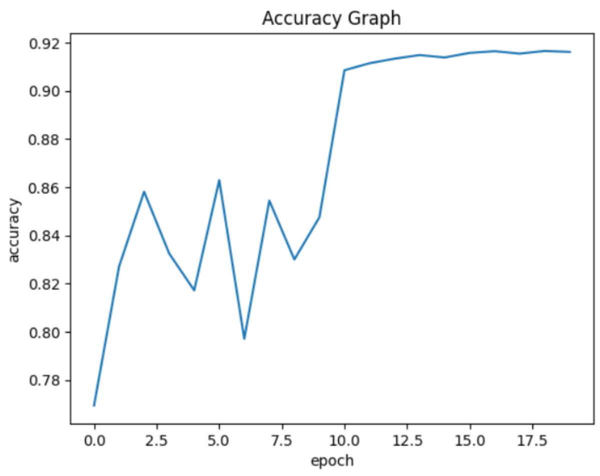

FIGURE 5.15: Accuracy vs. Pruning Ratio: Analyzing Model Performance under Pruning

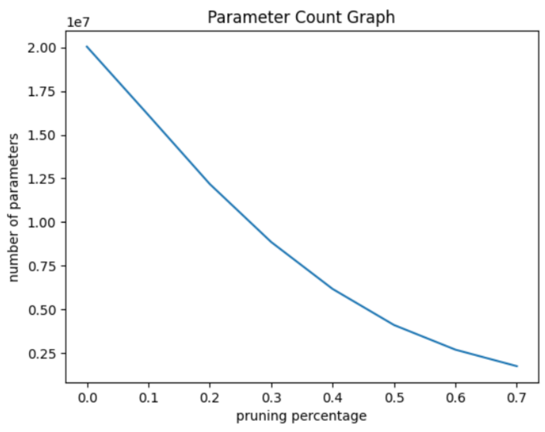

FIGURE 5.16: Parameter Count vs. Pruning Ratio: Understanding Model Complexity

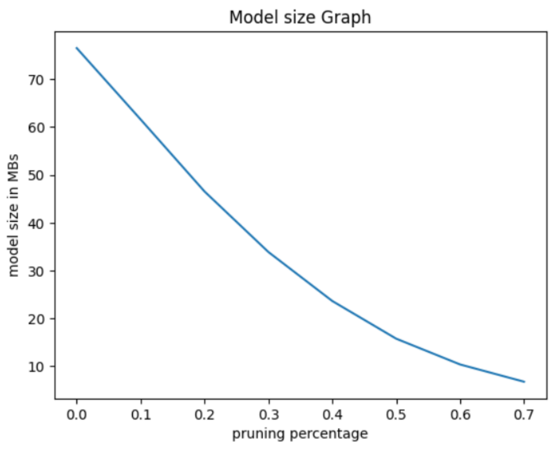

FIGURE 5.17: Model Size vs. Pruning Ratio: Analyzing Compression Efficiency

5.1.2.2 Pruning with TL_1 regularization

- **For a = 1.0**

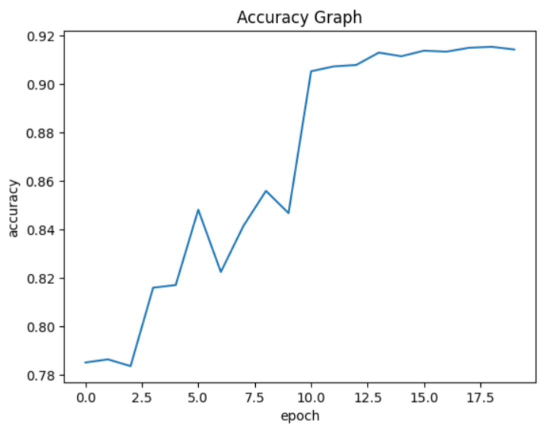

FIGURE 5.18: Accuracy Evolution over Epochs: Tracking Model Performance

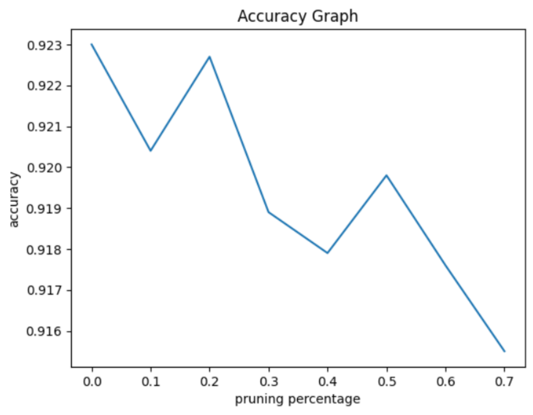

FIGURE 5.19: Accuracy vs. Pruning Ratio: Analyzing Model Performance under Pruning

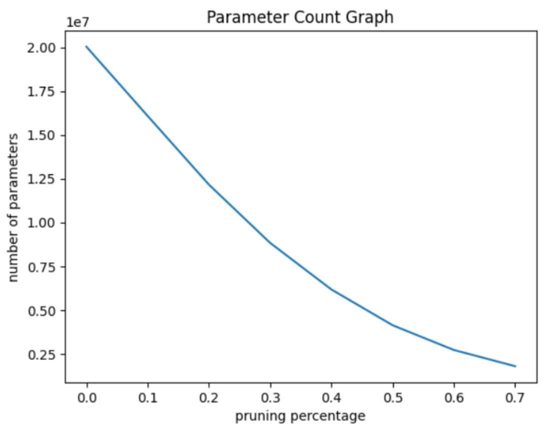

FIGURE 5.20: Parameter Count vs. Pruning Ratio: Understanding Model Complexity

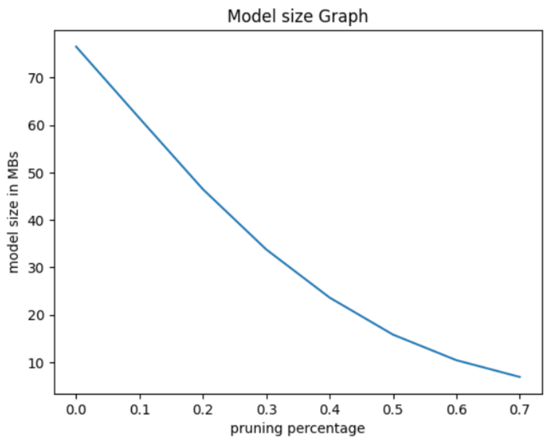

FIGURE 5.21: Model Size vs. Pruning Ratio: Analyzing Compression Efficiency

- **For a = 0.5**

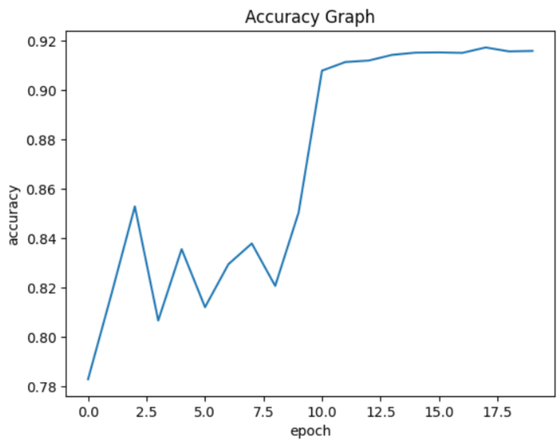

FIGURE 5.22: Accuracy Evolution over Epochs: Tracking Model Performance

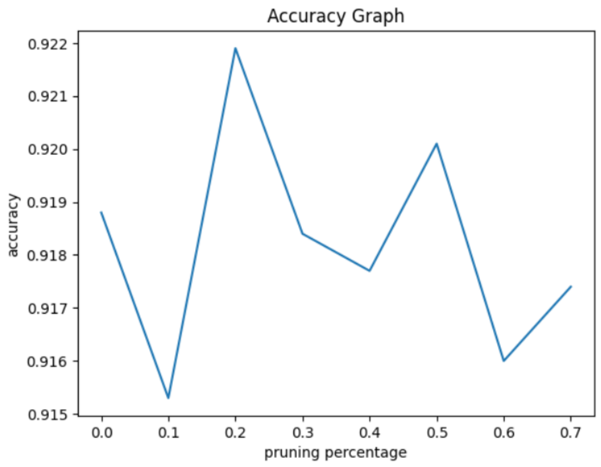

FIGURE 5.23: Accuracy vs. Pruning Ratio: Analyzing Model Performance under Pruning

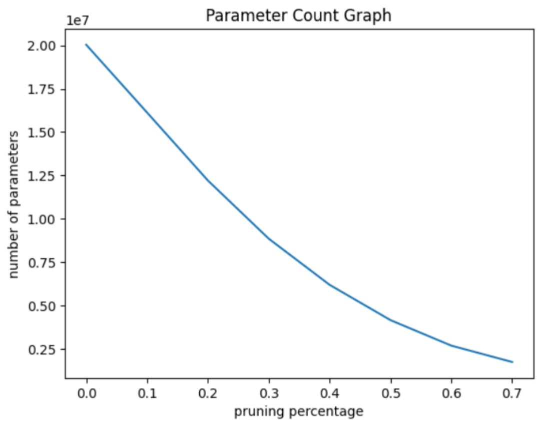

FIGURE 5.24: Parameter Count vs. Pruning Ratio: Understanding Model Complexity

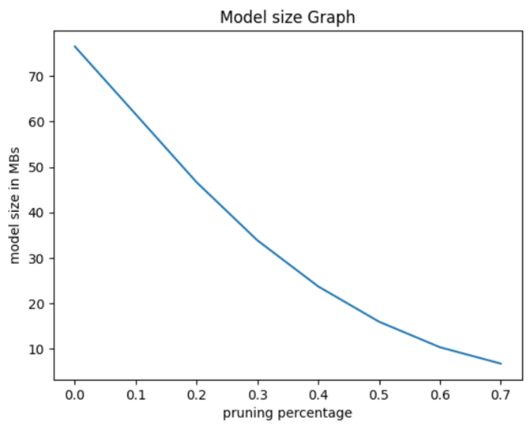

FIGURE 5.25: Model Size vs. Pruning Ratio: Analyzing Compression Efficiency

Table 5.4: Performance Metrics including Test Accuracy and No. of Parameters for various penalties on 30% and 70% channels pruned.

Penalty	**Acc % (30%)**	**Acc % (70%)**	**Params (30%)**	**Params (70%)**
L1	91.71	91.61	8838650	1779829
L0.25	91.24	91.12	8835852	1775829
L0.50	91.86	91.37	8859907	1772747
L0.75	91.59	91.65	8865653	1765178
TL1 (a = 1.0)	91.89	91.55	8832892	1792061
TL1 (a = 0.5)	91.84	91.74	8859391	1754922

5.2 Observations

5.2.1 Parameter and FLOP reductions

The primary objective of our approach is to minimize the computational resources required for model inference while maintaining comparable accuracy levels to the baseline model. Through channel pruning, we aim to achieve significant reductions in both model parameters and floating-point operations (FLOPs), thereby enhancing the model's efficiency and reducing its computational footprint.

In our experimentation, we observed that the pruned models were able to maintain accuracy levels similar to those of the baseline model, demonstrating the effectiveness of our approach in preserving classification performance while achieving significant parameter savings.

The parameter reduction achieved through channel pruning can be substantial, with potential savings of up to 10 times compared to the original model. This reduction in model parameters translates to a significant decrease in memory footprint and storage requirements, making the pruned models more lightweight and efficient for deployment in resource-constrained environments.

Additionally, channel pruning also results in reductions in floating-point operations (FLOPs), which are indicative of the computational complexity of the model during inference. Our experimentation revealed that FLOP reductions typically ranged around 50%, highlighting the substantial decrease in computational overhead achieved through channel pruning.

Overall, the results demonstrate that VGGNet architectures contain a significant amount of redundant parameters that can be effectively pruned without compromising model accuracy. By leveraging channel pruning techniques, we can create more efficient and lightweight models capable of achieving comparable performance to the baseline while requiring fewer computational resources for inference.

5.2.2 Regularization effect and Non-convex penalties

For our analysis on VGGNet applied to the CIFAR-10 dataset, we focused on models pruned at a 70% pruning percentage 5.3, which represents the highest percentage achievable for models trained with L1 regularization. Table 5.4 provides insights into the performance of nonconvex regularized models compared to L1-regularized models.

From the results presented in Table 5.4, it is observed that the nonconvex regularized models, with the exception of L1/4, achieve similar mean test accuracy levels after retraining as the L1-regularized models. Notably, test accuracies for L1, L3/4, and TL_1(a = 1.0) surpass the baseline mean test accuracy. However, it is worth mentioning that while L1 exhibits higher test accuracy compared to other nonconvex regularized models, it is less compressed in terms of model size compared to the other regularized models.

Furthermore, our analysis extended to exploring higher percentages for other nonconvex regularized models. We observed improvements in mean test accuracies for L1/2 and TL_1(a = 0.5), although slight decreases were observed for most other models. Notably, L1/4 experienced the most significant decrease in test accuracy, primarily due to having 70% of its channels pruned, resulting in a considerably higher number of weight parameters being pruned compared to other nonconvex regularized models.

Overall, our analysis highlights the trade-offs associated with different nonconvex regularization techniques in terms of model compression and test accuracy. While certain nonconvex regularized models exhibit improvements in mean test accuracy, the extent of model compression and pruning percentages play a crucial role in determining overall performance. These findings underscore the importance of carefully selecting and tuning regularization techniques to achieve the desired balance between model compression and classification accuracy.

5.3 Future Work

- **Experimentation with Larger Models:** We aim to extend our approach to other large models such as ResNet and DenseNet, which are widely used in deep learning tasks. By applying our method to these models, we can assess its effectiveness across a broader range of architectures and datasets.
- **Exploration with SVHN & CIFAR-100:** In addition to CIFAR-10, we plan to experiment with CIFAR-100, a larger dataset consisting of 100 classes. By evaluating our method on CIFAR-100, we can gain insights into its scalability and performance on more complex and diverse datasets. Similarly, we can try it with SVHN (Street View House Numbers) dataset.
- **Application of Relaxed Variable Splitting Methods:** We intend to explore the application of relaxed variable splitting methods to the regularization of scaling factors within batch normalization layers. This approach can provide a more flexible framework for incorporating various nonconvex regularizers, allowing us to further optimize model compression and accuracy preservation.
- **Assessment of Additional Nonconvex Regularizers:** Building upon our findings with Lp and TL1 penalties, we aim to investigate other nonconvex

regularizers that could offer additional benefits in terms of model compression and accuracy preservation. By systematically comparing different regularization techniques, we can identify the most effective strategies for optimizing model efficiency.

- **Optimization for Real-World Deployment:** Finally, we plan to optimize our method for real-world deployment, ensuring that it is scalable, efficient, and easy to implement in practical scenarios. This may involve further optimizations for inference speed, memory usage, and compatibility with existing hardware and software frameworks.

Chapter 6

Conclusions

In summary, our proposed method introduces sparsity-induced regularization on the scaling factors within batch normalization layers, allowing for the automatic identification and pruning of unimportant channels during training. Through experiments conducted on multiple datasets, we have demonstrated the significant reduction in computational cost (up to 10x) of state-of-the-art networks, with no loss in accuracy. Additionally, our method concurrently reduces model size, runtime memory, and computing operations, while introducing minimal overhead to the training process. Notably, the resulting models require no special libraries or hardware for efficient inference.

Moreover, we have suggested a novel improvement to our method by replacing the L1 penalty with either the Lp or TL1 penalties on the scaling factors in batch normalization layers. Our experiments with VGGNet on CIFAR 10 datasets have shown that nonconvex regularizers can compress models more effectively than L1 at similar channel pruning ratios. TL1 has been found to preserve model accuracy against channel pruning, while L3/4 and L1/2 result in more compressed models than L1 with similar or even higher model accuracy after retraining.

Bibliography

[1] Visualization of batch normalization on a feature map link

[2] Visualization of VGG-16 Architecture link

[3] VGG 16 default config reference link

[4] K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition.* In *ICLR*, 2015.

[5] A. Krizhevsky and G. Hinton. *Learning multiple layers of features from tiny images.* In *Tech Report*, 2009.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[8] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. *Exploiting linear structure within convolutional networks for efficient evaluation.* In *NIPS*, 2014.

[9] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. *Compressing neural networks with the hashing trick.* In *ICML*, 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation.* In *CVPR*, 2014.

[12] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[13] Sudha, V. and Ganeshbabu, Dr. (2020). A Convolutional Neural Network Classifier VGG-19 Architecture for Lesion Detection and Grading in Diabetic Retinopathy Based on Deep Learning. Computers, Materials and Continua. 66. 827-842. 10.32604/cmc.2020.012008.

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

[15] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[16] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *ECCV*, 2016.

[17] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

[18] Y. He, X. Zhang and J. Sun, Channel Pruning for Accelerating Very Deep Neural Networks, *2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy*, 2017

[19] Zhao, M.; Luo, T.; Peng, S.-L.; Tan, J. A Novel Channel Pruning Compression Algorithm Combined with an Attention Mechanism. *Electronics* 2023, 12, 1683

[20] S. Han, J. Pool, J. Tran, andW. Dally. Learning both weights and connections for efficient neural network. In *NIPS*, 2015.

[21] S. Srinivas, A. Subramanya, and R. V. Babu. Training sparse neural networks. CoRR, abs/1611.06694, 2016.

[22] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnornet: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[23] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

[24] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

[25] J. Jin, Z. Yan, K. Fu, N. Jiang, and C. Zhang. Neural network architecture optimization through submodularity and supermodularity. *arXiv preprint arXiv:1609.00074*, 2016.

[26] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[27] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[28] M. Lin, Q. Chen, and S. Yan. Network in network. In ICLR, 2014.

[29] S. Zagoruyko. 92.5% on cifar-10 in torch. https://github.com/szagoruyko/cifar.torch.

[30] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[32] S. Gross and M. Wilber. Training and investigating residual nets. https://github.com/szagoruyko/cifar torch.