EFFICIENT OBJECT DETECTION WITH MODEL COMPRESSION TECHNIQUES

by

Sasha Behrouzi

11017163

Supervisors

Prof. Dr. Binh Vu Prof. Dr. Swati Chandna

Submitted in fulfilment of the requirement for the degree of

MASTER OF SCIENCE

IN

BIG DATA AND BUSINESS ANALYTICS

School of Information, Media and Design

SRH Heidelberg University

September 2023

Declaration of Authorship

I hereby declare that my herewith submitted work presented in this thesis entitled, “Efficient Object Detection with Model Compression Techniques”, is my own original work. I have written it independently without outside help and have not used any sources other than those indicated - in particular, no sources not named in the references.

I have appropriately indicated any direct quotations or passages taken from literature, and the use of intellectual property from other authors, by providing the necessary citations within the work. This applies equally to the sources used for text generation by Artificial Intelligence (AI).

I hereby declare that this work was not previously presented to another examination board, and I also confirm that the PDF version of this paper is exactly identical in content to the hard copy.

Signature of the Candidate

Heidelberg, 15.09.2023
Sasha Behrouzi
11017163

Contents

Declaration of Authorship ii

List of Figures v

List of Tables ix

Acknowledgement xi

Abstract xii

1 Introduction 1

1.1 Problem Statement and Research Gap 3

1.2 Objective of this Reasearch 3

1.3 Thesis Structure Overview 4

2 Related Work 6

2.1 Object Detection 6

2.2 Pruning 7

2.3 Quantization 7

2.4 Knowledge Distillation 8

3 Technical Background 10

3.1 Technical Overview of Object Detection Models 10

3.1.1 Tracing the Evolution of Object Detection 10

3.1.2 Object Detection Datsets 23

3.1.3 Object Detection Metrics 23

3.2 Model Compression Techniques 25

3.2.1 Pruning 25

3.2.2 Quantization 37

3.2.3 Knowledge Distillation 46

4 Methodology 72

4.1 Datasets 73

4.2 Metrics . . . 74
4.2.1 Mean Average Precision (mAP) . . . 75
4.2.2 FLOPs (Floating-Point Operations) . . . 75
4.2.3 Number of Parameters . . . 75
4.3 Tools . . . 75
4.3.1 MMDetection . . . 76
4.3.2 PyTorch . . . 77
4.3.3 MLFlow . . . 77
4.4 Experimental Setup . . . 78
4.4.1 Anchor Pruning . . . 78
4.4.2 Focal and Global Knowledge Distillation . . . 80

5 Experimental Results **83**
5.1 Anchor Pruning . . . 83
5.2 Focal and Global Knowledge Distillation . . . 87

6 Conclusion **90**
6.1 Contributions and Key Findings . . . 90
6.1.1 Exploring Compression Techniques for Object Detection . . . 90
6.1.2 Tackling object detection specific challenges in Knowledge Distillation . . . 91
6.1.3 Experimental Validation of Hypothesis . . . 92
6.2 Future Directions . . . 93

References **95**

List of Figures

3.1 Evolution of Object Detection [1]. 11
3.2 RCNN Object Detection Pipeline [1]. 13
3.3 Fast RCNN architecture [2]. 15
3.4 Faster RCNN architecture [3]. 17
3.5 Faster RCNN architecture [3]. 18
3.6 Retinanet architecture. 20
3.7 Anchor pruning search algorithm. 34
3.8 The Pareto chart is indicating Accuracy and FLOPs trade-offs for different pareto-optimal configurations. The anchor configurations found by greedy anchor search algorithm [4]. 36
3.9 Uniform and non-uniform quantization. Continuous real values within the domain r are transformed into discrete, reduced precision values within the quantized domain Q. Adapted from [5]. 39
3.10 Illustration of symmetric quantization and asymmetric quantization [5]. 40
3.11 In the case of Quantization-Aware Training (QAT), an initially trained model undergoes quantization and is subsequently fine-tuned using training data. This fine-tuning is essential to recalibrate the parameters and mitigate the accuracy loss that can occur due to quantization effects. On the other hand, with Post-Training Quantization (PTQ), a pre-trained model is initially calibrated utilizing calibration data, often derived from a small subset of the training dataset. This calibration process is employed to compute the appropriate clipping ranges and scaling factors. After calibration, the model undergoes quantization based on the determined calibration outcomes [5]. 41

3.12 The sensitivity of the 8th layer when it's quantized to 4 bits ($\Omega_8(4)$) is calculated as shown in the figure. Distilled Data is inputted into both the full-precision ResNet18 model (at the top) and the same model with only the 8_{th} layer quantized to 4 bits (at the bottom). The sensitivity of the 8_{th} layer when it's quantized to 4 bits (($\Omega_8(4)$) is determined by measuring the KL-divergence between the outputs of these two models [6]. 44
3.13 The visualization illustrates that the distilled data reveals more distinct local patterns and structures [6]. 45
3.14 The visual representations of origins for response-based knowledge, feature-based knowledge, and relation-based knowledge within a deep teacher network based on [7]. 47
3.15 Objective functions in Vanilla Knowledge Distillation. 50
3.16 The proposed technique for the visual object detection task utilizes the Faster-RCNN architecture. It consists of the region proposal network and the region classification network. These two networks are simultaneously trained through a multi-task loss mechanism, enabling them to learn both the classifier and bounding-box regressor. To facilitate this, the ultimate output of the teacher's RPN and RCN as the targets for distillation are utilized, while utilizing intermediate layer outputs as hints. The direction of red arrows illustrates the pathways of backpropagation [8]. 54
3.17 In the left two images, there are red and green bounding boxes, which serve as anchor boxes chosen for specific positions. The red anchors are closest match to ground truth bounding boxes, while the green anchors represent examples of near object samples. This selection is driven by the concept that variations in feature responses among anchor locations near objects offer insights into how a trained teacher model generalizes its knowledge. As a result, this method first finds these spots with a lot of useful information and then lets the student model imitate how the teacher responds to them [9]. 59

3.18 Demonstration of the suggested technique for imitating detailed features. The student detector's training involves using both actual object supervision and mimicking the teacher's feature response at nearby object anchor points. The feature-adaptation layer ensures that the student's guided feature layer aligns with the teacher's. To pinpoint valuable areas, we carry out a step-by-step process: calculating the Intersection over Union (IOU) map for each real object bounding box in relation to anchor priors, then filtering and merging potential locations. This results in the creation of the ultimate imitation mask [9]. 60
3.19 Imitation mask examples, resized and placed over the input images [9]. . 62
3.20 Comparison between attention-guided and attention-guided methods [10]. 62
3.21 visualization and distribution of spatial attention for different values of T [10]. 66
3.22 FGD process , encompassing both focal distillation and global distillation [11]. 69
3.23 Global distillation using GcBlock [11]. 70

4.1 Sample images of train wagon from datasets. 73
4.2 Samples from ground truth classes. 73
4.3 Learning curves for both baseline models. 79
4.4 80
4.5 Model summary of diffrent ResNet types using torchvision model summary. 80
4.6 Different distillation schemes. Red for pre-trained means the networks are learned before distillation, yellow for to be trained, it means the networks are learned during distillation [7]. 81

5.1 Pareto charts are output of search algorithm. Each point is indicating a pareto-optimal anchor configuration which can be used for re-training of the model. 84
5.2 84
5.3 Model Complexity comparison comparison between Retinanet baseline model and pruned models after re-training 85
5.4 Comparative mAP Analysis of Pruned Models Comparison of mean Average Precision (mAP) values across pruned model configurations. The mAP values exhibit notable decreases for most pruned models, except for the Retinanet model trained using Configuration A, which showcases an improved performance. 86

5.5 Distillation result. Student mAP gained more accuracy than baseline model during distillation. 87
5.6 Visualiztion of distillation process for Retinanet Res50-Res18. 88
5.7 Self-distillation result for Retinanet Res50. mAP improved for 3.6% after distillation. 88
5.8 Visualiztion of distillation process for self-distillation Retinanet Res50. . 89

List of Tables

3.1 Statistics of some recognized Object Detection Datasets. 23
3.2 -I shows the results of pruning only the internal layers, -M is full method which normalizes importance scores by memory reduction and prunes coupled channels via layer grouping, -F normalizes the importance scores with FLOPs reduction and -U uses the raw Fisher information without normalization. 30
3.3 The table illustrates the correlation between number of anchors, number of bounding boxes and mAP of SSD300 model [12]. 32
3.4 Comparing Accuracy, Computational Complexity (FLOPs), and Inference Time for Retinanet Configurations [4]. 37
3.5 Object detection on Microsoft COCO using RetinaNet. W-bit and A-bit refers to a quantization configuration. For example for W-bit = 32 and A-bit=32, weights are quantized to 32 bits and activations are quantized to 32 bits as well [6]. 46
3.6 Comparison of high-resolution teacher model in [8]. 58
3.7 Experimental Results Using FKD Distillation Technique as Presented in [10]. 67
3.8 Comparisons of different distillation areas [13]. 68
3.9 Outcomes achieved using the FGD method on RetinaNet and RCNN detection frameworks on the COCO dataset. Here, T and S denote the teacher and student detectors, respectively. 71

4.1 Baseline Model configuration and parameters. 79
4.2 Teacher, Student and baseline model configuration and parameters. Student model is trained during distillation. 82

5.1 Evaluation results for different anchor configurations of SSD300 and Retinanet. 86
5.2 Evaluation Results for Knowledge Distillation Experiments. Promising improvements in the mean average precision (mAP) of student models suggest the effectiveness of FGD method. 89

Acknowledgement

I would like to extend my sincere appreciation to my respected thesis supervisors, Prof. Dr. Binh Vu and Prof. Dr. Swati Chandna, for their unwavering support, guidance, and mentorship throughout my academic journey. Your expertise has played a crucial role in guiding the course of this thesis.

I am deeply grateful to Max Dellman from DB Cargo for his exceptional companionship and consistent support throughout the course of this research.

Furthermore, I wish to acknowledge the invaluable support of Dr. Christopher Klein from DB Cargo and Patrick Goldschmidt from DB company for their generous provision of the necessary infrastructure and unwavering support, which played a pivotal role in the successful completion of this research.

In addition, I wish to express my profound appreciation to Maxim Bonnaerens for his invaluable guidance and the time he dedicated to assisting me with my research.

I would also like to extend my heartfelt thanks to my beloved wife for her continuous support and unwavering understanding throughout the challenging process of researching and writing this thesis.

Lastly, I want to express my deep gratitude to my parents, whose enduring love and encouragement have been the bedrock of my life. I also carry in my heart the memory of my beloved grandmother, whom I lost during the course of writing this thesis, and whose hope and love continue to inspire me.

Abstract

Over the last decade, Artificial Intelligence (AI) has undergone a significant transformation, particularly within the domain of Neural Networks (NNs). These advancements have notably enhanced the accuracy of NNs across a wide range of applications, largely attributed to the development of large, over-parameterized models. However, the substantial size of these models presents challenges, especially in resource-limited settings. Simultaneously, a similar evolution has occurred in the realm of object detection models, which are essential for applications such as autonomous driving, fault detection, and security. While these models have seen improvements in accuracy, their over-parameterization introduces computational demands, potentially leading to latency issues and increased operational costs.

While many studies and survey papers have looked into model compression in deep learning, there is a noticeable lack of research on how to apply these techniques to object detection models. Object detection models possess unique characteristics and challenges, necessitating specialized compression approaches. This research aims to bridge the gap in applying model compression techniques to object detection models. Through a comprehensive examination of existing methodologies and practical experimentation, we aim to provide insights into effective compression techniques tailored to object detection models. This study sheds light on the specific challenges inherent in compressing object detection models, assisting researchers and practitioners in selecting appropriate compression techniques for their specific use cases.

To assess the practical viability of these methods, experiments have been designed and implemented within an industrial context, involving object detection in train wagon images. These experiments serve as benchmarks for two state-of-the-art compression techniques: anchor pruning and Focal and Global Knowledge Distillation for Detectors (FGD). Findings indicate significant reductions in GFLOPs in anchor pruning technique. For Retinanet, model complexity reduced by 13% and 40%, with a trade-off of 3.6% and 18% drop in mAP. In FGD, the experiments have been established based on two distinct distillation schemes (offline-distillation and Self-distillation) to assess the FGD technique's capabilities. The results on Retinanet offline-distillation are promising, showcasing reduced complexity of 40% and 54% in backbone size. In both cases FGD managed to improve accuracy by 4% and 14%. Also, in Retinanet Self-distillation FGD improved model accuracy by 5%.

Chapter 1

Introduction

Neural networks have undergone significant transformations in recent years, marking a notable shift in the field of Artificial Intelligence. Progress in this area has resulted in significant improvements in the accuracy of neural networks for a wide range of different types of problems. These improvements have often been the result of pioneering efforts to construct models characterized by an abundance of parameters, often referred to as over-parameterized models. While these over-parameterized neural networks have undoubtedly achieved unprecedented levels of precision in various tasks, they come with a significant drawback: Their cumbersome size makes them impractical for use in resource-limited settings. The rise of these enormous neural network models has exposed a fundamental challenge: finding the right balance between model accuracy, computational efficiency, and energy consumption. This balance is of paramount importance within the framework of our vision for simplified deep learning—an approach where deep learning models seamlessly integrate into nearly every aspect of our daily lives.

Compact deep learning models have the potential to spark a transformation in various fields, such as real-time intelligent healthcare monitoring, autonomous driving, audio analysis, and speech recognition. In essence, it aspires to empower these domains with the ability to harness deep learning's transformative capabilities while meeting the strict requirements of resource-constrained environments. In this regard, the significance of smaller deep learning models cannot be overstated. Think about the potential outcomes in the field of healthcare, where real-time intelligent monitoring could significantly enhance patient care. Deep learning enables the continuous analysis of physiological data, allowing for the early detection of anomalies and immediate intervention. Similarly, in the context of autonomous driving, deep learning makes decisions based on sensor inputs, ensuring the safe navigation of vehicles in complex and dynamic envi-

ronments. Audio analytics and speech recognition, when underpinned by the principles of compressed deep learning, enable devices to respond promptly to user commands, enhancing user experience and accessibility.

As we explore the need for smaller deep learning models, one other aspect centers around the optimization of model size and computational efficiency, with a particular focus on edge devices. Edge devices, which include smartphones, embedded systems, and IoT devices, often operate under severe constraints in terms of processing power and memory. Deploying over-parameterized models on such devices is impractical, as it results in prohibitively high computational demands and energy consumption. Furthermore, privacy concerns surrounding data processing and transmission have emerged as a powerful incentive for the adoption of edge devices. As individuals and organizations become increasingly wary of data security and privacy risks, the appeal of localized data processing at the edge grows stronger. Edge devices provide a solution by allowing data processing to occur nearer to its origin, decreasing the necessity for sensitive information to travel through external networks. This approach not only mitigates privacy concerns but also grants users greater control over their data. Thus, privacy considerations contribute significantly to the motivation for utilizing edge computing solutions, further underscoring the advantages of adopting simplified and compressed deep learning paradigms. Thus, the search for resource-efficient deep learning has prompted the study of methods to make deep learning models smaller and less computationally complex for edge devices.

Additionally, it is essential to address another critical dimension of this exploration —the cost of processing. The world of computational expenses in deep learning is changing rapidly, in tune with the dynamic pace of technological advancements. While some applications might not directly use deep learning models on edge devices, they are more and more open to finding smart ways to reduce the high costs linked to intensive model processing. The computational demands of deep learning, whether it be for model training or real-time inference, have historically placed substantial financial burdens on organizations. Even with the availability of powerful GPUs and cloud-based servers, the cost of provisioning and maintaining such infrastructure can be formidable. More and more companies are recognizing the imperative to optimize computational costs as they scale up their deep learning applications. This entails exploring novel techniques for model compression, and the strategic allocation of resources. In doing so, businesses aim to strike a delicate balance between achieving high model accuracy and minimizing the expences.

Considering the points discussed, the vision of simplified deep learning provides opportunities to apply deep learning in resource-constrained settings. To succeed in this

endeavor, it is important to persist in the research of efficient model compression techniques.

1.1 Problem Statement and Research Gap

A similar transformation has been observed in the field of object detection models. These models, which play a vital role in applications have experienced remarkable advancements in accuracy. These improvements, in line with other deep learning models, are often attributed to the development of over-parameterized models. Consider the case of autonomous vehicles, where real-time object detection is crucial for safe navigation. Over-parameterized object detection models, while promising higher accuracy, exert significant computational demands, potentially leading to latency issues. On the other hand, in various industries, object detection models are extensively utilized, including applications like fault detection, quality control, and security. However, these models often strain computational resources, leading companies to seek ways to streamline their inference pipelines and reduce operational costs.

While several studies and survey papers have thoughtfully explored model compression techniques within the broader deep learning domain, there exists a distinct gap concerning their application to object detection models. This gap is notable because object detection models possess unique characteristics and requirements that distinguish them from other deep learning tasks. Object detection models exhibit a multi-level structure comprising backbones, necks, and heads, which makes them inherently different from single-task classifiers. This multi-level structure serves as a vital component in their ability to locate and classify objects within images efficiently. Moreover, the vast variety of architectural designs for object detection presents a complex challenge when attempting to compress these models while preserving their detection accuracy. Given these traits, it is crucial to thoroughly explore, assess, and study techniques tailored to these models. This endeavor is vital to determine the optimal balance between computational efficiency and improved detection accuracy in compressed models.

1.2 Objective of this Reasearch

This research aims to address this issue by investigating novel model compression techniques designed for the specific challenges posed by object detection models. The unique requirements of object detection, such as maintaining high accuracy while re-

ducing model size and computational complexity, necessitate specialized compression approaches. Through this study, we aim to provide a comprehensive survey of existing research papers in the field, exploring various methods that have been proposed to address the challenges of compressing object detection models.

By conducting an in-depth examination of these methodologies, our research aims to make it easier for researchers and practitioners to choose the right compression techniques for object detection models by highlighting the challenges involved in compression of these models. Furthermore, to evaluate the effectiveness of these methods in practical scenarios, we have designed, implemented, and rigorously tested experiments on an industrial use case involving object detection in train wagons images. These experiments serve as benchmarks for two state-of-the-art compression techniques, namely anchor pruning [12] and Focal and Global Knowledge Distillation for Detectors [13]. The results of these experiments will be discussed in detail, providing valuable insights into the practical applicability and performance of these compression methods in real-world object detection scenarios.

1.3 Thesis Structure Overview

In the forthcoming chapters of this thesis, we will delve into the complex domain of object detection models and compression techniques. The upcoming chapter, Chapter 2, will thoroughly explore previous research in the realm of object detection and methodologies for compressing models. In Chapter 3, we delve deep into the background, starting with a concise historical overview of object detection models. As we navigate through time, we'll explore the evolving complexities within their architectures, highlighting the diversity of models that have emerged. This technical review sets the stage for the challenges that necessitate novel compression techniques. Subsequently, within Chapter 3, we further dissect compression techniques into three distinct parts. In each of these sections, we present the results of the original study for better understanding and comparison.

First, we introduce the concept of pruning, illuminating the fundamental principles that underpin this method. Here, we explore the specifics of state-of-the-art techniques such as Group Fisher Pruning [14] and Anchor Pruning, dissecting their mechanisms and their potential impact on model efficiency.

Moving forward, in the second part our exploration continues with the basics of quantization, a crucial compression technique. We introduce the concept and rationale behind quantization methods, ultimately leading to the presentation of a state-of-the-art tech-

nique, ZeroQ quantization [6], tailored for object detection models.

In the third part, we embark on an in-depth exploration of knowledge distillation. We begin with a comprehensive review of knowledge distillation techniques and then venture into the nuanced challenges introduced by applying these techniques to object detection models. Here, we thoroughly examine these challenges and the various proposed methods designed to address them.

Chapter 4 centers on the research methodology employed in this study. We discuss the two meticulously designed experiments aimed at uncovering how two state-of-the-art compression methods, Anchor Pruning and Focal and Global Knowledge Distillation for Detectors, can enhance efficiency while reducing model size. These investigations are facilitated through well-thought-out benchmarking experiments, supported by the versatile MMDetection library.

In Chapter 5, we will present and discuss the results of the experiments. Finally, in conclusions chapter, we will provide a comprehensive summary of our findings, effectively concluding our research journey. We will bring together the most important findings, examine what they mean for the field of object detection and model compression, and suggest potential directions for future research. This conclusion will serve as a pivotal point, providing a comprehensive understanding of the contributions and significance of this work.

Chapter 2

Related Work

Object detectors have experienced a significant revolution, driven by the collective efforts of numerous researchers working towards their enhancement. Likewise, model compression techniques have evolved over time, as reflected in the existing body of literature. In preparation for our in-depth exploration, this section provides an overview of the related works and the current state of the art within these domains.

2.1 Object Detection

Historically, object detectors can be classified into two primary categories: Traditional Detectors and CNN-Based Two-Stage Detectors. Traditional detectors like [15] , HOG Detector [16] and Deformable Part-Based Model (DPM) [17], paved the way for advancements in CNN-based approaches. The CNN-based detection networks are divided into two-stage and one-stage detectors.

In 2014, Girshick et al. [18] [19], made a breakthrough with their Regions with CNN features (RCNNs), which kickstarted the rapid evolution of object detection. AFurthermore, in 2014, He et al. [20] introduced spatial pyramid pooling networks (SPPNet), demonstrating a speed improvement of more than 20 times compared to R-CNN, while preserving the same level of detection accuracy. In 2015, Fast RCNN [2] is proposed, enhancing both SPPNet and R-CNN. Shorty after FasterRCNN [3] is introduced.

In one-stage object detection, the YOLO (You Only Look Once) [21] method was introduced in 2015. YOLO was faster than two-stage detectors but had lower accuracy in locating objects. Later, SSD [22] was introduced to address this accuracy issue. However, none of these methods could outperform two-stage detectors until Retinanet [11] was developed. It's important to note that all of these methods used anchors as a funda-

mental part of object detection.

CornerNet [23] and CenterNet [24] were introduced as keypoint-based object detectors, offering solutions to challenges such as addressing category imbalance, reducing the need for extensive hand-designed hyperparameters, and shortening the convergence time in comparison to anchor-based detectors. More recently, DETR [25] and Deformable DETR [26] have been proposed, demonstrating remarkable performance improvements.

2.2 Pruning

Network pruning is a technique used to make deep learning models more efficient by removing unnecessary parts. Optical Brain Damage [27] and Optical Brain Surgeon [28] are early works on pruning. They utilize second-order derivative information to estimate weight importance.

There are two main types of network pruning methods: unstructured and structured. Unstructured Methods, such as those discussed in [29] and [30], focus on removing unimportant parts within the model. However, these pruning methods often require specialized hardware for efficient utilization. Structured Methods, on the other hand, target larger portions of the model, like entire channels or filters that are deemed less important. Recent research efforts, as highlighted in [31] and [32], are exploring ways to accomplish this efficiently, particularly by harnessing the power of advanced GPUs. One significant advantage of structured methods is their ability to significantly enhance model speed without the need for specialized hardware.

In the domain of network pruning, various methods and algorithms have been introduced to identify and rank the best parameters for pruning. Group Fisher Pruning, as presented in [14], utilizes a greedy search algorithm and Fisher information to assess the importance of individual channels as well as coupled channels. For the pruning of one-stage anchor-based object detectors, Anchor Pruning, proposed in [12], introduces a method to trim anchors in the detection head without sacrificing accuracy.

2.3 Quantization

Research papers such as [33] and [5] conducted surveys on various quantization methods. In 1984, Claude Shannon authored a groundbreaking paper [34], which marked the first discussion of quantization's effects and its application in coding theory. Shan-

non's work inspired the development of Huffman coding, introduced in [35]. In his subsequent work in 1959 [36], Shannon introduced distortion-rate functions and introduced the concept of vector quantization. This idea was later expanded upon and made practical for real-world communication applications in papers.

In quantization, a significant is, it can result in a notable drop in performance. To mitigate this issue, existing methods advocate for quantization-aware fine-tuning, aiming to recover lost performance, as seen in [37], [38], [39]. Additionally, some papers have focused on developing post-training quantization methods. For instance, in [40], the OMSE method is proposed. Furthermore, [41] presents the ACIQ method. Moreover, [42] introduces the outlier channel splitting (OCS) method to address outlier channel problems, but these methods require access to limited data to reduce performance degradation.

A recent development in this field is the Data Free Quantization (DFQ) method proposed in [43]. This approach extends post-quantization to zero-shot scenarios, where neither training nor testing data are available during quantization. However, it experiences a significant performance drop when quantizing neural networks to 6-bit or lower precision. Additionally, a concurrent paper [44] independently suggests using Batch Normalization statistics to reconstruct input data. They introduce a knowledge-distillation-based technique to further enhance accuracy by generating input data. Similarly ZeroQ method [6] proposes distilled data to improve accuracy.

2.4 Knowledge Distillation

In the domain of knowledge distillation, researchers have explored various strategies to teach smaller student models. One common approach is Vanilla Knowledge Distillation, where the teacher model provides guidance to the student model by sharing either logits or activations from its final layer [45] [46] [47] [48]).

Other reasearches focus on feature-based knowledge distillation [49] [50] [51] [52] [53]. As part of the feature-based distillation category, cross-layer Knowledge Distillation has been introduced, dynamically assigning appropriate teacher layers to student layers using attention mechanisms [54].

Relationship-Based methods has been explored in various papers, including [55] [56] [57] [58] [59].

Several studies have been conducted to enhance knowledge distillation specifically aimed at addressing challenges related to accuracy drops encountered by object de-

tection models [8] pioneered the application of knowledge distillation to object detection, focusing on distilling knowledge from the neck feature, classification head, and regression head. [60] leverages features, which are sampled from the Region Proposal Network (RPN) for the computation of the distillation loss. [9] introduced a fine-grained mask to distill regions calculated by ground-truth bounding boxes. [61] employed Gaussian Masks to cover the ground truth for distillation. GID [62] targeted areas where the student and teacher models exhibited performance differences for distillation. [63] highlighted the significance of both foreground and background information in distillation, emphasizing separate distillation for better student performance. FKD [10] utilizes attention masks and non-local distillation to guide the student. FGD [13] proposed Global Distillation to mitigate accuracy loss during attention-guided distillation.

Chapter 3

Technical Background

3.1 Technical Overview of Object Detection Models

Object detection constitutes a pivotal task within the realm of computer vision, addressing the identification of specific instances of visual objects belonging to particular classes, within images. Two metrics, accuracy and model complexity, play a crucial role in assessing the efficacy of object detection. In this section, we explore the history of object detection from various angles. Our investigation encompasses milestone detectors, datasets employed in object detection research, the evolution of evaluation metrics, and the progression of key techniques. With a specific emphasis on one-stage detectors like Retinanet, and RCNN family from two-stage detectors, we aim to construct a robust technical foundation that will underpin the subsequent parts of our study. By explaining the historical background and offering a detailed technical overview, we prepare for a detailed examination of compression techniques aimed at improving the performance and speed of object detection in the next section.

3.1.1 Tracing the Evolution of Object Detection

The history of object detection can be partitioned into two distinct time periods: the period before 2014, characterized by traditional object detection methods, and the period after 2014, marked by the adoption of deep learning for detection, as shown in Figure 3.1. Overview of the milestone detectors that emerged during these periods is provided in this section [1].

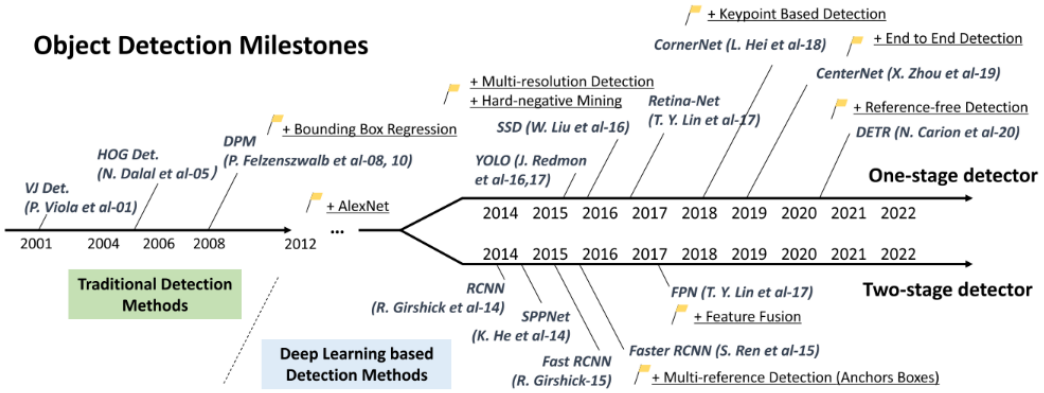


Figure 3.1: Evolution of Object Detection [1].

3.1.1.1 Traditional Detectors:

If we consider the current state of object detection techniques as a significant advancement powered by deep learning, it's important to acknowledge the innovative designs and long-term visions that marked the early days of computer vision in the 1990s. During this era, object detection algorithms primarily relied on manually crafted features due to the limitations of available image representations. As a result, researchers had to create complex feature representations and use various techniques to improve detection speed.

In the year 2001, Viola and Jones made a notable advancement [15] when they created a real-time system for detecting human faces that didn't depend on particular restrictions such as skin color segmentation. Operating on a 700-MHz Pentium III CPU, their system outperformed other algorithms from that era in terms of speed, while still achieving similar levels of accuracy in face detection. The fundamental concept at the core of their approach was a simple technique known as sliding windows, which involved systematically examining all conceivable positions and dimensions within an image to detect human faces [1].

In 2005, Dalal and Triggs introduced a new way to describe images called the Histogram of Oriented Gradients (HOG) [16]. HOG was better than the techniques used before, it could understand images while taking into account things like size, lighting, and position changes, and it could also capture complex patterns. To do this, HOG looked at the image in small pieces and compared the light and dark areas in those pieces. While HOG could be used to find different objects, it was mainly created to spot pedestrians. To make it work with objects of different sizes, HOG resized the image multiple times while keeping its searchs window the same size. HOG became really important for many ways we find objects in pictures, and it's used in many different computer vision

tasks [1].

In 2008, Felzenszwalb et al. introduced the Deformable Parts Model (DPM) [17], which was a big advancement in traditional object detection. DPM built on the concepts of the HOG detector and used a strategy called divide and conquer. This means that when DPM learned how to find objects, it broke them down into smaller parts, like the window, body, and wheels of a car. Then, during the actual detection process, it put together the information from these different parts to figure out if there was a complete object, like a car [1].

The foundational principles and conceptual frameworks established by DPM continue to shape and guide the development of many present-day object detection methods. Despite the significant advancements in object detection techniques, traditional methods still have a strong presence in the field. Today's detectors have seamlessly integrated the wealth of knowledge gained from the early era, incorporating advanced deep learning methodologies and leveraging more effective image representations.

3.1.1.2 CNN-Based Two-Stage Detectors:

In 2012, the emergence of convolutional neural networks (CNNs) was a major breakthrough [64]. The capability of deep convolutional networks to learn robust and high-level feature representations from images could be leveraged for object detection. Taking the initiative in 2014, Girshick et al. pioneered the breakthrough with the proposal of Regions with CNN features (RCNNs) [18] [19]. This seminal work shattered the previous limitations and pushed object detection into a phase of rapid evolution. Within the domain of deep learning-based object detection, two distinct groups of detectors emerged: two-stage detectors and one-stage detectors. The former approach frames the detection process as a "coarse-to-fine" progression, which means they first make a rough guess about where an object might be in an image (coarse), and then refine that guess to get a more precise location (fine). The latter approach aims to complete the detection in a single step. This difference in design ideas has led to various ways of detecting objects, each with its own strengths and weaknesses. Here, we'll dive into Two-Stage Detectors based on Convolutional Neural Networks.

RCNN: RCNN (Regions with CNN features) [18] revolutionized object detection by introducing the integration of Convolutional Neural Networks (CNNs) into the detection pipeline. The framework begins by generating a set of object proposals, also known as object candidate boxes, using a region proposal algorithm like selective search [65].

These proposals serve as potential bounding boxes for objects of interest within the image. Each object proposal undergoes individual processing.

First, the proposed region is re-scaled to a fixed size and fed into a pretrained CNN model, often initialized with weights learned from large-scale image classification tasks such as ImageNet. This CNN acts as a feature extractor, transforming the input region into a high-dimensional feature representation that captures discriminative visual information. Following feature extraction, RCNN utilizes linear Support Vector Machine (SVM) classifiers for each object category of interest. These SVM classifiers are trained to predict the presence of an object category within the proposed region and classify it accordingly. This process enables accurate object localization and recognition. These major steps are given in Figure 3.2.

Figure 3.2: RCNN Object Detection Pipeline [1].

the RCNN comprises three essential modules:

1. The initial module employs the Selective Search algorithm to generate 2,000 region proposals, which serve as potential bounding boxes encompassing objects of interest within the image.

2. The second module resizes each region proposal to a fixed, predefined size and extracts a feature vector with a length of 4,096 from the region. This feature vector encapsulates relevant visual information that characterizes the proposed region.

3. The third module utilizes a pre-trained Support Vector Machine (SVM) algorithm to classify the region proposal into either the background category or one of the specific object classes of interest. This classification process enables accurate identification and categorization of the region proposals.

While RCNN achieved remarkable performance gains, it also had drawbacks:

1. RCNN exhibits a multi-stage architecture, where each stage functions as an autonomous component. Consequently, training the model end-to-end is not feasible.

2. To accommodate the training process, RCNN stores the extracted features from the pre-trained CNN onto disk. This approach necessitates a substantial amount of storage, potentially amounting to hundreds of gigabytes.

3. The reliance on the Selective Search algorithm for region proposal generation introduces certain challenges in terms of time efficiency. The algorithm's inherent time-consuming nature, coupled with its limited customizability for specific detection problems, can impede the overall performance of the RCNN framework.

4. Each region proposal is independently processed by CNN for feature extraction. As a result, the RCNN framework cannot operate in real-time, hindering its applicability in scenarios that require immediate or rapid object detection capabilities.

Fast RCNN: In 2015, Girshick [2] introduced the Fast RCNN detector, an enhanced version of the RCNN. Fast RCNN offers the capability to train both the detector and the bounding box regressor concurrently within a unified network configuration. The following highlights the key contributions of Fast RCNN:

1. Introducing a novel layer termed ROI Pooling, designed to extract feature vectors of equal length from all proposals (i.e., ROIs) within the same image.

2. Enhancing computational efficiency, Fast RCNN enables shared computations (specifically, convolutional layer calculations) across all proposals (i.e., ROIs) rather than performing calculations independently for each proposal. This efficiency is achieved through the utilization of the newly introduced ROI Pooling layer, contributing to Fast RCNN's superior speed compared to RCNN.

3. Differing from RCNN, Fast RCNN eliminates the need for caching extracted features, thereby significantly reducing the disk storage requirements associated with RCNN, which typically demand substantial amounts of storage.

4. Demonstrating improved accuracy over RCNN, Fast RCNN exhibits superior performance in object detection tasks. When evaluated on the VOC07 dataset, Fast RCNN exhibited significant improvements compared to RCNN. It demonstrated an increased mean Average Precision (mAP) from 58.5% achieved by RCNN to a notable 70.0%. Remarkably, this performance enhancement was accompanied by a substantial boost in detection speed, surpassing RCNN by over 200 times.

The general architecture of Fast RCNN is shown in Figure 3.4. The ROI Pooling layer, which receives the feature map from the final convolutional layer as its input, has a

critical role in extracting fixed-length feature vectors from region proposals. The ROI Pooling process involves splitting each region proposal into a grid of cells and applying max pooling to each cell to obtain a single value. The collective values from all cells form the resulting feature vector. By employing a grid size of 2x2, the feature vector length is set to 4. The obtained feature vector is then input into fully connected (FC) layers. The output of the last FC layer branches out into two streams: a softmax layer for predicting class scores and an FC layer for predicting the bounding boxes of detected objects. Compared to RCNN, Fast RCNN offers improved efficiency through shared computations across multiple region proposals. In RCNN, each region proposal is processed independently, leading to a cumulative processing time of $S \times N$ for N regions, where S represents the processing time for a single region. Fast RCNN, on the other hand, shares computations between region proposals, resulting in faster processing times. For example, while RCNN samples a single region of interest (ROI) from each image, Fast RCNN samples multiple ROIs from the same image. This allows for a batch of 128 regions to be selected from just 2 images (64 regions per image), reducing the overall processing time to $2 \times S$. However, it is worth noting that sampling multiple regions from the same image may degrade performance due to the correlation among the regions.

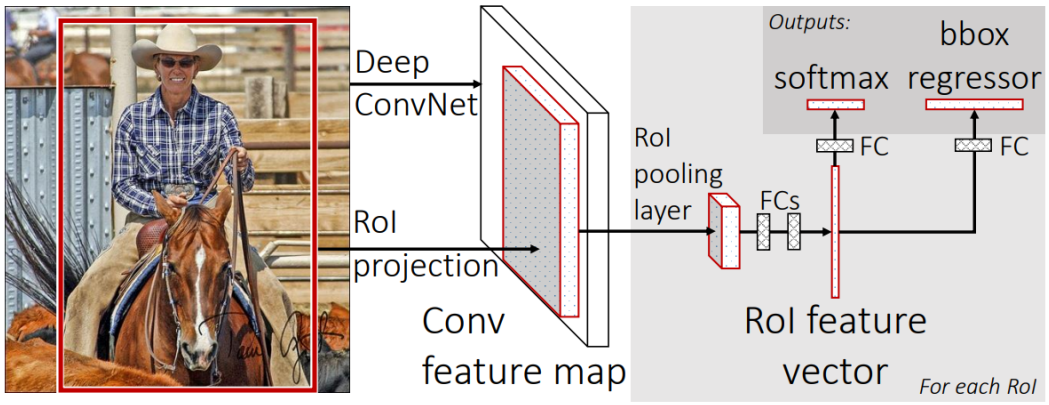


Figure 3.3: Fast RCNN architecture [2].

Faster RCNN: In 2015, Ren et al. [3] introduced a ground-breaking advancement in object detection called Faster RCNN, which followed shortly after the Fast RCNN. Notably, Faster RCNN represented the first deep learning-based detector capable of near-real-time performance. It achieved impressive results, with MS COCO [66] mAP@.5 of 42.7%, VOC07 mAP of 73.2%, and a speed of 17 frames per second using ZF-Net [67]. The main contribution of Faster R-CNN is the incorporation of a region proposal network (RPN). This innovative approach revolutionized the generation of re-

gion proposals by enabling nearly cost-free and highly efficient proposals. The RPN component significantly enhanced the overall detection pipeline, facilitating faster and more accurate object detection capabilities [1]. The main contributions of Faster RCNN paper are:

1. Introducing a Region Proposal Network (RPN). RPN is a CNN to create region proposals. The RPN incorporates the concept of neural network attention, effectively guiding object detection (Fast RCNN) to focus on relevant regions of interest.

2. The paper pioneers the utilization of anchor boxes, eliminating the need for image pyramids or filter pyramids. Anchor boxes serve as reference boxes with specific scales and aspect ratios, allowing for multiple scales and aspect ratios to be represented within a single region. By associating each region with multiple reference anchor boxes, objects at different scales and aspect ratios can be effectively detected, akin to a pyramid of reference anchor boxes.

3. The RPN and Fast RCNN models share the computations performed by the convolutional layers. This sharing of convolutional computations leads to reduced computational time, enhancing the overall efficiency of the detection process.

The architecture of Faster RCNN is shown in Figure 3.4. The operational flow of the Faster RCNN can be summarized as follows:

- RPN generates region proposals.
- The ROI Pooling layer extracting features from each region proposal.
- The extracted feature vectors undergo classification using the Fast RCNN.
- The output of the Fast RCNN includes the returned class scores and corresponding bounding boxes for the detected objects.

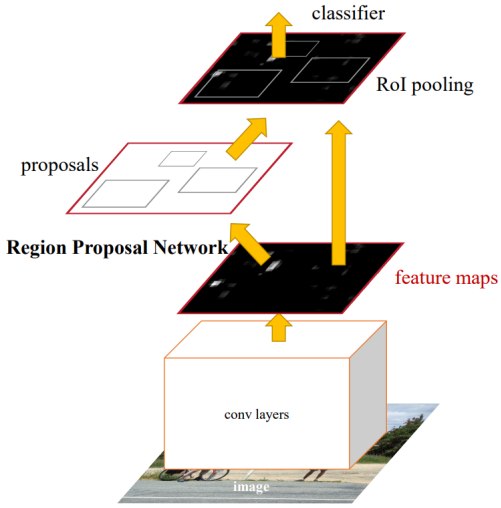


Figure 3.4: Faster RCNN architecture [3].

The RCNN and Fast RCNN models traditionally rely on the Selective Search algorithm to generate region proposals, which are subsequently classified using a pre-trained CNN. However, the Faster RCNN paper proposes the utilization of a novel network known as the region proposal network (RPN) to generate region proposals, offering notable advantages:

1. The RPN facilitates the generation of region proposals using a network that can be trained and customized specifically for the detection task at hand. This network-based approach enables end-to-end training, resulting in improved region proposals compared to generic methods such as Selective Search and EdgeBoxes.

2. By employing a network for proposal generation, the RPN can be trained in a customized manner, tailoring its performance to the specific detection task. This end-to-end training process leads to the production of superior region proposals.

3. The RPN leverages the same convolutional layers used in the Fast RCNN detection network to process the image. Consequently, the RPN incurs no additional time overhead compared to algorithms like Selective Search when generating proposals.

4. The RPN and the Fast RCNN can be effectively merged or unified into a single network due to their shared convolutional layers. This integration allows for training to be performed only once, enhancing efficiency and reducing computational overhead.

The RPN operates on the output feature map obtained from the last shared convolutional layer with the Fast RCNN. A sliding window, with dimensions of $n \times n$, traverses the

feature map, generating multiple candidate region proposals for each window. It is important to note that these proposals are preliminary and will undergo subsequent filtering based on their objectness score. Objectness Score will be explained in the following paragraphs.

To generate region proposals, the feature map from the last shared convolutional layer is processed using a sliding window. This rectangular window has a size of $n \times n$, where for the VGG-16 net, n is set to 3. Within each window, multiple region proposals, called anchor boxes, are created. Each anchor box is defined by two parameters: scale and aspect ratio.

Typically, there are three different scales and three aspect ratios, resulting in a total of nine anchor boxes $K = 9$. However, the number of anchor boxes, denoted as K, may vary and might not always be nine. In other words, from each region proposal, K regions are generated, and each region varies either in terms of scale or aspect ratio. Figure 3.5 illustrates some examples of these anchor box variations.

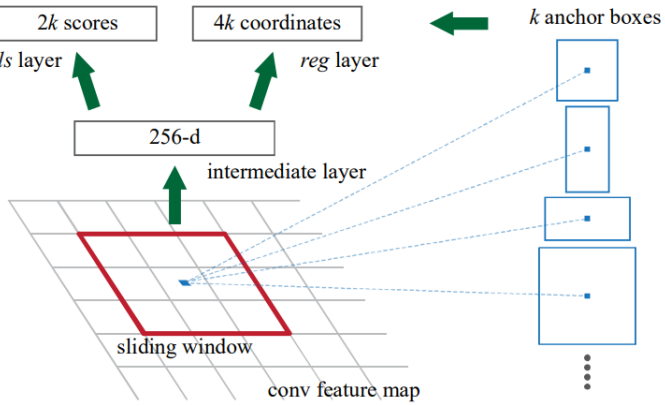


Figure 3.5: Faster RCNN architecture [3].

3.1.1.3 CNN-Based One-Stage Detectors:

Most two-stage detectors adhere to a coarse-to-fine processing approach. The coarse stage focuses on enhancing the ability to recall objects, while the fine stage refines the localization based on the initial detection, placing greater emphasis on discriminative features. These detectors can achieve high precision without additional complexities, but their usage in practical applications is limited due to their slow speed and significant computational complexity.

On the contrary, single-stage detectors have the capability to detect all objects in just one inference step. They are particularly favored for their real-time performance and

ease of deployment, making them suitable for mobile devices. However, when it comes to detecting dense and small objects, their performance noticeably deteriorates.

YOLO: YOLO was introduced by Joseph et al. [21] in 2015, marking the advent of one-stage detectors in the deep learning era. YOLO stands out for its exceptional speed, with a fast variant achieving 155 frames per second (fps) and achieving a VOC07 mean Average Precision (mAP) of 52.7%. An enhanced version of YOLO operates at 45 fps while achieving a VOC07 mAP of 63.4%.

Unlike two-stage detectors, YOLO follows a distinct approach by employing a single neural network to process the entire image. This network divides the image into sections and at the same time forecasts bounding boxes and probabilities for each of these sections. Although YOLO demonstrates remarkable improvements in detection speed, it faces challenges in terms of localization accuracy, particularly for smaller objects, when compared to two-stage detectors.

Single-Shot Multibox Detector (SSD): SSD was introduced by Liu et al. [22] in 2015. One of the key contributions of SSD is the incorporation of multireference and multiresolution detection techniques, which significantly enhance the detection accuracy of a one-stage detector, particularly for smaller objects.

SSD offers advantages in terms of both detection speed and accuracy. It achieves a MS COCO [66] mean Average Precision (mAP) of 46.5% at a threshold of 0.5, while a faster variant operates at a speed of 59 frames per second (fps).

The primary distinction between SSD and previous detectors lies in the way objects of different scales are detected across different layers of the network. Unlike previous approaches that only perform detection on the top layers, SSD incorporates detection at various network layers, enabling it to effectively handle objects of varying sizes.

Retinanet: One-stage detectors have lagged behind two-stage detectors in terms of accuracy, despite their notable speed and simplicity. In 2017, Lin et al. [11] investigated the underlying reasons and proposed Retinanet as a solution. Their investigation uncovered that the main factor responsible for this lag, lies in the substantial imbalance between foreground and background classes encountered during the training of dense detectors.

To address this issue, Retinanet introduced a novel loss function called focal loss. This loss function modifies the standard cross entropy loss by reshaping it in a way that

encourages the detector to focus more on challenging and misclassified examples during training.

The incorporation of focal loss empowers one-stage detectors to achieve comparable accuracy to their two-stage counterparts while maintaining a remarkably high detection speed. Retinanet achieves a MS COCO [66] mean Average Precision (mAP) of 59.1% at a threshold of 0.5, demonstrating the efficacy of focal loss in striking a balance between accuracy and speed in one-stage detectors [1].

Retinanet is a composite network comprising a backbone and two task-specific subnetworks. The backbone network operates as a dedicated convolutional network, generating convolutional feature maps that cover the entire image. Meanwhile, the first subnet conducts convolutional classification using the output of the backbone, while the second subnet performs convolutional bounding box regression using the same output. Although the overall architecture appears straightforward, the authors have meticulously fine-tuned each component to enhance the overall performance. To construct the backbone network for Retinanet, the Feature Pyramid Network (FPN) introduced by T. Y. Lin et al. [68] is adopted. FPN is known for its ability to create a comprehensive feature pyramid at multiple scales through the use of top-down and lateral connections. In the case of Retinanet, the FPN was integrated into the ResNet [69] architecture to leverage its robust features and enhance the overall performance of the network. Figure 3.6 is illustrating the ResNet architecture.

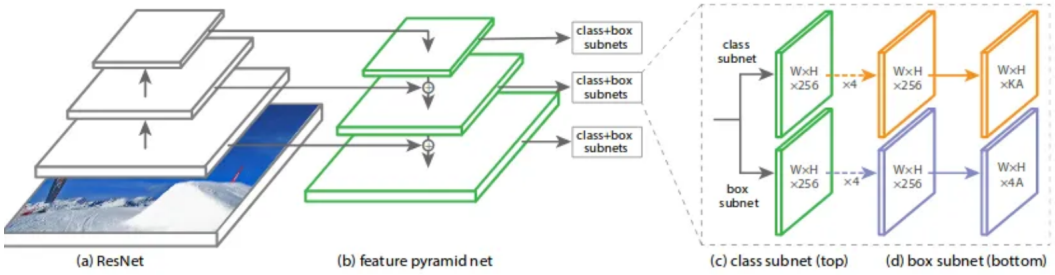


Figure 3.6: Retinanet architecture.

The pyramid structure in Retinanet consists of five levels, denoted as P_3 to P_7. Each level corresponds to specific areas and is associated with translation-invariant anchor boxes. The anchor boxes have varying areas, ranging from 32^2 to 512^2, corresponding to the P_3 to P_7 levels, respectively. To achieve a more comprehensive scale coverage, additional anchors are introduced with sizes defined by the values 2^0, $2^{\frac{1}{3}}$, $2^{\frac{2}{3}}$. As a result, each pyramid level contains nine anchors. For each anchor, two sets of targets are assigned: a one-hot vector representing classification targets K and a four-vector repre-

senting box regression targets. This assignment ensures that each anchor is associated with the appropriate classification and regression objectives.

The classification Subnet in Retinanet is a Fully Convolutional Network that is connected to each level of the pyramid structure. The parameters of this subnet are shared across all levels. With C representing the number of channels (set as 256), A representing the number of anchors (set as 9), and K representing the number of classification targets, the feature map undergoes a series of operations. Firstly, it is passed through four 3×3 convolutional layers with C filters. Subsequently, ReLU activations are applied, followed by another 3×3 convolutional layer with $K \times A$ filters. Finally, sigmoid activations are employed to produce $K \times A$ binary predictions for each spatial location. Consequently, the final output has dimensions $W, H, K \times A$, where W and H denote the width and height of the feature map, respectively. The parameters of the classification subnet were separated from those of the regression subnet. This alteration was introduced to enhance the performance of the detection process. Referring to Figure 3.6, it is evident that the classification subnet and regression subnet in Retinanet operate in parallel as they simultaneously receive the feature map.

To facilitate regression tasks, an additional small FCN is attached to each level of the pyramid structure. This FCN is responsible for estimating the offset from each anchor box to the nearby ground-truth object, provided such an object exists. The structure of this regression subnet is similar to that of the classification subnet. However, the difference lies in the output format, where instead of $K \times A$, the output is $4 \times A$. In the context of the regression subnet, the number 4 represents the specific parameters utilized to determine the offset, including the center coordinates, width, and height. These parameters play a crucial role in accurately localizing the objects within the anchor boxes. It is important to note that although the object classification subnet and the box regression subnet share a common structural framework, they employ separate sets of parameters to fulfill their respective tasks.

CornerNet: Previous approaches primarily relied on anchor boxes for providing references for object classification and regression. However, objects often exhibit variations in terms of their number, location, scale, aspect ratio, and other factors. As a result, previous methods had to employ numerous reference boxes to better align with the ground truth and achieve high performance. Unfortunately, this approach led to category imbalance, the need for numerous hand-designed hyperparameters, and a lengthy convergence time.

In an effort to address these challenges, Law and Deng [23] proposed a novel approach

that deviates from the traditional detection paradigm. Instead, they framed the task as a keypoint prediction problem, specifically predicting the corners of the bounding boxes. Once the keypoints are obtained, the approach decouples and regroups the corner points, utilizing additional embedding information to form the final bounding boxes. This innovative methodology, known as CornerNet, demonstrated superior performance compared to most one-stage detectors of that time, achieving a MS COCO [66] mAP@.5 score of 57.8%.

CenterNet: In 2019, Zhou et al. introduced CenterNet as a novel approach for object detection [24]. CenterNet follows a keypoint-based detection paradigm but distinguishes itself by eliminating costly post-processing steps found in other methods, such as group-based keypoint assignment and non-maximum suppression (NMS). By doing so, CenterNet establishes a fully end-to-end detection network.

CenterNet considers an object as a single point, specifically the center point of the object. It then regresses all relevant attributes of the object, including its size, orientation, location, and pose, based on this reference center point. This approach simplifies the model architecture while maintaining its elegance. Moreover, CenterNet can seamlessly integrate additional tasks, such as 3D object detection, human pose estimation, optical flow learning, and depth estimation, within a unified framework.

Despite its concise detection concept, CenterNet achieves competitive detection results, with a MS COCO [66] mAP@.5 score of 61.1%. This demonstrates the effectiveness and versatility of the CenterNet approach in addressing object detection tasks.

3.1.1.4 Transformer based Detectors:

In recent years, the emergence of Transformers has had a profound impact on various domains of deep learning, including computer vision. Unlike traditional convolutional operators, Transformers rely solely on attention mechanisms to address the limitations of convolutional neural networks (CNNs) and achieve a global-scale receptive field. This paradigm shift has significantly influenced the field of object detection. In 2020, Carion et al. [25] introduced DETR, a groundbreaking approach that treats object detection as a set prediction problem and leverages Transformers to create an end-to-end detection network. With the adoption of Transformers, object detection has entered a new era where anchor boxes or anchor points are no longer necessary for accurate detection.

Building upon the foundation laid by DETR, Zhu et al. [26] proposed Deformable

DETR to further enhance the performance and convergence time of the detection system, particularly for detecting small objects. Deformable DETR has demonstrated state-of-the-art performance on the MS COCO [66] dataset, achieving a MS COCO [66] mAP@.5 score of 71.9%.

These advancements in the form of DETR and Deformable DETR highlight the remarkable progress made in object detection by embracing the power of Transformers and signal a promising future for the field.

3.1.2 Object Detection Datsets

Developing advanced detection algorithms relies heavily on the availability of large and unbiased datasets. Over the past decade, several well-known detection datasets have been released, including the datasets of PASCAL VOC [70], the ImageNet Large Scale Visual Recognition Challenge (e.g., ILSVRC2014) [71], the MS COCO [66] Detection Challenge [72], the Open Images Dataset [73], [4], Objects365 [74], and others. Table 3.1 provides the statistics of these datasets, offering insights into their scale and characteristics. By leveraging these datasets, researchers can train and evaluate detection algorithms more effectively, driving advancements in the field.

Dataset	train		validation		trainval		test	
	images	objects	images	objects	images	objects	images	objects
VOC-2007	2,501	6,301	2,510	6,307	5,011	12,608	4,952	14,976
VOC-2012	5,717	13,609	5,823	13,841	11,540	27,450	10,991	-
ILSVRC-2014	456,567	478,807	20,121	55,502	476,688	534,309	40,152	-
ILSVRC-2017	456,567	478,807	20,121	55,502	476,688	534,309	65,500	-
MS-COCO-2015	82,783	604,907	40,504	291,875	123,287	896,782	81,434	-
MS-COCO-2017	118,287	860,001	5,000	36,781	123,287	896,782	40,670	-
Objects365-2019	600,000	9,623,000	38,000	479,000	638,000	10,102,000	100,000	1,700,00
OID-2020	1,743,042	14,610,229	41,620	303,980	1,784,662	14,914,209	125,436	937,327

Table 3.1: Statistics of some recognized Object Detection Datasets.

3.1.3 Object Detection Metrics

In the early stages of detection research, various evaluation metrics were used to assess detection accuracy. For instance, in early studies on pedestrian detection [16], the miss rate versus false positives per window (FPPW) metric was commonly employed. However, this per-window measurement had limitations and failed to provide a comprehensive evaluation of overall performance at the image level. In 2009, the introduction of the Caltech pedestrian detection benchmark led to a shift in the evaluation metric, transitioning from FPPW to false positives per image (FPPI) [75].

In recent years, average precision (AP) has become the most widely used evaluation

metric for object detection, initially introduced in VOC2007. AP is calculated as the average detection precision at different recall levels and is typically assessed on a category specific basis. The mean average precision (mAP) across all categories is often used as the final performance metric. Object localization accuracy is measured using the inter section over union (IoU) between the predicted bounding box and the ground truth, with a predefined threshold, typically 0.5. If the IoU exceeds the threshold, the object is considered detected, otherwise, it is labeled as missed. The 0.5-IoU mAP has become the de facto metric for evaluating object detection.

Since 2014, with the introduction of MS COCO [66] datasets, there has been increased emphasis on the accuracy of object localization. Instead of using a fixed IoU threshold, the MS COCO [66] AP is calculated by averaging over multiple IoU thresholds ranging from 0.5 to 0.95. This encourages more precise object localization and proves crucial for various real-world applications.

3.2 Model Compression Techniques

3.2.1 Pruning

Pruning is a widely used technique in the field of deep learning that aims to reduce the size and computational complexity of neural network models without significantly compromising their performance. By selectively removing connections, neurons, or entire structures from a trained model, pruning helps eliminate redundant or less important parameters, leading to more efficient models.

A neural network model is represented as $f(x, W)$, where W represents the parameters. Neural network pruning involves taking a model $f(x : W)$ as input and generating a new model $f(x : M \odot W')$. In this context, W' refers to a set of parameters that may differ from W, $M \in \{0, 1\}$ represents a binary mask that selectively fixes certain parameters to 0. In practice, instead of using an explicit mask, pruned parameters in W are either set to zero or entirely removed. Several methods exist for generating a pruned model, from an initial untrained model $f(x : W_0)$, where W_0 is sampled from an initialization distribution D. The majority of neural network pruning strategies are derived from a technique proposed by Han et al. in [29]. This technique involves training the network until convergence, assigning a score to each parameter or structural element, and subsequently pruning the network based on these scores. Due to the reduction in accuracy caused by pruning, the network is further trained through a process known as fine-tuning to regain performance. This pruning and fine-tuning process is often iterated multiple times, gradually decreasing the network's size. Pruning methods within the proposed technique have evolved depending on the specific choices made regarding three key elements: the sparsity structure, the scoring, the scheduling, and the fine-tuning process.

- **Structure:** Some pruning methods focus on removing individual parameters, which is known as unstructured pruning. This approach results in a sparse neural network with a reduced number of parameters. However, such sparse networks may not be organized in a way that allows for efficient speed improvements using modern libraries and hardware. On the other hand, other methods employ structured pruning, where parameters are considered in groups. This approach involves removing entire neurons, filters, or channels from the network. By doing so, these methods take advantage of hardware and software optimizations specifically designed for dense computation. Generally structured pruning can better utilize hardware and software optimizations for faster computations. Examples of structured pruning studies include [76] and [77].

- **Scoring:** In pruning it is common to score parameters by their absolute values, importance coefficients determined during training, or their impact on network performance, which can involve factors like activations or gradients. In some pruning methods, scores are compared locally within specific parts of the network, such as layers. A portion of parameters with the lowest ratings within each structural part is subsequently pruned. In [29], authors explored this approach. On the other hand, other pruning methods consider scores globally by comparing them across the entire network, without distinguishing the location of each parameter. This approach is investigated in [78] and [79]. When pruning, parameters are assigned scores based on their importance.

- **Scheduling:** Pruning methods vary in terms of how much of the network is pruned at each pruning step. Some methods opt to prune all the targeted weights simultaneously in a single step, as demonstrated in [80]. On the other hand, other methods take an iterative approach, pruning a fixed portion of the network in multiple steps. For instance, [29] employ this method, gradually pruning a fraction of the network with each iteration. There are also techniques that adjust the pruning rate based on a more intricate function. In their paper [81] explore this approach, where the rate of pruning may change over time based on certain criteria.

- **Fine-tuning:** When it comes to methods involving fine-tuning after pruning, the usual practice is to continue training the network using the weights that were trained prior to pruning. In this approach, the network builds upon the knowledge it gained before pruning and refines it further through additional training. However, alternative proposals have been suggested. One such proposal, introduced in [79], suggests rewinding the network to an earlier state before pruning. This means reverting the network to a previous set of weights and then training it again from that point, essentially undoing the pruning process. Another alternative proposed by [80] involves completely reinitializing the network. In this approach, the network is reset to its initial state with random weights, and the training process starts fresh after pruning. Generally, after pruning, the most common method is to continue training the network using the weights obtained before pruning.

Pruning's effectiveness is a well-established observation, with numerous techniques capable of significantly reducing the model size without sacrificing accuracy. In fact, pruning can even enhance accuracy for slight compression levels, as demonstrated in studies in [29] and [82].

In this study, specifically, the pruning of object detection models is examined. Extensive research, including studies referenced in this article, has already demonstrated the

effectiveness of pruning methods when implemented in the backbone network of object detection models. By conducting a thorough review of numerous papers to gain insights into the concept of pruning and analyze experimental outcomes, this study places particular emphasis on two significant papers. Firstly, [14]introduces Group Fisher pruning technique specifically tailored for the backbone network. Secondly, [12] proposes a technique that targets anchor pruning in the head part of the object detection network.

3.2.1.1 Group Fisher Pruning for network compression

In this section, the focus is on the utilization of pruning technique proposed in [14]. The paper addresses the challenge of network compression, aiming to reduce memory and computation costs during inference. Previously discussed methods have faced difficulties when dealing with complex network structures that involve interdependencies among channels, such as residual connections, group/depthwise convolution, and feature pyramid networks. To overcome these challenges, authors introduces a general channel pruning technique specifically designed for such complex structures.

The paper proposes a layer grouping algorithm that can automatically identify coupled channels. It also derives a unified metric based on Fisher information [83] to evaluate the importance of individual channels and coupled channel groups. Furthermore, the paper highlights the correlation between GPU inference speedup and memory reduction, leading to the adoption of memory reduction as a normalization factor for importance assessment. The effectiveness of the method is demonstrated through extensive experiments on various backbones, including ResNet, ResNeXt, MobileNetV2, and RegNet, in both image classification and object detection tasks. The results validate the paper's approach, showing significant improvements in inference speed without sacrificing accuracy. In the following, we will explore these concepts in detail. Finally, the experiments done by the authors is being discussed.

Fisher Information Importance In order to achieve structured pruning, a binary mask is introduced and initialized as 1 for each input channel. If we set the channel mask to 0, the channel can be pruned. This means that during the pruning the input tensor $A \in \mathbb{R}^{n \times c \times h \times w}$ for a convolution or fully connected layer is multiplied by the mask $m \in \mathbb{R}^n$ (elementwise multiolication). This operation generates the masked input $\tilde{A} = A \odot m$, which serves as the actual input for each layer, whether it is a convolutional (Conv) or fully connected (FC) layer (with $h = w = 1$ in the case of FC). During inference, the channels with mask values of 0 are explicitly discarded, not only within a layer but also in its parent layers in the computation graph. This is because pruning

input channels of one layer simultaneously prunes the corresponding output channels of its preceding layers. To assess the significance of channel i, the importance s_i is evaluated by employing Taylor expansion on the network loss function L and approximating the change in loss when the channel is discarded (by setting its mask to 0) [14]:

$$s_i = L(m-e_i)-L(m) \approx -e_i^T \bigtriangledown_m L+1/2 e_i^T (\bigtriangledown_m^2 L) e_i = -e_i^T g+1/2 e_i^T H e_i = g_i+1/2 H_{ii} \tag{3.1}$$

$m = 1$ is vector of ones and $e_i \in \mathbb{R}^c$ is one-hot vector in which the i-th element is 1. g is the gradient with respect to m and H is the Hessian matrix. as the model converged, before pruning $\tilde{A} = A \odot m, m = 1$ so $\bigtriangledown_{\tilde{A}} L = \bigtriangledown_A L$. Then we can obtain $g = \bigtriangledown_{\tilde{m}} L \approx 0$ that is $g_i = \frac{\partial L}{\partial m_i} = \frac{1}{N} \sum \frac{\partial L_n}{\partial m_i} \approx 0$ where L_n is the negative log-likelihood loss of n-th sample and $\frac{\partial L_n}{\partial m_i}$ is a sample-wise gradient. Now we need to calculate H_{ii} which are diagonal elements of Hessian matrix H:

$$H_{ii} = \frac{\partial^2 L}{\partial m_i^2} = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial^2 L}{\partial m_i^2} \approx -\frac{\partial^2}{\partial m_i^2} \mathbb{E}[\log p(x|y)] = \mathbb{E}[-\frac{\partial}{\partial m_i} \log p(y|x)]^2 \approx \frac{1}{N} \sum_{n=1}^{N} (\frac{\partial L_n}{\partial m_i})^2 \tag{3.2}$$

In the equation, Fisher information is employed to transform the second-order derivative to the square of the first-order derivative. Consider that $g_i \approx 0$ and $H_{ii} \geq 0$ which correspond to the mean and variance of the sample-wise gradient. Assuming that $A_n \in \mathbb{R}^{c \times h \times w}$ represent the feature map of n_{th} example, we have the masked feature $\tilde{A}_n = A_n \odot \tilde{m}$ where $\tilde{m} \in \mathbb{R}^{c \times h \times w}$ is created by broadcasting $m \in \mathbb{R}^c$, and this allows us to calculate $\bigtriangledown_{\tilde{m}} L_n = A_n \odot \bigtriangledown_{\tilde{A}_n} L_n \in \mathbb{R}^{c \times h \times w}$ in which $\bigtriangledown_{\tilde{A}_n} L_n$ is available during the backward pass so additional computation is not required. Subsequently, the gradient with respect to m, computed on a per-sample basis, can be obtained by summing over the spatial dimensions h and w: $\bigtriangledown_m L_n = sum(\bigtriangledown_{\tilde{m}} L_n) \in \mathbb{R}^c$. Finally the importance score for a channel will be computed by averaging the sample-wise gradients:

$$s_i = \frac{1}{2N} \sum_{n=1}^{N} (\frac{\partial L_n}{\partial m_i})^2 \propto \sum_{n=1}^{N} (\frac{\partial L_n}{\partial m_i})^2 \tag{3.3}$$

So the importance is proportional to the squared gradient of the mask. This calculation is based on the model convergence and to satisfy it, a greedy pruning strategy is employed. We start with a dense model and gradually prune the least important channel. After each pruning step, we fine-tune the model and re-evaluate the importance scores

for the remaining channels. This process repeats, removing the least important channel each time, until the desired result is achieved [14].

Prune Coupled Channels With the proposed method fisher information importance, we can find and prune of layers networks like AlexNet [84] and VGG-16 net [85]. These networks consist of normal Conv layers and sequential structures, where pruning affects only a single layer and the one preceding it. However, more recent networks have become more complex, incorporating structures such as residual connections [69], group convolutions (GConv) [86], depth-wise convolutions (DWConv) [87], and feature pyramid networks (FPN) [68] in object detection. Consequently, coupled channels are necessitating simultaneous pruning of these channels to achieve greater speedup compared to pruning isolated channels alone.

In their paper [14], the authors propose a method called mask sharing in coupled channels to address this issue. The approach involves a layer grouping algorithm applied to the network computation graph G, which includes nodes representing convolution (Conv), batch normalization (BN), ReLU, and pooling (Pool) layers. Using a depth-first search (DFS) algorithm, the parents $P(l_i)$ of each Conv/FC layer l_i is identified. Based on the parents of each layer, we assign layers to different groups. Layers within the same group possess coupled channels that should be pruned simultaneously. This grouping encompasses following scenarios:

- Isolated channels, on the other hand, form their own groups, consisting of only one layer, such as the 3×3 Conv layer in a residual bottleneck.
- When layers have the same parent layers, they are grouped together. This is because their input channels (or channels from their parent layers output) are coupled and should be pruned jointly.
- Layers that have GConv in their parent layers are placed together with their parent layers. The reason for this is that the input and output channels of GConv are coupled.

Once the coupled channels are identified through layer grouping, we ensure their shared connectivity by assigning them the same mask. This enables us to calculate the combined impact of the coupled channels.

Importance Normalization The initial importance scores do not account for the varying computational costs of different channels. However, it is more efficient to prioritize pruning the least important channel that incurs the highest computation cost.

Otherwise, excessive pruning of channels may be performed to achieve the desired speedup, potentially compromising accuracy due to the loss of important parameters. To address this, it is suggested to normalize the importance scores based on the computation reduction associated with each channel. The normalization of importance scores can be done using FLOPs proxy $s_i / \triangle C$ or memory reduction $s_i / \triangle M$

Experiments The authors conducted ablation studies to validate their layer grouping, mask sharing, and memory normalized importance scores for pruning coupled channels. They demonstrated accelerated inference with minimal accuracy loss in various networks, including ResNet, ResNeXt, MobileNetV2, and RegNet. The experiments encompassed image classification and object detection tasks on the ImageNet and COCO [66] datasets, involving channel pruning every 25 iterations. The pruning pipeline was implemented in PyTorch.

Pruning object detection models presents unique challenges compared to image classification due to their larger input size and complex network architectures. While many pruning methods based on BN scaling parameters are not directly applicable, the proposed method overcomes this limitation. It offers a general importance estimation technique and introduces layer grouping to prune coupled channels, enabling its application in both image classification and object detection tasks. To thoroughly validate the effectiveness of their method, the authors conducted extensive pruning experiments on various object detection frameworks, including Retinanet, FSAF, ATSS, PAA, and Faster RCNN. The results, for Retinanet, are presented in Table 3.2 as they are relevant to the experiments in this study. The results demonstrate that their approach, which combines importance normalization with memory reduction and pruning of coupled channels, achieves the highest efficiency. Remarkably, the method effectively prunes object detection networks without sacrificing average precision. In fact, in some cases, the pruned models even outperform the unpruned baselines in terms of mean average precision (mAP). Furthermore, this method offers practical inference speedup.

Model	AP	GFLOPs	Memory	Inference Time
Retinanet	36.5	238.5	297.4	39.73
Retinanet-I	36.3	119.2	246.1	28.49
Retinanet-M	36.5	119.2	190.2	25.36
Retinanet-F	36.8	119.2	254.1	30.08
Retinanet-U	35.8	119.2	244.2	28.53

Table 3.2: -I shows the results of pruning only the internal layers, -M is full method which normalizes importance scores by memory reduction and prunes coupled channels via layer grouping, -F normalizes the importance scores with FLOPs reduction and -U uses the raw Fisher information without normalization.

3.2.1.2 Anchor Pruning

In the context of object detection using one-stage anchor-based detectors, the paper by Bonnaerens et al. [12] introduces a novel technique called anchor pruning. While pruning techniques have been commonly employed to reduce the computational burden of convolutional neural networks, their focus has primarily been on optimizing the backbone networks, which typically entail the majority of computations. However, this work takes a different approach by presenting an additional pruning method specifically tailored for object detection, namely anchor pruning. With the current trend of utilizing object detectors in embedded systems, where post-processing steps like non-maximum suppression can be a bottleneck, the significance of the anchors employed in the detection head is gaining prominence. Through their research, Bonnaerens et al. demonstrate that many anchors in the object detection head can be eliminated without any detrimental impact on the overall detection accuracy, thus offering a promising means of enhancing efficiency and resource utilization in object detection systems. The authors meticulously evaluate the effectiveness of their method by subjecting it to extensive testing on two widely used object detectors, SSD300 and Retinanet. The evaluation is performed using the challenging MS COCO [66] and PASCAL VOC datasets, providing comprehensive insights into the method's performance and applicability.

Redundant Anchors In a single-stage anchor-based object detector, we can determine the total count of predicted bounding boxes per image, denoted as N, through the following calculation.:

$$N = \sum_{i=1}^{k} N_i = \sum_{i=1}^{k} A_i \times H_i \times W_i \tag{3.4}$$

where:

- N represents the overall count of predicted bounding boxes within an image.
- k is number of feature map layers
- N_i represents the number of predicted bounding boxes at the i_{th} feature map layer.
- A_i represents the number of anchor boxes used at each spatial location on the i_{th} feature map layer.
- H_i and W_i represent the height and width of the i_{th} feature map.

To break down the calculation, the object detector processes the image through multiple feature map layers (k in total). For each feature map layer, there are A_i anchor boxes placed at each spatial location. The total number of spatial locations on the i_{th} feature map is $H_i \times W_i$. Hence, the number of predicted bounding boxes at the i_{th} feature map layer is $A_i \times H_i \times W_i$. Finally, the total number of predicted bounding boxes N is the sum of all predicted bounding boxes across all feature map layers. The total amount of predicted bounding boxes can also be influenced by the number of classes C in the dataset. However, most recent models separate the bounding box regression (predicting the coordinates of the boxes) from the object classification (predicting the object class). Therefore, C is not included in the formula for N here [12].

Table 3.3 displays the relationship between the count of anchors per feature map layer and the overall count of bounding boxes. It also shows the corresponding accuracy results on the MS COCO [66] dataset for each SSD300 model. The first row represents the outcomes achieved by applying the six anchors per layer, as specified in the original SSD paper's scale and aspect ratios section [22]. The second row includes the outcomes for the final SSD300 version as reported in the original paper's experimental results section, where the number of anchors is reduced from 6 to 4 in the first and last two layers. Although the original paper does not provide a specific rationale for why certain layers have only four anchors instead of six, the table results suggest that this choice might have been motivated by the observation that having six anchors in the first layer leads to slightly lower accuracy at a higher computational cost. Remarkably, the last row in the table indicates that further decreasing the number of anchors in the first layer could potentially enhance both accuracy and efficiency even further.

$A_1, A_2, A_3, A_4, A_5, A_6$	N	COCO mAP
{6,6,6,6,6,6}	11640	25.6
{4,6,6,6,4,4}	8732	25.7
{2,6,6,6,4,4}	5844	25.8

Table 3.3: The table illustrates the correlation between number of anchors, number of bounding boxes and mAP of SSD300 model [12].

As we highlighted earlier, determining the optimal number of anchors remains a hyper-parameter requiring meticulous tuning. This example serves as evidence that a higher number of anchors does not necessarily translate to improved accuracy. The authors of the anchor pruning method observe that various anchors often predict similar bounding box shapes. Furthermore, certain anchors generate bounding boxes that exhibit substantial overlap with the predictions of neighboring anchors. This observation provides an intuitive explanation for the possibility of eliminating specific anchors without compromising overall performance. These anchor candidates that show significant re-

dundancy become prime targets for the authors to prune from the model, allowing for more efficient and streamlined object detection while maintaining high accuracy.

Optimal anchor search Unlike conventional pruning methods, anchor pruning requires decisions on both the amount of pruning and the specific anchors to prune. Anchors with significant overlap from neighboring anchors are preferable for pruning, while anchors producing unique shapes are retained for accurate object detection. The number of potential anchor subsets resulting from model pruning is given by $|P(A)| = 2^{|A|}$, where A represents the set of all anchors in the unpruned model, and $P(A)$ denotes the power set of A [12].

To grasp this concept, let's examine an instance: Suppose the original set of anchors A contains 3 anchor boxes: A = anchor1, anchor2, anchor3. The power set $P(A)$ will have $2^3 = 8$ subsets, which are P(A) = { {}, {anchor1}, {anchor2}, {anchor3}, {anchor1, anchor2}, {anchor1, anchor3}, {anchor2, anchor3}, {anchor1, anchor2, anchor3}}. Each subset in the power set represents a different combination of anchor boxes from the original set. The empty set {} represents the scenario where no anchor is included, and {anchor1, anchor2, anchor3} represents the complete set of all anchors. The other subsets represent various combinations of individual or multiple anchors.

Given the large number of anchors (30 for SSD and 45 for Retinanet), evaluating all possible combinations becomes impractical. To address this, anchor pruning proposes a more efficient approach using a greedy search algorithm to explore the search space. You can find a visual representation of this process in Figure 3.7.

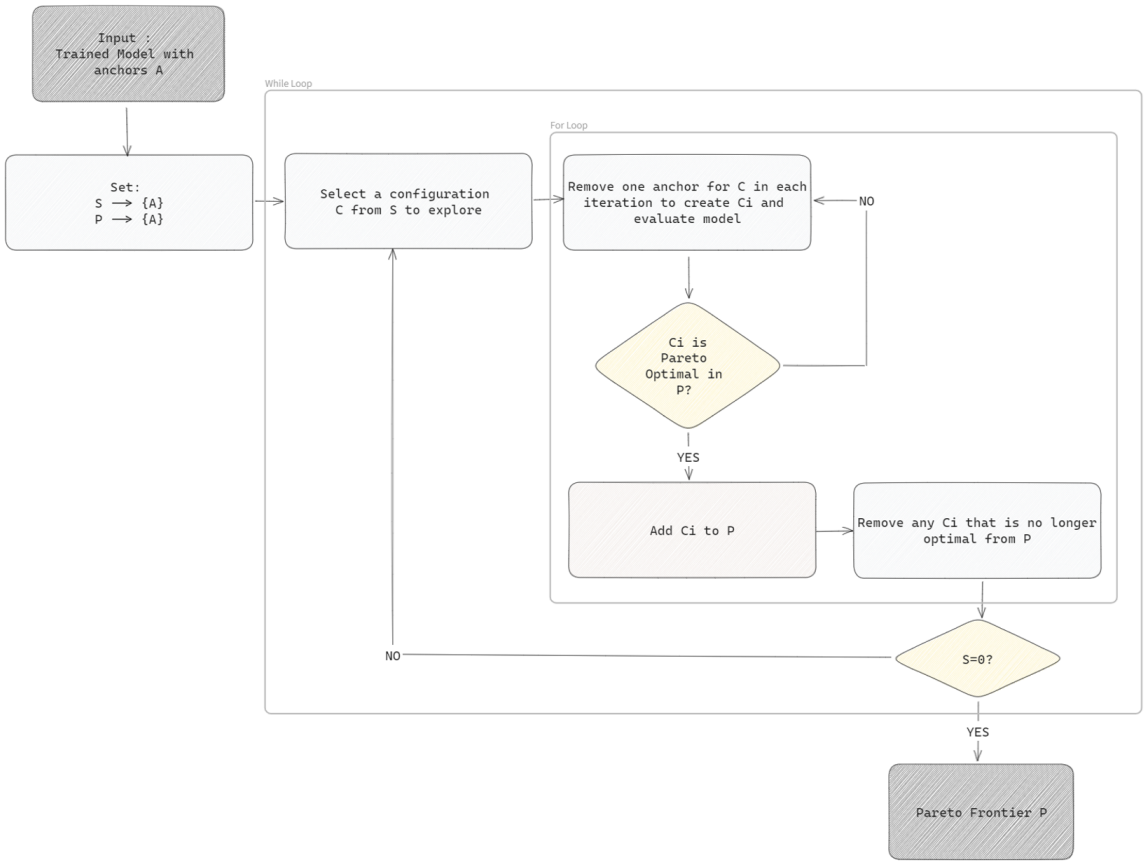


Figure 3.7: Anchor pruning search algorithm.

The anchor pruning algorithm commences with a fully trained model M, equipped with anchor configuration $C = A$. During each iteration, the algorithm explores an adaptive set of unexplored anchor configurations denoted by S and constructs a collection of Pareto-efficient anchor configurations represented by P. The process initiates with $S = \{S\}$ and $P = \{A\}$, and at each step, an unexplored configuration C where $C \subseteq A$ is selected from S, continuing until S becomes empty. From the anchor configuration C, we evaluate various configurations $C_1....C_n$, where $C_i = C \backslash a_i$ (C with the removal of the i_{th} anchor), and $a_i \in C$.

The evaluation of a configuration C involves assessing the accuracy of model M using only the predictions made by anchors $a_i \in C$. To streamline this evaluation, we initially store all predicted bounding boxes from M before any are discarded by post-processing steps like NMS (Non-Maximum Suppression). The re-evaluation process then focuses only on the bounding boxes produced by anchors a_i in the configuration C. This method allows us to avoid the extensive computation required for running all validation inputs through the model for each new configuration. Accuracy evaluation should be performed on a validation set to prevent overfitting the anchor configurations

on the test set. The study emphasizes not only achieving the highest accuracy but also considering the trade-off between accuracy and resources. Consequently, both accuracy metrics (e.g., mean average precision (mAP), F-score, etc.) and resource metrics (e.g., inference speed, FLOPs, model size, number of predicted bounding boxes, etc.) are evaluated. Resource metrics that can be calculated, such as FLOPs, are favored over those that require experimental measurements, such as inference speed [12].

A configuration is considered Pareto-optimal or Pareto-efficient when no other node using fewer or equal resources attains higher accuracy. Throughout the search phase, P stores the evaluated Pareto-optimal configurations. To accomplish this, we assess the accuracy and resource cost of C_i in comparison to all $C_j \in P$. If C_i is Pareto-optimal in P, it is added to P and S, and any C_j which is not Pareto-optimal is eliminated from P. To expedite the search algorithm's runtime, an accuracy threshold θ could be incorporated. This ensures that C_i is added to P and S only if it is Pareto-optimal in P and its accuracy $accuracy_{C_i}$ is greater than or equal to θ. This is particularly useful in practice when the primary interest lies in exploring accuracy/resources trade-offs while maintaining an acceptable level of accuracy. Once S becomes empty, P contains all the Pareto-efficient anchor configurations achieved during the process.

Experiments Authors present the experiments and results of the proposed anchor pruning technique, which showcases its versatility in being applicable to various anchor-based one-stage object detectors. They demonstrate their findings primarily on SSD300 [22], a widely used and influential one-stage object detector. SSD300 utilizes a VGG-16 backbone, a pyramid of convolutional feature maps in the neck, and a head comprising a $3 \times 3 \times (A_i \times (Classes + 4))$ convolution on top of each of the 6 feature maps. To assess the performance, the authors utilized the FLOPs metric for comparison.

The initial experiments focused on SSD300, which allowed the authors to evaluate the impact of anchor pruning on the model's efficiency and accuracy. By selectively pruning redundant anchors, they achieved notable improvements in the FLOPs metric without compromising the overall detection accuracy.

Following the successful results obtained with SSD300, the authors proceeded to report the experiments with Retinanet. Retinanet employs a different architecture from SSD300, featuring 4 additional $3 \times 3 \times 256$ convolutional layers before applying a $3 \times 3 \times (A_i \times (Classes + 4))$ convolution in the detection head. Additionally, the prediction layers in the detection head are reused across all feature maps. By applying the anchor pruning approach to Retinanet, the authors explored its potential for optimizing this architecture as well. The results highlighted the adaptability of the method, demon-

strating significant reductions in FLOPs while maintaining accurate object detection.

Figure 3.8 displays the Pareto frontier of pruned anchor configurations achieved through search algorithm. These configurations represent the points where reducing FLOPs can only be achieved at the expense of accuracy. The figure also includes the original unpruned model for comparison, as well as the results obtained by randomly pruning anchors. The method consistently outperforms random pruning across all FLOPs, underscoring its effectiveness in identifying and pruning redundant anchors that lead to minimal accuracy loss.

In cases where pruned models experience a significant reduction in FLOPs and require more anchors to be pruned, the selection of the appropriate anchors becomes crucial. It can make the difference between a model with acceptable accuracy and one that becomes unusable due to a considerable drop in accuracy. The wider gaps observed on the x-axis within the Pareto frontier are attributed to pruned anchors in the first layer of the head, which have a substantial impact on the number of FLOPs.

Notably, the search algorithm successfully identifies the most accurate configuration, referred to as Configuration-A pruned, which achieves an impressive 15% reduction in FLOPs within the detection head without any compromise in accuracy. This achievement highlights the effectiveness of our approach in significantly reducing computational overhead while maintaining high accuracy in object detection.

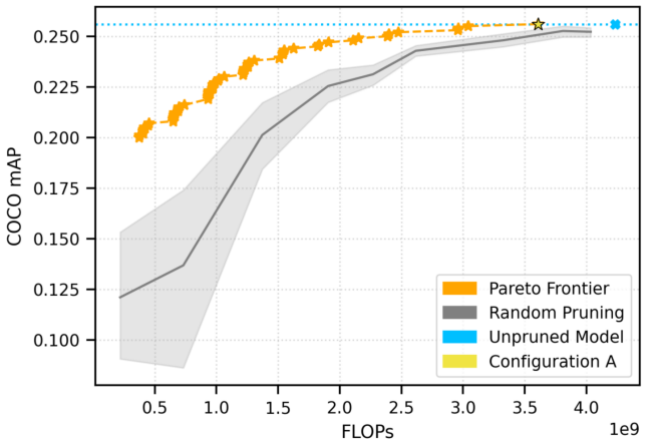


Figure 3.8: The Pareto chart is indicating Accuracy and FLOPs trade-offs for different pareto-optimal configurations. The anchor configurations found by greedy anchor search algorithm [4].

Regarding Retinanet, the authors assert that their approach achieves a remarkable reduction in the FLOPs (floating-point operations) of the entire Retinanet model by 44% with only 1.8% decrease in accuracy. This represents a notable 3 $\times$ improvement in compression efficiency compared to the default anchor scaling method, denoted as Reti-

nanet(s1a1), which only reduces FLOPs by 14% but incurs a larger accuracy drop of 5.9%. These findings underscore the significance of customizing anchor pruning to the specific object detection architecture. Notably, allowing an entire layer to be dropped plays a critical role in achieving substantial FLOP reduction. The training configurations adopted for Retinanet are built upon the MMDetection by Chen et al. [88], using the Retinanet baseline configuration with a ResNet50 backbone and 1 $\times$ learning rate schedule. Table 3.4 presents a comparison of the retrained FLOPs frontier configuration with both the baseline and an alternative Retinanet versions that uses only one and three anchor in each layer, as reported in the original paper.

Model	AP .5:.95	FLOPs head	FLOPs total	inf time
Retinanet(s3-a3)	36.9	129B	224B	137 ms
Retinanet(s1-a1)	31.0	98B	193B	99 ms
Pruned	33.9	31B	126B	79 ms
Re-trained after prunning	35.1	31B	126B	79 ms

Table 3.4: Comparing Accuracy, Computational Complexity (FLOPs), and Inference Time for Retinanet Configurations [4].

3.2.2 Quantization

Quantization has a rich history with connections to the foundations of calculus dating back to the early 1800s. When we apply quantization to Neural Networks, we encounter both challenges and opportunities. Firstly, the computation demands in Neural Network inference and training are substantial. This implies that we must be mindful of how we represent the numerical values of weights and activations within the network. Modern Neural Network models often have many parameters. Interestingly, we can reduce the number of bits used to represent these parameters without significantly impacting accuracy. This is because the models are often over-parameterized, meaning they have more parameters than necessary. Moreover, these models are structured in layers, and each layer contributes differently to the overall loss function. This suggests that a mixed-precision approach to quantization, where different layers are assigned different levels of precision, could be beneficial. Quantization involves finding efficient ways to represent numbers in the face of heavy computational demands, leveraging the over-parameterization of modern models [5].

3.2.2.1 Basic concept of quantization

To describe the quantisation technique in neural network, first we need to set some notations. Let's consider a scenario where the neural network consists of L layers, each

with adjustable parameters, the overall set of these parameters is denoted as θ. The goal is to minimize this loss:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} l(x_i, y_i; \theta) \tag{3.5}$$

The pair (x, y) represents the given input data and its corresponding label. The function $l(x, y; \theta)$ is utilized as the loss function, for instance, it could be Mean Squared Error or Cross Entropy loss. The variable N corresponds to the total count of data points. The provided formula explains how a neural network is trained in a supervised approach, notably highlighting the significance of the term θ, which signifies the network's parameters. Additionally, we use h_i to represent the hidden input activations of the i^{th} layer, and a_i to represent the resulting hidden output activation.

Assuming that the model's parameters θ have been trained and are stored using floating point precision, the focus in quantization is to decrease the level of precision for both the parameters θ and the intermediate activation maps h_i, a_i. This reduction in precision aims to have minimal impact on the model's ability to generalize and maintain accuracy. To achieve this, it becomes necessary to establish a quantization operator that converts a floating-point value into a quantized version [5].

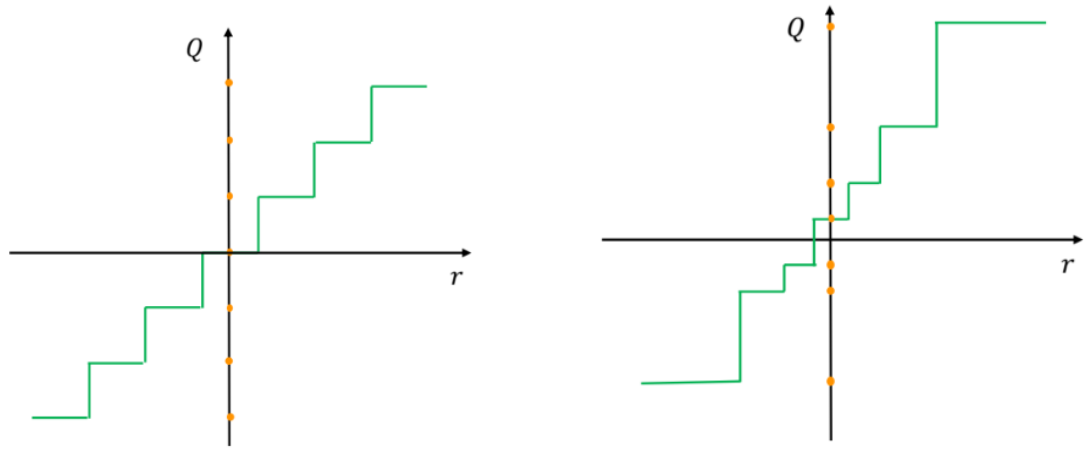
Our first step involves defining a function to quantize NN weights and activations into a finite set of values. This function accepts real values in floating point format and converts them into a reduced precision range. A frequently employed choice for this quantization function is as follows:

$$Q(r) = Int(r/S) - Z \tag{3.6}$$

Here, Q represents the quantization operator, r stands for a real-valued input which is activation or weight, S denotes a scaling factor, and Z represents an integer zero point. Additionally, the Int function transforms a real value into an integer value using rounding mechanisms like rounding to the nearest integer or truncation. In essence, this function creates a connection between real values in r and corresponding integer values.

This quantization approach is commonly referred to as uniform quantization. In this technique, the resulting quantized values (also known as quantization levels) are evenly spaced, as shown in Figure 3.11a. However, there are also non-uniform quantization methods where the quantized values are not necessarily evenly spaced, as depicted in Figure 3.11b. rounding operation:

$$\hat{r} = S(Q(r) + Z) \tag{3.7}$$



(a) Uniform quantization (b) Non-uniform quantization

Figure 3.9: Uniform and non-uniform quantization. Continuous real values within the domain r are transformed into discrete, reduced precision values within the quantized domain Q. Adapted from [5].

We can reverse the equation 3.6 in the opposite direction and retrieve the real values from Q(r). This recovery process is often referred to as dequantization. We denote the dequantized values as $\hat{r}$ while recovered values do not match to because of mentioned. Essentially, this scaling factor divides a specific range of real values in r into several distinct partitions [39] [38]:

$$S = \frac{\beta - \alpha}{2^b - 1} \tag{3.8}$$

the range $[\alpha, \beta]$ is clipping range. Here, clipping means that we limit values within this interval, and b is quantization bit width. Hence, for the scaling factor to be defined, it's essential to establish the bounds of the clipping range $[\alpha, \beta]$. This determination of the clipping range is often known as calibration.

One simple choice for calibration is to use the minimum and maximum values of the signal itself, resulting in $\alpha = r_{min}$ and $\beta = r_{max}$. This creates an asymmetric quantization scheme since the clipping range might not be symmetric around the origin, meaning that $-\alpha \neq \beta$, as shown in Figure 3.9 (Right). Alternatively, a symmetric quantization scheme can be achieved by selecting $\alpha = -\beta$, which can be based on the signal's min/max values: $-\alpha = \beta = max(|rmax|, |rmin|)$. Asymmetric quantization usually leads to a narrower clipping range compared to symmetric quantization. This distinction becomes particularly relevant when dealing with imbalanced target weights

or activations, such as activations after Rectified Linear Unit (ReLU) that are always non-negative.

On the other hand, using symmetric quantization simplifies the quantization function by setting $Z = 0$ in place of the zero point.

$$Q(r) = Int(r/S) \tag{3.9}$$

there are two possible choices for determining the scaling factor.

1. Full Range Symmetric Quantization: In this approach, the scaling factor S is chosen as $2 \times \frac{max(|r|)}{2^n - 1}$ (using floor rounding mode), where n represents the number of bits. This choice is made to utilize the entire range of INT8 values, which spans from -128 to 127. Essentially, this scaling factor ensures that the quantized values can fully cover the range of possible values within the chosen bit-width.

2. Restricted Range Symmetric Quantization: In this scenario, the scaling factor S is selected as $\frac{max(|r|)}{2^{n-1} - 1}$ which results in the utilization of a narrower range of values, spanning from -127 to 127. This approach intentionally restricts the quantized range, sacrificing some range coverage for computational efficiency and other considerations.

The Full Range approach generally provides greater accuracy since it spans a wider range of values, offering more precision. This is particularly common when quantizing weights. In practice, symmetric quantization is often preferred for weight quantization due to its advantages in terms of computational efficiency during inference and simpler implementation. Symmetric quantization divides the clipping process with an evenly balanced range. This approach is advantageous in terms of simpler implementation, nevertheless, it may not be the best choice for situations where the range is skewed and lacks symmetry. In such instances, asymmetric quantization is more suitable. Figure 3.10 is illustration of symmetric quantization and asymmetric quantization.

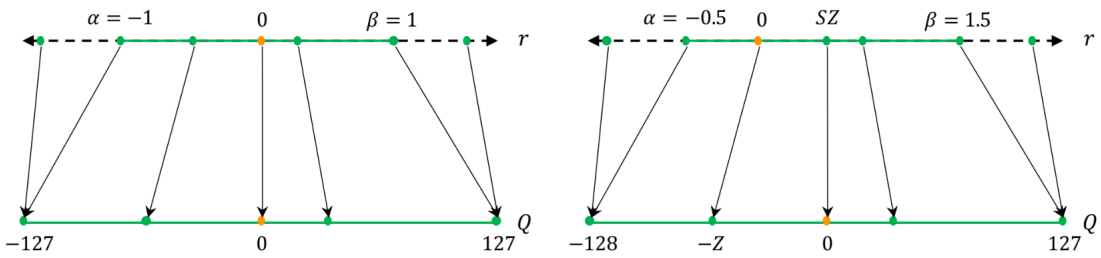
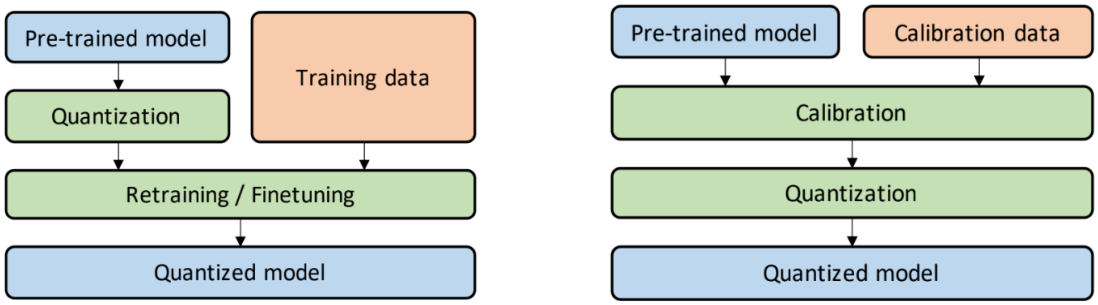


Figure 3.10: Illustration of symmetric quantization and asymmetric quantization [5].

As outlined in the work [89] by Nagel et al., there is a common need to modify the neural network's parameters following quantization. This adjustment can take two routes:

1. Quantization-Aware Training (QAT): This involves retraining the model while taking into account the quantization process.
2. Post-Training Quantization (PTQ): Here, quantization is made without re-training.

A visual comparison of these two methods is presented in Figure 3.11.



(a) Quantization-Aware Training (b) Post-Training Quantization

Figure 3.11: In the case of Quantization-Aware Training (QAT), an initially trained model undergoes quantization and is subsequently fine-tuned using training data. This fine-tuning is essential to recalibrate the parameters and mitigate the accuracy loss that can occur due to quantization effects. On the other hand, with Post-Training Quantization (PTQ), a pre-trained model is initially calibrated utilizing calibration data, often derived from a small subset of the training dataset. This calibration process is employed to compute the appropriate clipping ranges and scaling factors. After calibration, the model undergoes quantization based on the determined calibration outcomes [5].

Quantization-Aware Training: Given a trained model, the act of quantization can introduce a form of alteration to the model's learned parameters. This alteration has the potential to drift the model away from the convergence point it reached during training with floating point precision. To tackle this effect, a feasible approach involves re-training the neural network using quantized parameters. Through this process, the model can gradually converge to a state characterized by improved loss.

A commonly adopted technique for this purpose is known as Quantization-Aware Training (QAT). In QAT, the customary forward and backward passes are executed on the quantized model using floating point computations. However, during each gradient update, the model parameters are quantized [90] [91] [92] [93].

Post-Training Quantization: A cost-effective substitute for the resource-intensive QAT technique is Post-Training Quantization (PTQ). This approach achieves quantization and weight adjustments without the need for fine-tuning. In contrast to QAT, which necessitates a substantial quantity of training data for the purpose of retraining, PTQ

boasts an added advantage. It can be implemented effectively even when data availability is scarce or when data is unlabeled. However, this advantage typically arrives with a trade-off in terms of reduced accuracy compared to QAT, particularly when dealing with low-precision quantization [41] [94] [40] [95] [96] [97] [98] [99] [43] [100] [42].

3.2.2.2 Quantizing Object detection models: Zero Shot Quantization Framework

For quantizing object detection models, numerous approaches have been put forth to achieve a suitable balance between model efficiency and performance. Google introduced an 8-bit quantization technique [38], which resulted in notable enhancements concerning the balance between latency and accuracy in the context of MobileNets [101]. These improvements were observed across tasks such as ImageNet classification [102] and COCO object detection. In their publication, Wei et al. [103] employed activation quantization within object detection models with the aim of facilitating knowledge transfer from larger to more compact models. In another work Fully Quantized Network for Object Detection [104] the researchers introduced innovative techniques to stabilize the fine-tuning stage of the quantization process for neural networks. This approach effectively addresses challenges in achieving accurate quantization for intricate tasks like object detection. Applying these techniques to develop fully quantized 4-bit detectors based on Retinanet and Faster RCNN architectures, the study showcased exceptional performance, achieving an impressive over 3.8 $\times$ reduction in mean average precision (mAP) loss compared to existing methods. However, for the purpose of this research, we have chosen to employ the ZeroQ quantization method [6]. This particular approach stands out as a comprehensive framework that adeptly navigates the intricate trade-offs between model compression and preservation of detection accuracy. Its versatility and promise in achieving noteworthy results make it a compelling choice for our study.

ZeroQ : Zero Shot Quantization Framework Consider the loss function presented in equation 3.5 for model M. This model is trained using a dataset comprising N data points. The input data undergoes standard preprocessing that involves normalization, aiming to attain a mean μ_0 of 0 and a unit variance σ_0 of 1.

Furthermore, the model incorporates L Batch Normalization BN layers, denoted as BN_1, BN_2, and so forth. Prior to reaching these BN layers, the activations are denoted as z_i, which correspond to the outputs of the i^{th} convolutional layer. During the inference phase, these z_i activations undergo normalization utilizing the running mean μ_i and variance σ_i^2 parameters, which are specific to their corresponding BN layer (BN_i). It's

worth noting that these parameters are pre-computed during a preceding training phase. It is assumed that all the parameters and activations are stored with 32 bit precision. In ZeroQ method asymetric uniform quantization is utilized.

ZeroQ framework supports two types of quantization: fixed-precision and mixed-precision. As discussed, precision refers to the number of bits used to represent a numerical value. Fixed-precision quantization allocates the same number of bits to all the layers in a neural network. On the other hand, mixed-precision quantization allows different layers of the model to have different bit precisions. For instance, one layer might use 2 bits, another might use 4 bits, and yet another could use 8 bits. The key concept behind mixed-precision quantization is to allocate higher bit precision to layers that are more sensitive to quantization errors while aggressively quantizing layers that are less sensitive, all without significantly increasing the overall size of the model.

This approach becomes particularly useful when aiming for extremely low bit precision settings, such as 4-bit quantization, while still maintaining a high level of accuracy. The framework typically considers three choices for bit precisions (k) for each layer: 2, 4, 8 bits. It's important to note that since each layer can have one of these bit precision settings, the search space (all the possible combinations of bit precision choices for all layers of NN) becomes exponentially large, making it challenging to explore all possibilities.

However, the ZEROQ framework proposes a solution to address this challenge. Instead of exhaustively searching through all possible combinations, the framework suggests measuring the sensitivity of the model to quantization for each layer. Sensitivity refers to how much the model's performance is affected by reducing the bit precision of a particular layer. This sensitivity measurement helps identify which layers should be assigned higher bit precision to maintain accuracy, and which layers can be more aggressively quantized [105] [106] [107].

For post-training quantization, which involves quantizing a pretrained model without additional fine-tuning, the framework proposes using a Kullback-Leibler (KL) divergence sensitivity metric. KL divergence is a mathematical measure of how one probability distribution (e.g., the output of the original model) differs from another (e.g., the output of the quantized model). By calculating KL divergence between the original and quantized models, it's possible to estimate the impact of quantization on the model's performance. This information then guides the decision-making process to allocate the appropriate bit precisions to different layers, maximizing accuracy while still achieving efficient quantization. So the sensitivity of i^{th} layer , denoted as Ω_i can be represented as:

$$\Omega_i(k) = \frac{1}{N} \sum_{j=1}^{N_{dist}} KL(M(\theta; x_j)), (M(\hat{\theta}_i(k - bit); x_j)) \tag{3.10}$$

This process is illustrated in Figure 3.12 for the ResNet18 model.

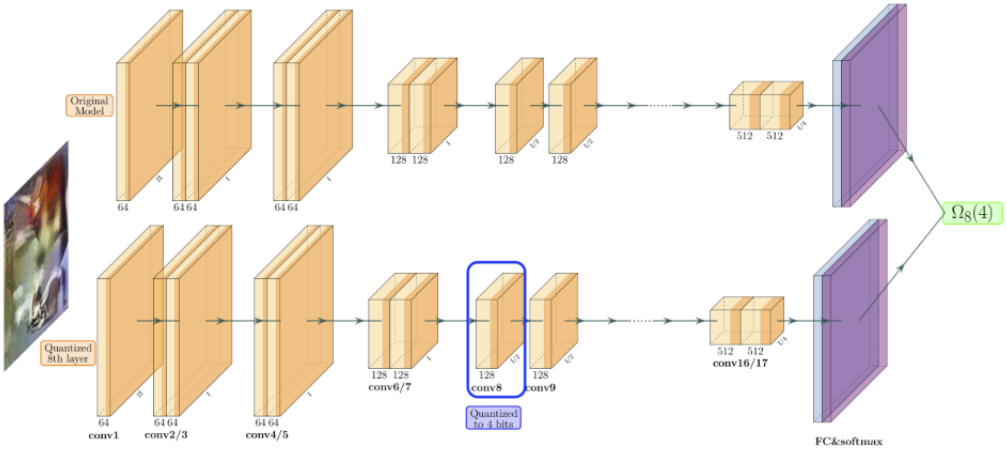


Figure 3.12: The sensitivity of the 8th layer when it's quantized to 4 bits ($\Omega_8(4)$) is calculated as shown in the figure. Distilled Data is inputted into both the full-precision ResNet18 model (at the top) and the same model with only the 8_{th} layer quantized to 4 bits (at the bottom). The sensitivity of the 8_{th} layer when it's quantized to 4 bits (($\Omega_8(4)$) is determined by measuring the KL-divergence between the outputs of these two models [6].

The challange to calculate the sensitivity is the fact that we do not have access to original training x_j. To overcome this challange , ZeroQ method utilizes distilled data to generate artificial input data that emulates the statistical characteristics of the original training dataset. The Distilled Data is derived exclusively by examining the trained model without requiring access to the actual training dataset.

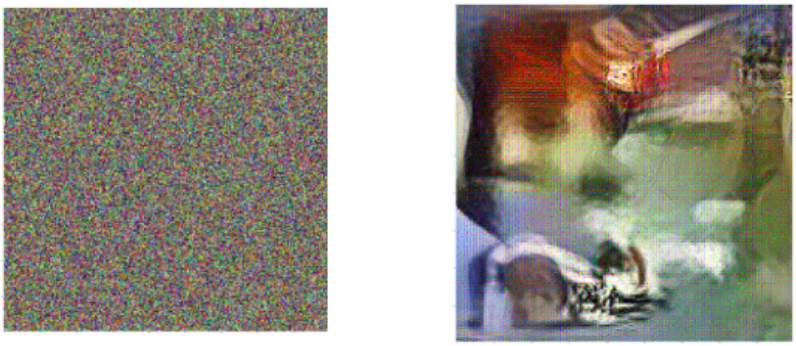
The need for distilled data arises from another challenge: we have to know the range of values that the activations in each layer can take. This range is important for the quantization process, where we limit the values within a certain range ($[a, b]$ range mentioned earlier). However, figuring out this range is not possible without access to the training data. One simple but not very effective way to tackle these issues is by creating random input data using a Gaussian distribution with zero average and standard deviation of one, and then feeding it into the model. However, this approach falls short in capturing the accurate characteristics of the activation data that corresponds to the original training dataset. The Gaussian data fails to accurately capture how sensitive the model is to certain variations. To solve this problem, the ZeroQ method suggests an innovative approach: distilling input data from the neural network model itself. This means generating synthetic data that is carefully designed based on the properties of the neural network. The idea is to solve an optimization problem to learn a distribution for input

data that closely matches the statistical patterns stored in the Batch Normalization (BN) layer of the model. In more detailed terms, the ZeroQ approach formulates and solves a specific optimization problem to achieve this task.

$$\min_{x^r} \sum_{i=0}^{L} ||\hat{\mu}_i^r - \mu_i||_2^2 + ||\hat{\sigma}_i^r - \sigma_i||_2^2 \tag{3.11}$$

Equation 3.11 is an optimization problem, which by solving it we can distill input data, the distilled data here is denoted as x^r, mean and standard deviation of data for layer i are μ_i and σ_i, and $\hat{\mu}$ and $\hat{\sigma}$ for distilled data. The authors have taken an interesting step by visually comparing the distilled data obtained from ResNet50 with randomly generated Gaussian data. This figure is provided once again in Figure 3.13 to clearly demonstrate the concept of distilled data.

A visual comparison of these two methods is presented in Figure 3.11.



(a) Visualization of Gaussian data

(b) Visualization of distilled data

Figure 3.13: The visualization illustrates that the distilled data reveals more distinct local patterns and structures [6].

Now that we have obtained the distilled data, we can utilize Equation 3.10 to effectively narrow down the search space for mixed-precision quantization. As an illustrative example, considering ResNet50 with only three available bit precisions: 2, 4, 8, the initial search space contains a substantial number of configurations, approximately 7.2×10^{23}. The central concept is to employ higher bit precision for layers that exhibit greater sensitivity, while utilizing lower bit precision for layers with less sensitivity. This approach establishes a relative ranking of bit quantities. To precisely determine the suitable bit precision settings, the authors employ a method based on the Pareto frontier.

Let us assume a target size for our quantized model, denoted as S_{target}, our objective is to determine an optimal manner of assigning distinct bit precisions to different layers of the model, such that the resulting model size aligns precisely with S_{target}. To accomplish this goal, we start by assessing the model's overall sensitivity using different

combinations of bit precisions that match the desired S_{target} size. Each combination represents a unique way of distributing bit values among different components of the model. The primary aim is to minimize the model's sensitivity. In simpler terms, we seek to identify the combination of bit precisions that exerts the least influence on the model's performance during quantization. This is where the Pareto frontier method proves valuable. The Pareto frontier approach assists in selecting the configuration of bit precisions that achieves minimal sensitivity, while simultaneously maintaining the desired model size. This process finds the most suitable trade-off between precision and size, resulting in the most accurate quantized model possible.

Experiments The researchers of this technique, ZeroQ, conducted thorough testing on a diverse set of models and datasets. These models included ResNet18, ResNet50, MobileNet-V2, and ShuffleNet, all evaluated on the ImageNet dataset. Additionally, they examined Retinanet using the Microsoft COCO dataset [66]. For the purpose of illustrating their findings in this study, we will focus on their results for Retinanet.

Retinanet, compared to the previously studied neural networks on ImageNet, has a distinct characteristic. Some of the convolutional layers in Retinanet are not immediately followed by Batch Normalization (BN) layers. This difference arises due to the presence of a feature pyramid network (FPN) [68]. Consequently, the number of BN layers is slightly fewer than the number of convolutional layers. Nonetheless, as authors claim, this distinction doesn't get in the way of the ZeroQ framework's effectiveness. In particular, the researchers extracted the backbone structure of Retinanet and derived Distilled Data from it. Subsequently, they input this Distilled Data into Retinanet to assess its sensitivity and determine the range of activations across the entire neural network. Following this, they proceeded with optimizing the Pareto Frontier, a concept previously discussed. The outcomes of their experimentation are presented in Table 3.5.

Method	W-bit	A-bit	Size(MB)	mAP
Baseline	32	32	145.1	36.4
ZeroQ	MP	8	18.13	33.7
ZeroQ	MP	6	24.17	35.9
ZeroQ	8	8	36.25	36.4

Table 3.5: Object detection on Microsoft COCO using RetinaNet. W-bit and A-bit refers to a quantization configuration. For example for W-bit = 32 and A-bit=32, weights are quantized to 32 bits and activations are quantized to 32 bits as well [6].

3.2.3 Knowledge Distillation

First introduced in 2015, Knowledge Distillation stands as a transformative technique at the crossroads of model compression and transfer learning. Since its inception, this

methodology has gained considerable attention, leading to the formulation of numerous strategies aimed at enhancing and customizing its foundational principles. These innovations have pushed the expansion of Knowledge Distillation beyond its initial form, often referred to as the vanilla Knowledge Distillation method.

Knowledge distillation can be classified according to the types of knowledge being utilized . In vanilla distillation the teacher knowledge is derived by utilizing the logits of an expansive, deep model [45] [46] [47] [48]. This type of knowledge is referred to as Response-based knowledge which utilizes the neural response of the output layer of the teacher model. The primary objective here is to closely replicate the teacher model's ultimate prediction. Other type of knowledge which can be employed is the activations, neurons or features of intermediate layers to guide the learning of the student model [49] [50] [51] [52] [53]. This is called feature-based knowledge. The relationship between activations and neurons contains valuable information known as relation-based knowledge [55] [56] [57] [58] [59]. Both response-based and feature-based knowledge draw upon the outputs of particular layers within the teacher model. Relation-based knowledge, on the other hand, delves deeper into the connections between distinct layers or individual data samples [7]. The various knowledge categories present within a teacher model are depicted in Figure 3.14

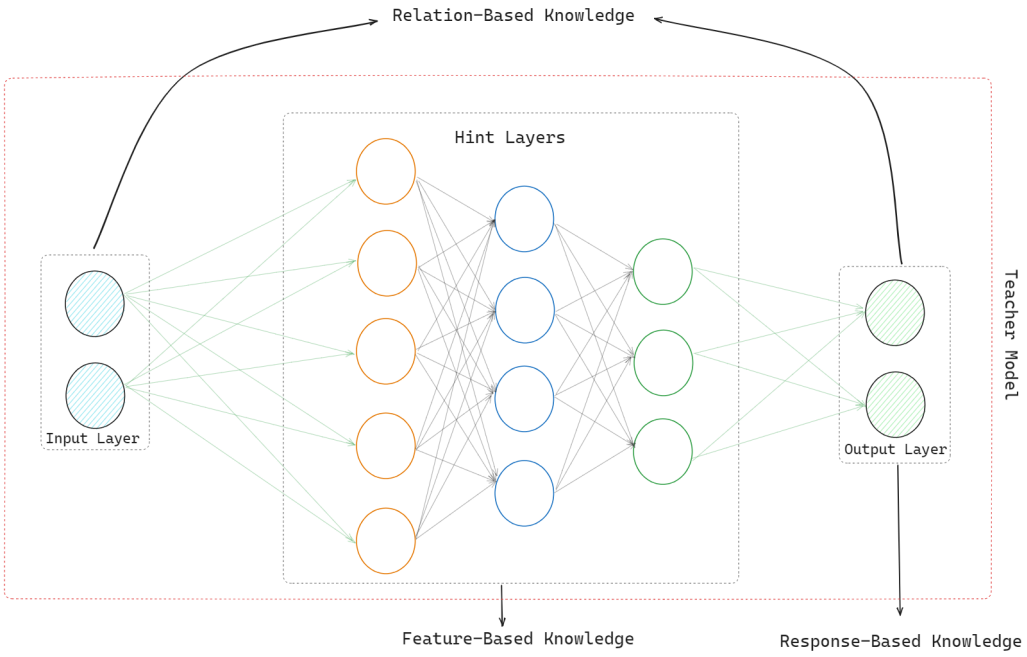


Figure 3.14: The visual representations of origins for response-based knowledge, feature-based knowledge, and relation-based knowledge within a deep teacher network based on [7].

This section extensively explores the various approaches that have emerged in the field of Knowledge Distillation for object detection. These methodologies involve the com-

bination of various forms of knowledge as detailed earlier. The primary objective of these approaches is to identify suitable techniques for Knowledge Distillation in the context of object detection, aimed at addressing the unique challenges encountered by object detection models.

3.2.3.1 Vanilla Knowledge Distillation

The concept of knowledge distillation was first introduced in the research paper [45] authored by Hinton et al. The authors proposed a method to address the disparity between the requirements of large-scale machine learning during training and deployment phases. To overcome this, the authors proposed concept of distillation, during which they initially train a cumbersome model that could effectively extract structure from the data and then transfer the acquired knowledge from the cumbersome model to a smaller, more deployment-friendly model. A similar strategy had already been explored by Caruana et al. in their seminal work [108], demonstrating that the knowledge obtained from a large model or ensemble of models can be effectively transferred to a single compact model.

The authors argue that while it is widely acknowledged that the training objective should align closely with the user's actual objective, models are typically trained to optimize performance specifically on the training data, even though the ultimate goal is to achieve good generalization to new, unseen data. Ideally, models should be trained to generalize effectively, but doing so requires knowledge about the correct generalization approach, which is often unavailable. In the process of knowledge distillation, when transferring knowledge from a large model to a smaller one, an opportunity arises to train the smaller model to generalize in the same manner as the large model.

A straightforward approach to impart the generalization ability of the cumbersome model (which is called Teacher) to a smaller one (which is called Student) involves employing the class probabilities generated by the teacher model as soft targets during the training of the student model. Soft targets with high entropy offer a wealth of information per training instance compared to hard targets and exhibit lower variance in the gradient across training instances. Consequently, training the student model on such soft targets often necessitates much less data than what was used for the original teacher model, allowing for the use of a significantly higher learning rate.

In further detailing the techniques, Hinton et al. discuss how, in tasks like MNIST, where the teacher model consistently yields highly confident correct answers, crucial information concerning the learned function can be found in the ratios of extremely

small probabilities present in the soft targets. These probabilities might express, for instance, that one version of a digit "2" has a probability of 10^{-6} of being classified as a "3" and 10^{-9} of being classified as a "7" while another version could have the reverse probabilities. This information provides valuable insights into the underlying similarity structure within the data, revealing which "2"s resemble "3"s and which resemble "7"s. However, during the transfer stage, this information has minimal impact on the cross-entropy cost function because the probabilities are exceedingly close to zero. To tackle this challenge, they used logits (the inputs to the final softmax) as training targets for the student model, rather than using the probabilities generated by the softmax. They achieve this by minimizing the squared difference between the logits generated by the teacher model and those produced by the student model.

Introduced approeach, which involves training the student model using both soft targets and correct labels in a dual objective approach, is commonly referred to as vanilla knowledge distillation. In neural networks, class probabilities are commonly generated through a softmax output layer. This layer transforms the computed logit, z_i, for each class into a corresponding probability, q_i, by comparing z_i with the other logits:

$$q_i = \frac{exp(\frac{z_i}{T})}{\sum_j exp(\frac{z_j}{T})} \tag{3.12}$$

In knowledge distillation, the temperature parameter T plays a significant role in shaping the behavior of the softmax function in neural networks. Typically, T is set to 1, resulting in a standard softmax output, where the class probabilities represent the model's confidence in its predictions. However, when a higher value is chosen for T, it induces a softening effect on the probability distribution over classes. This means that the probabilities become less concentrated, providing a more nuanced representation of the model's uncertainty and capturing more subtle relationships between classes.

Consider a teacher network with an output softmax $P_T = softmax(a_T)$, where a_T represents the teacher's pre-softmax activations. S is a student network with W_S parameters. The output probability for student network is $P_S = softmax(a_S)$, where a_S is the student's pre-softmax output. The objective is to train the student network so that its output P_S becomes similar not only to the true labels y_{True} but also to the teacher's output P_T. To account for the possibility that P_T may be very close to the one-hot encoded representation of the true label, a Temperature $T > 1$ is introduced during training. This temperature softens the signal coming from the teacher network's output, providing additional information during the training process. Same Temperature in teacher's output P_T is used for student output P_S. This ensures consistency in the

training process and allows the student network to effectively learn from the teacher's knowledge.

$$P_T^T = softmax(\frac{a_T}{T}), P_S^T = softmax(\frac{a_S}{T}) \tag{3.13}$$

Subsequently, the student network undergoes training to optimize the given loss function:

$$L(W_S) = H(y_{True}, P_S) + \lambda H(P_T^T, P_S^T) \tag{3.14}$$

In the equation, the symbol H represents the cross-entropy, and λ is an adjustable parameter used to balance the two cross-entropy terms. The first term in equation 3.14 denotes the conventional cross-entropy between the student network's output and the true labels. The second term guarantees that the student network acquires knowledge from the teacher network's softened output. See Figure 3.15.

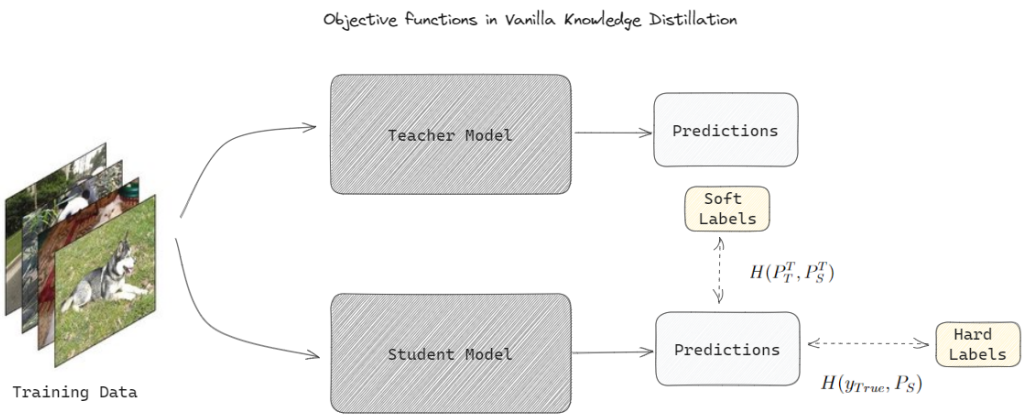


Figure 3.15: Objective functions in Vanilla Knowledge Distillation.

By combining the two loss functions, we encourage the student model to learn from both sources of information: the rich knowledge captured in the softened probabilities from the teacher model and the precise guidance from the correct labels. This combined approach allows the student model to capture the complex relationships present in the data and improves its ability to generalize and make accurate predictions.

3.2.3.2 Hints for thin deep Nets

Depth is essential in representation learning because it encourages the reuse of features and the creation of more abstract and consistent representations in upper layers [109]. However, training deep neural architectures is difficult due to the presence of successive

non-linearities, resulting in highly non-convex and non-linear functions [110] [111]. In their paper [49] Romero et al., propose a novel approach named FitNets to address network compression with knowledge distillation and at the same time, leveraging the advantages of depth. FitNets introduce a modified Knowledge Distillation framework [45], enabling the training of thinner but deeper student models. The method involves using intermediate-level hints from the teacher's hidden layers to guide the training of the student network. The goal is for the student network to learn an intermediate representation that aligns with the teacher's intermediate representations. These hints enable the training of narrower and deeper networks. The suggested technique can achieve similar or even superior results compared to the teacher's approach.

Authors of [49] claim that, while the Knowledge Distillation framework has shown promising results, even with slightly deeper student architectures, they observed that as they further increase the depth of the student network, the training still encounters challenges in optimizing deep networks.

To enhance the training process of deep FitNets, which are characterized by having a greater depth compared to their teacher network, a technique involving the incorporation of hints from the teacher network is introduced by the authors. A hint is essentially the output of a specific hidden layer in the teacher network that serves the purpose of guiding the student's learning process. In a corresponding manner, a particular hidden layer within the FitNet, referred to as the guided layer, is designated to receive guidance from the teacher's hint layer. The intention is for the guided layer to become proficient at predicting the output generated by the hint layer of the teacher.

It is crucial to note that the utilization of hints in this context serves as a form of regularization. As a result, the selection of the hint/guided layer pairing must be made judiciously to ensure that the student network does not become overly constrained by regularization. In summary, the depth of the guided layer is inversely related to the network's flexibility. A deeper guided layer reduces the network's flexibility, potentially increasing the likelihood of over-regularization and disrupting the network's learning capacity. This choice of layer depths has a significant impact on the balance between incorporating teacher guidance and allowing the student network to learn independently.

Considering that the teacher network typically possesses a greater width compared to the FitNet, the chosen hint layer within the teacher network might comprise more output units than the guided layer of the FitNet. To address this discrepancy, a solution involves incorporating a regressor into the guided layer. This regressor is designed to generate an output that matches the size of the hint layer's output. During training, we optimize the parameters of both FitNet, from the initial layer to the guided layer, and the parameters

of the regressor. This optimization is carried out by minimizing a specific loss function. The goal of this approach is to harmonize the dimensions of the output from the guided layer of the FitNet and the hint layer of the teacher network, even when the teacher network has more outputs. By introducing a regressor into the guided layer, the FitNet can accommodate the additional outputs present in the hint layer. This allows for a more effective training process, ensuring that the information provided by the teacher's guidance aligns with the student network's learning dynamics. Loss function to be minimized is as follow:

$$L_{HT}(W_{Guided} - W_r) = \frac{1}{2} ||u_h(x; W_{hint}) - r(v_g(x; W_{Guided}); W_r)| |^2 \tag{3.15}$$

u_h and v_g represent the nested functions of the teacher and student networks, extending up to their corresponding hint and guided layers. These functions are defined by their parameters W_{Hint} and W_{Guided}. Additionally, r signifies the regressor function located above the guided layer and characterized by the parameters W_r. It's important to note that the outputs of both u_h and r should be comparable, meaning that u_h and r should employ the same type of non-linearity.

The training procedure entails training the FitNet in a step-by-step manner, adhering to the student-teacher approach. This commences by utilizing a trained teacher network alongside a FitNet that is initialized with random weights. Subsequently, a regressor parameterized by W_r is introduced on top of the guided layer of the FitNet. The process involves training the FitNet parameters W_{Guided} up to the guided layer, aiming to minimize the objective stated in equation 3.15.

Following this, leveraging the pre-trained parameters obtained so far, the subsequent phase involves training the complete FitNet's parameters denoted as W_S. The aim here is to minimize the objective as outlined in equation 3.14.

Training FitNet has become commonly referred to as hint tecnique, in future papers that apply this technique.

3.2.3.3 Learning Efficient Object Detection Models with Knowledge Distillation

The section derives its title from a paper of identical name published in 2017 [8]. Within this study, the authors introduced an innovative framework for acquiring streamlined and efficient object detection networks, enhancing accuracy through the application of knowledge distillation. Prior to this, knowledge distillation had primarily exhibited noteworthy enhancements, but only within more straightforward classification scenar-

ios.

Applying distillation techniques to multi-class object detection, as opposed to image classification, poses distinct challenges due to several factors:

1. When we compress detection models using distillation techniques, we observe a more substantial degradation in performance compared to similar compression in image classification tasks. The reason behind this lies in the scarcity and higher cost associated with acquiring detection labels. Unlike image classification, where labels are usually abundant and less resource-intensive to obtain, object detection labels require meticulous annotation of object locations and extents within images. As a result, distilling knowledge while maintaining detection model accuracy becomes a challenging balancing act.

2. The foundational assumption in traditional knowledge distillation is that all classes in classification tasks are equally important. However, this assumption doesn't hold true in the context of object detection. In detection scenarios, the background class—representing areas without any object present—typically dominates in terms of occurrence. This skewed class distribution makes it more challenging to transfer knowledge because distillation might not fully capture the details of the less common but important object classes.

3. Object detection is a complex task, encompassing both classification and precise bounding box regression. In contrast, image classification focuses solely on assigning an image to a specific class. Combining these elements adds complexity to the knowledge distillation process, as both the accuracy of class predictions and the precision of object localizations need to be preserved.

4. The specific challenge addressed in the proposed framework is the transfer of knowledge within the same domain. In the proposed method, they work exclusively with images from the same dataset, without introducing external data or labels. This means that the images used to train the teacher model and the ones used to teach the student model come from the same set or source of data. This choice is deliberate and specific to this approach to preserve the integrity of the domain and ensure that the knowledge transfer remains within the bounds of the original dataset.

In the proposed framework, they adopt the Faster-RCNN as the foundational object detection architecture. As discussed in the 3.1.1.2, this architecture consists of three main modules:

1. Shared feature extraction performed by convolutional layers.
2. A region proposal network (RPN) responsible for generating potential object proposals.
3. A classification and regression network (RCN).

Both the RCN and RPN modules utilize the output of the shared feature extraction, and the RCN module also takes the results of the RPN as an input. To achieve highly accurate object detection outcomes, it's crucial to develop robust models for all three of these components. The complete learning framework is depicted in Figure 3.16. Initially, the structure embraces the concept of hint-based learning (described in 3.2.3.2), which encourages the student network's feature representation to closely align with that of the teacher network. Subsequently, the classification modules within both the RPN and RCN is enhanced using the knowledge distillation technique. To address the challenge of severe category imbalance in object detection, authors employed a weighted cross-entropy loss for the distillation process. Lastly, the teacher's regression output as an upper bound is incorporated, meaning that if the student's regression output surpasses that of the teacher, no additional loss is imposed.

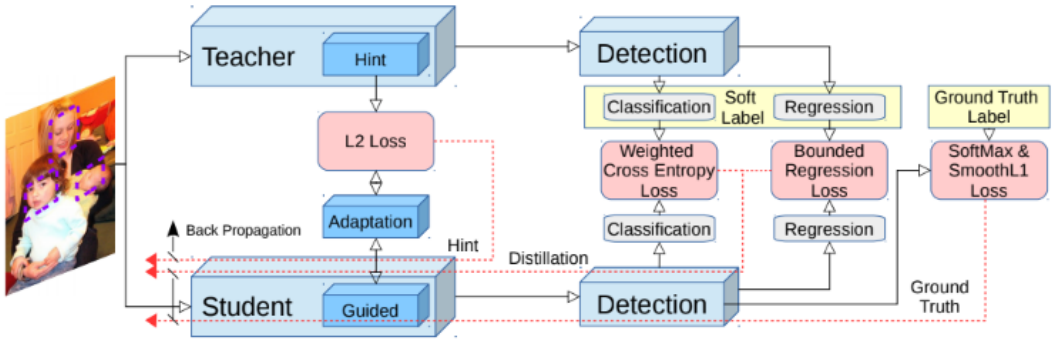


Figure 3.16: The proposed technique for the visual object detection task utilizes the Faster-RCNN architecture. It consists of the region proposal network and the region classification network. These two networks are simultaneously trained through a multi-task loss mechanism, enabling them to learn both the classifier and bounding-box regressor. To facilitate this, the ultimate output of the teacher's RPN and RCN as the targets for distillation are utilized, while utilizing intermediate layer outputs as hints. The direction of red arrows illustrates the pathways of backpropagation [8].

The overall objective to be optimized is as follow:

$$L_{RCN} = \frac{1}{N}\sum_{i} L_{cls}^{RCN} + \lambda\frac{1}{N}\sum_{j} L_{reg}^{RCN} \tag{3.16}$$

$$L_{RCN} = \frac{1}{M}\sum_{i} L_{cls}^{RPN} + \lambda\frac{1}{M}\sum_{j} L_{reg}^{RPN} \tag{3.17}$$

$$L_{RCN} + L_{RPN} + \gamma L_{Hint} \tag{3.18}$$

N and M represent the batch sizes for the RCN and RPN networks, respectively. Within this context, L_{cls} signifies the classifier loss function, encompassing a fusion of the hard softmax loss that utilizes ground truth labels and the soft knowledge distillation loss. Furthermore, L_{reg} denotes the bounding box regression loss, which combines a smoothed $L1$ loss with the teacher's constrained $L2$ regression loss. Lastly, L_{hint} stands for the hint-based loss function, designed to incentivize the student network to emulate the feature response of the teacher network. In the equation 3.18, hyper-parameters λ and γ, serving to balance the various loss components. In the experiments and benchmarks, we can set these hyper-parameters to 1 and 0.5, respectively. Each component of the objective function will be elaborated upon in the subsequent sections.

Knowledge Distillation for Classification In the vanilla distillation method we talked about in 3.2.3.1, both the hard and soft losses are basically different forms of cross entropy losses (equation 3.14). However, unlike simpler classification scenarios, the challenge in object detection lies in effectively handling a significant category imbalance. Specifically, the background class tends to dominate. In typical image classification, errors mainly stem from misclassifying foreground categories. In contrast, object detection faces a more complex situation where failing to distinguish between background and foreground instances can dominate the error metric, while misclassification among foreground categories is relatively infrequent. To deal with this situation, cross entropy loss is used, that takes into account the uneven class distribution. This helps the learning process focus on the more important parts of object detection:

$$H_{soft}(P_T, P_S) = -\sum w_c P_T log P_S \tag{3.19}$$

w_c, aims to address the problem of imbalanced classes. It involves assigning a higher weight to the background class and a comparatively smaller weight to the other classes. For instance, you might set w_0 to 1.5 for the background class and w_i to 1 for all the other classes.

As previously discussed, when the model's prediction P_T is very close to the actual label y_{True}, with one class having a probability close to 1 and most others close to 0, the temperature parameter T comes into play. This parameter softens the output probabilities. If you use a higher temperature, it makes the predictions more spread out, preventing the classes with near-zero probabilities from being ignored by the training

process. This is especially important for simpler tasks like classification. However, in more challenging problems where prediction errors are already high, using a larger T value introduces more noise, which can hinder learning.

Interestingly, in this approach, for even more difficult tasks like object detection, it's suggested that not using a temperature parameter at all (which is equivalent to setting T = 1) in the distillation loss actually works best in practice. This means that a simple and direct approach seems to be effective in these cases.

Knowledge Distillation for Regression In the context of object detection, achieving accurate regression modeling plays a crucial role in ensuring the overall detection accuracy. However, when it comes to regression tasks, such as predicting bounding box coordinates, the guidance provided by the teacher's regression outputs can be problematic. This is because these regression outputs are continuous and unbounded, making it challenging for the student model to directly replicate them. Unlike in cases where we're dealing with discrete categories (classification), regression outputs are not confined to specific values. This means that the teacher's regression predictions might deviate significantly from the actual ground truth values. Furthermore, the teacher's regression directions might even contradict the correct ground truth directions.

To address this issue, a different approach is taken to consideration. Instead of using the teacher's regression outputs as direct targets for the student, we consider them as an upper limit that the student should aim to achieve. The main idea is that the student's regression predictions should ideally align closely with the true ground truth labels. However, if the student's regression quality surpasses that of the teacher's by a certain margin, we don't enforce any additional loss on the student. This concept is termed the teacher bounded regression loss, denoted as L_b. In practical terms, we incorporate this teacher bounded regression loss into the overall regression loss, L_{reg}. This way, we encourage the student model to not only improve its regression accuracy but also strive to surpass the teacher's performance when possible. This approach helps in guiding the student model towards achieving better regression predictions while preventing it from blindly following potentially incorrect teacher guidance. L_b and L_{reg} is formulated as follow:

$$f(n) = \begin{cases} \|R_S - y\|_2^2, & \text{if } \|R_S - y\|_2^2 + m > \|R_T - y\|_2^2 \\ 0, & \text{otherwise} \end{cases}$$

$$L_{reg} = L_{sL1}(R_S, y_{reg}) + \nu L_b(R_S, R_T, y_{reg}) \tag{3.21}$$

In the provided equation, the variable m represents a designated margin, while y_{reg} signifies the true regression label. R_S refers to the regression output of the student network, and R_T corresponds to the predictions of the teacher network. The parameter ν is a weight factor, which we have set at 0.5 in original paper experimental settings. Within this context, $LsL1$ represents the smooth $L1$ loss as introduced [2]. The teacher bounded regression loss denoted as L_b serves to penalize the network solely when the student's error exceeds that of the teacher. It's worth noting that although we utilize an $L2$ loss within L_b, alternative regression loss functions such as $L1$ and smoothed $L1$ can be incorporated into L_b. The composite loss function we employ encourages the student network to approximate or even surpass the teacher network in terms of regression performance. However, it doesn't excessively push the student network beyond the level of the teacher's performance, once that benchmark has been achieved. This strategy maintains a balanced approach to training the student network.

Hint Learning In their work Chen et al. utilized Hint learning technique, as discussed in 3.2.3.2, during the process of applying Hint learning, a critical requirement is that the dimensions of neurons, encompassing channels, width, and height, should be consistent between corresponding layers within both the teacher and student networks. To achieve this alignment in the number of channels within hint and guided layers, an adaptation mechanism is introduced immediately following the guided layer. This additional adaptation layer is engineered to output data in the same size as the hint layer. Let's consider an example involving a teacher network and a student network in the context of image recognition. The teacher network has a convolutional layer with dimensions $32 \times 32 \times 64$ ($width \times height \times number of channels$), and this layer provides valuable feature representations. The student network, which is meant to learn from the teacher, also has a layer intended to capture similar features. In this case, the teacher's layer has 64 channels. To ensure alignment, an adaptation mechanism is introduced right after the student's corresponding layer. This adaptation layer is designed to output data in the same format as the hint layer, which means it should also have 64 channels.

Interestingly, the presence of an adaptation layer is shown to be crucial in achieving effective knowledge transfer, even when the channel numbers in both hint and guided layers are identical. This adaptation layer is capable of bridging differences in feature norms between hint and guided layers. In cases where hint or guided layers are convolutional and the resolutions of these layers differ (as observed in networks like VGG-16

and AlexNet), a padding technique can be is employed. This technique helps harmonize the number of output features across layers, thereby facilitating the knowledge transfer process.

Experiments The researchers in the study employed a variety of datasets, including KITTI [112], PASCAL VOC 2007 [70], and MS COCO [66] [66], coupled with diverse models, notably employing AlexNet [84] and AlexNet with Tucker decomposition [113]. The ensuing table presents the outcomes of their teacher-student model analysis, specifically applied to both the AlexNet and Tucker architectur.

Model	High-res Teacher		Low-res Baseline		Low-res Distilled Student	
	mAP	Speed	mAP	Speed	mAP	Speed
AlexNet	57.2	1,205/74ms	53.2	726/47ms	56.7(+3.5)	726/47ms
Tucker	54.7	663/41ms	48.6	430/29ms	53.5(+4.9)	430/29ms

Table 3.6: Comparison of high-resolution teacher model in [8].

3.2.3.4 Fine-grained Feature Imitation

Just like knowledge distillation, hint learning 3.2.3.2 enhances the quality of student models by reducing the gap between the comprehensive high-level features of teacher and student models. However, employing hint learning directly on detection models has a detrimental impact on performance. The rationale behind this is that detectors prioritize local areas that align with actual objects in the image, whereas classification models focus more on broader context. Consequently, directly applying complete feature imitation would inevitably introduce considerable noise from irrelevant regions, particularly in the case of object detection. This problem gets even worse when there are many different background instances, which can make it harder for the learning process.

As discussed, One valuable piece of information which can be utilized for distillation is the relative probabilities assigned to different classes by the teacher model. Meanwhile, detectors focus on identifying local areas in an image where objects are found. These areas are often linked to anchor points or boxes that enclose possible object instances. When you move around the anchor point or box close to an object, the detector's feature responses (like neuron activation in a neural network) change. This variation is important because it demonstrates how the detector's features shift their attention across different parts of the object and its surroundings. This is referred to as the difference in feature response on nearby anchor points near the object. In their papers [9], Wang et al., use this inter-location discrepancy for distilling knowledge in object detector, cre-

ated a mechanism that makes use of actual object bounding boxes and anchor priors to accurately determine the meaningful nearby anchor positions around objects, and subsequently, guide the student model in mimicking the teacher's behavior in those positions. They call their technique Fine grained feature imitation.

The approach aims to tackle the noted difficulties by going beyond the conventional knowledge distillation of a classification model, which solely relies on the softened output of a teacher model. Instead, it centers on an inter-location discrepancy present in the teacher's high-level feature response. By incorporating fine-grained feature imitation layer prior to the classification and localization layers (as illustrated in Figure 3.18), enhancements are observed in both of these sub-tasks. Experimental results further demonstrate that this technique significantly improves the student model's capacity for both class discrimination and localization. Furthermore, this technique avoids the incorporation of noisy and less informative background regions.

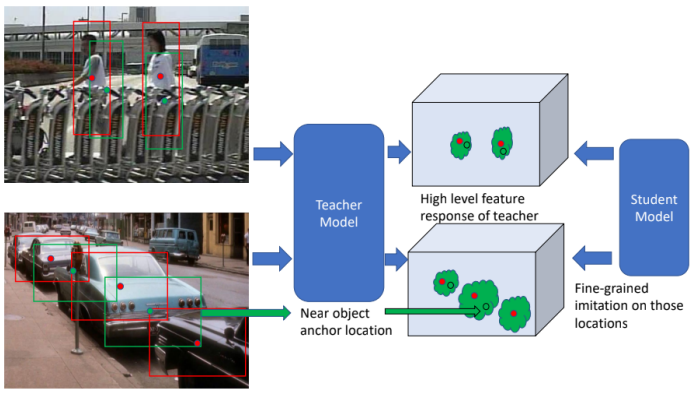


Figure 3.17: In the left two images, there are red and green bounding boxes, which serve as anchor boxes chosen for specific positions. The red anchors are closest match to ground truth bounding boxes, while the green anchors represent examples of near object samples. This selection is driven by the concept that variations in feature responses among anchor locations near objects offer insights into how a trained teacher model generalizes its knowledge. As a result, this method first finds these spots with a lot of useful information and then lets the student model imitate how the teacher responds to them [9].

The method can be divided to 2 basic steps, Imitation region estimation and Fine-grained feature imitation.

Imitation region estimation In order to clearly establish and examine the concept of local feature regions, we employ both the accurate ground truth bounding boxes and the predetermined anchor points. These elements allow us to determine these regions using a mask called I, and this is done separately for each image. Additionally, the technique has a way to control the size of these regions using a factor known as the thresholding factor ψ. Let's delve into the process further:

As illustrated in Figure 3.17, for every ground truth box, we compute IOU between that

box and all the anchor points. This computation results in an IOU map, denoted as m which has dimensions $W \times H \times K$. W and H are width and height of the feature map and K is number of anchor boxes.

Now, we're interested in finding the highest value within this IOU map, which we denote as M. This maximum IOU value is then multiplied by the thresholding factor ψ, leading us to a filter threshold called F. This threshold, F, is used to filter the IOU map. Specifically, we keep only those locations where the IOU values are larger than the filter threshold F. These selected locations are combined using a logical OR operation, resulting in a new mask with dimensions $W \times H$.

By applying this process iteratively for each ground truth box, we obtain separate masks for each one. These masks help us define the local feature regions for the objects in the image, taking into account their size and position. This thorough exploration of the local feature regions enables us to gain insights into how features are distributed around objects and how they contribute to the understanding of the objects' characteristics and relationships.

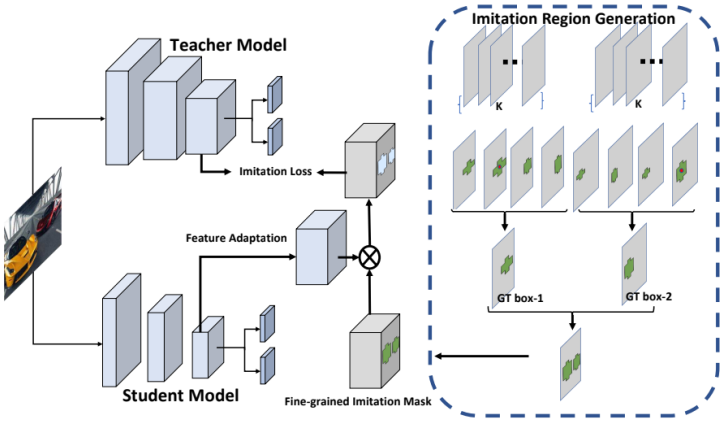


Figure 3.18: Demonstration of the suggested technique for imitating detailed features. The student detector's training involves using both actual object supervision and mimicking the teacher's feature response at nearby object anchor points. The feature-adaptation layer ensures that the student's guided feature layer aligns with the teacher's. To pinpoint valuable areas, we carry out a step-by-step process: calculating the Intersection over Union (IOU) map for each real object bounding box in relation to anchor priors, then filtering and merging potential locations. This results in the creation of the ultimate imitation mask [9].

Fine-grained feature imitation To facilitate the process of imitation, a convolution adaptation layer is inserted following the student model, just before computing the distance measure between the student's and teacher's feature responses, as depicted in Figure 3.18. The introduction of this adaptation layer is motivated by two considerations:

1. The channel count in the student's feature may not be in sync with that of the teacher's model. By introducing this additional layer, we can harmonize the stu-

dent's feature channels with the teacher's, thereby enabling the calculation of the distance measure.

2. Interestingly, even when we make the student imitate teacher's features, performance improvement is small and is not comparable with using adaption layer.

To explain how feature imitation works, we first define s as the guided feature map from the student model and t as the teacher's feature map. For every nearby object anchor spot (i, j) on the feature map with dimensions W (width) and H (height), the student model is trained to minimize the objective function:

$$l = \sum_{c=1}^{C} (f_{adapt}(s)_{ijc} - t_{ijc})^2 \tag{3.22}$$

To distill knowledge from Teacher model with the help of near anchor positions we use imitation mask I to minimize the loss:

$$L_{imitation} = \frac{1}{2N_p} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{c=1}^{C} I_{ij} (f_{adapt}(s)_{ijc} - t_{ijc})^2, \tag{3.23}$$

$$\text{Where} \quad N_p = \frac{1}{2N_p} \sum_{i=1}^{W} \sum_{j=1}^{H} I_{ij}$$

I represents the imitation mask, N_p stands for the count of positive points within the mask, and $fadap()$ represents the adaptation function. Consequently, the complete training loss for a student model can be expressed as:

$$L = L_{gt} - \lambda L_{imitation} \tag{3.24}$$

Where L_{gt} denotes the training loss for detection, and λ serves as the balancing factor for the imitation loss weight.

Visualization of imitation mask To gain a clearer understanding of the imitation regions generated by this approach, we visualizes some example masks I on input images, utilizing a detector devised by the authors. Figure 3.19 illustrates these example imitation masks, which have been scaled and imposed onto the input images.

Among the set of six images, Figure 3.19(a) represents the original image. Meanwhile, Figures 3.19(b), 3.19(c), and 3.19(d) depict images generated with different values of the parameter ψ, namely $\psi = 0.2$, $\psi = 0.5$, and $\psi = 0.8$ respectively. Additionally,

Figures 3.19(e) and 3.19(f) exhibit images filtered with a constant threshold value of F, where $F = 0.5$ and $F = 0.8$ respectively. It's evident from the visualizations that some objects are omitted when utilizing only $F = 0.5$, and nearly all imitation masks vanish when employing $F = 0.8$. This behavior is attributed to the constant filter threshold of F, which exhibits bias towards ground truth boxes of comparable size to prior anchors. The method introduced by the authors, featuring an adaptive filter threshold, effectively mitigates this issue.

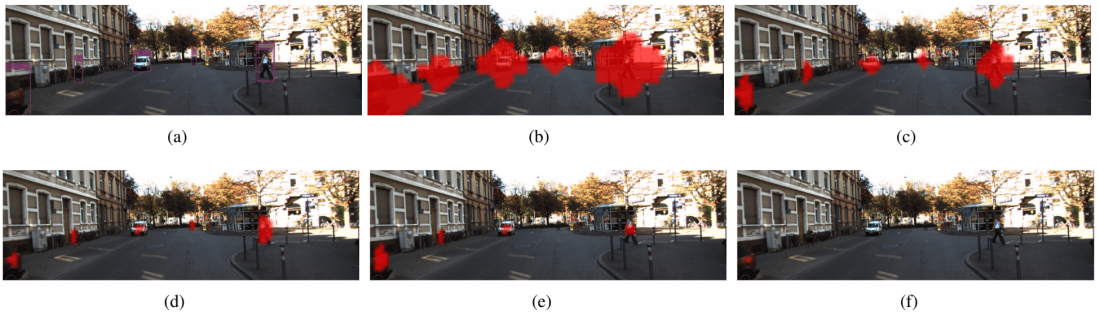


Figure 3.19: Imitation mask examples, resized and placed over the input images [9].

3.2.3.5 Improved Knowledge distillation with attention guided distillation (FKD)

In their paper [10] Linfeng Zhang and Kaisheng Ma present an innovative solution to address the challenges posed by the imbalance problem and to enhance the Fine-grained Feature Imitation technique. Their proposed method, known as Fast Knowledge Distillation (FKD), involves attention-guided distillation, which is designed to overcome the challenge of imbalance by selectively distilling essential foreground pixels which previous researchers tried to tackle. Fine-grained feature imitation the focus is on distilling features only from anchor locations near objects. Figure 3.20 shows the difference between the fine-grained mask-based detection and FKD distillation method.

Figure 3.20: Comparison between attention-guided and attention-guided methods [10].

In contrast to prior works, this method identifies foreground object pixels using an attention mechanism, which can be easily derived from features. This approach enables a more flexible and versatile way of distilling information, free from the constraints of specific annotations or bounding box definitions.

In this approach, authors leverage the attention map, which provides insights into the location of crucial pixels within an image, as shown by Zhou et al. in [114]. This attention map serves as a mask during the process of knowledge distillation. Specifically, pixels with higher attention values are identified as belonging to foreground objects and are given higher priority in the student model's learning process. Notably, this attention map-derived mask offers a more detailed representation compared to previous binary mask methods like [9], while also eliminating the need for additional supervision. Furthermore, this method distinguishes itself from prior attention-based distillation approaches like [53]. In this method, the attention map not only serves as the information to be distilled but also functions as the mask signal for feature distillation. This dual usage of the attention map enhances the efficacy of the approach.

As we've discussed thus far, there's a widely accepted understanding that valuable information in object detection lies within the relationships between different objects. Researchers have effectively enhanced detector performance by enabling detectors to perceive and utilize these relationships. Notable examples include non-local modules [9] and relation networks [115]. However, discussed methods for knowledge distillation in object detection focus solely on individual pixel information, disregarding the relationships among different pixels. To deal wth this issue, FKD introduces the concept of non-local distillation. But this approach aims to capture the relationship information between teachers and students using non-local modules and then transfer this information from teachers to students.

One advantageous aspect of the proposed approach is that the non-local modules and attention mechanism are only employed during the training phase. This ensures that no additional computation or parameters are introduced during the inference phase. Additionally, this method falls under the category of feature-based distillation, which makes it versatile across various detectors without requiring any specific modifications.

Furthermore, authors study, explores the relationship between teachers and students within the context of object detection. Notably, they claim to find that knowledge distillation for object detection demands a high average precision (AP) teacher—an observation that differs from conclusions drawn in image classification scenarios where a high AP teacher might adversely affect student performance [48].

To summarize the findings of this study:

1. It introduces the attention-guided distillation, a technique that directs the student's learning focus towards foreground objects, while reducing emphasis on background pixels.

2. It presents the non-local distillation approach, enabling students to gain knowledge not just from individual pixels, but also from the connections between various pixels.

3. The study indicates that a teacher with a higher average precision (AP) tends to be a more effective instructor for knowledge distillation in object detection. This contrasts with conclusions drawn from image classification scenarios.

Attention guided distillation We use the symbol A to represent the feature obtained from the backbone of an object detection model. The dimensions of feature A are denoted as C, H, W, where C represents the number of channels and H, W are height and weight of the feature. Creating the spatial attention map and channel attention map means finding special functions that transform feature A. The goal of these maps is to emphasize the importance of certain elements within the feature for further processing. To achieve this, two mapping functions are defined:

1. ζ^s: This function maps the original feature A from C, H, W to a new map in H, W dimensions. This new map highlights the spatial attention by considering the sum of absolute values across the channel dimension. ζ^s can be formulated as $\zeta^s = \frac{1}{C}\sum_k^C |A_{k,.,.}|$ To explore the idea lets consider an example feature:

$$A_{C:1} = \begin{pmatrix} 1.2 & 0.8 & 1.5 & 2.0 \\ 0.5 & 1.0 & 1.2 & 0.7 \\ 2.5 & 1.8 & 0.6 & 1.3 \\ 0.9 & 1.4 & 2.2 & 1.0 \end{pmatrix},$$

$$A_{C:2} = \begin{pmatrix} 1.3 & 0.7 & 1.6 & 2.1 \\ 0.6 & 1.1 & 1.3 & 0.6 \\ 2.4 & 1.7 & 0.5 & 1.4 \\ 0.8 & 1.3 & 2.3 & 1.1 \end{pmatrix},$$

$$A_{C:3} = \begin{pmatrix} 1.1 & 0.9 & 1.4 & 1.9 \\ 0.4 & 0.9 & 1.1 & 0.8 \\ 2.3 & 1.6 & 0.7 & 1.2 \\ 0.7 & 1.2 & 2.1 & 0.9 \end{pmatrix}$$

$$\zeta^s(A) = \begin{pmatrix} 1.2 & 0.8 & 1.5 & 2.0 \\ 0.5 & 1.0 & 1.2 & 0.7 \\ 2.4 & 1.7 & 0.6 & 1.3 \\ 0.8 & 1.3 & 2.2 & 1.0 \end{pmatrix}$$

2. ζ^c: This function maps the original feature A from C, H, W to a new map in C dimensions. This new map emphasizes channel attention by summing the absolute values across the width and height dimensions.ζ^c can be formulated as $\zeta^c = \frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}|A_{.,i,j}|$. Using previous example calculation for $\zeta^c(A)$ is as follow:

$$\zeta^c(A)_{c=1} = \begin{pmatrix} |1.2| + |0.8| + |1.5| + |2.0| + \\ |0.5| + |1.0| + |1.2| + |0.7| + \\ |2.5| + |1.8| + |0.6| + |1.3| + \\ |0.9| + |1.4| + |2.2| + |1.0| \end{pmatrix} = 20.6$$

$$\zeta^c(A)_{c=2} = \begin{pmatrix} |1.3| + |0.7| + |1.6| + |2.1| + \\ |0.6| + |1.1| + |1.3| + |0.6| + \\ |2.4| + |1.7| + |0.5| + |1.4| + \\ |0.8| + |1.3| + |2.3| + |1.1| \end{pmatrix} = 20.8$$

$$\zeta^c(A)_{c=3} = \begin{pmatrix} |1.1| + |0.9| + |1.4| + |1.9| + \\ |0.4| + |0.9| + |1.1| + |0.8| + \\ |2.3| + |1.6| + |0.7| + |1.2| + \\ |0.7| + |1.2| + |2.1| + |0.9| \end{pmatrix} = 19.2$$

$$\zeta^c(A) = \frac{1}{4\times 4}\begin{pmatrix} 20.6 & 20.8 & 19.2 \end{pmatrix}$$

$$\zeta^c(A) = \begin{pmatrix} 1.28 & 1.3 & 1.2 \end{pmatrix}$$

Subsequently, the spatial attention mask Ms and the channel attention mask Mc employed in attention-guided distillation are derived by aggregating the attention maps from both the teacher and the student detector.

$$M^s = HW.Softmax((\zeta^s(A^S) + \zeta^s(A^T))/T) \tag{3.25}$$

$$M^c = HW.Softmax((\zeta^c(A^S) + \zeta^c(A^T))/T) \tag{3.26}$$

T, introduced in section 3.2.3.1, is a hyper-parameter in the softmax function. Its purpose is to fine-tune the distribution of elements within attention masks. In Figure 3.21, the visualization and distribution of spatial attention are presented for various values of T. When T is smaller, knowledge distillation places greater emphasis on pixels with high attention values.

Figure 3.21: visualization and distribution of spatial attention for different values of T [10].

The attention-guided distillation loss L_{AGD} comprises two main elements: the attention transfer loss L_{AT} and the attention-masked loss L_{AM}. The purpose of L_{AT} is to prompt the student model to replicate the spatial and channel attention patterns exhibited by the teacher model:

$$L_{AT} = L_2((\zeta^s(A^S), \zeta^s(A^T)) + L_2((\zeta^c(A^S), \zeta^c(A^T)) \tag{3.27}$$

L_{AM} is employed to prompt the student to imitate the teacher model's features through an L_2 norm loss, adjusted by the masks M^s and M^c, as follows:

$$L_{AM} = (\sum_{k=1}^{C}\sum_{i=1}^{H}\sum_{j=1}^{W}(A^T_{k,i,j} - A^S_{k,i,j})^2.M^s_{i,j}.M^c_k)^{\frac{1}{2}} \tag{3.28}$$

None-local distillation The Non-local module, introduced in section 3.2.3.4. in 2018, proves to be a potent technique for enhancing neural network performance by capturing overarching contextual information. Within this study, the authors employ non-local modules to grasp the interconnectedness among pixels within an image. This relationship can be mathematically represented as shown in equation 3.29

$$r_{i,j} = 1/WH\sum_{i'}^{H}\sum_{j'}^{W} f(A_{.,i,j}, A_{.,i',j'})g(A_{.,i',j'}) \tag{3.29}$$

r is the relation information. i and j denote the spatial indexes of an output position for

which we intend to calculate the response. i' and j' act as the spatial indexes, encompassing all the potential positions. The f function helps us compare two pixels, while g gives us information about just one pixel. L_{NLD} or non-local distillation loss function serves as a mechanism to quantify the disparity between the relational information possessed by the students and that by the teachers. To compute this dissimilarity, we employ the L2 norm, a mathematical method: $L_{NLD} = L2(r^S, S^T)$

Overall loss function The overall distillation loss can be defined using the formulation presented in equation 3.30. This equation introduces three hyper-parameters that are used to balance various distillation losses. The comprehensive distillation loss remains model-agnostic, allowing it to be directly incorporated into the original training loss of any detection model.

$$L_{Distill}(A^T, A^S) = \underbrace{\alpha L_{AT} + \beta L_{AM}}_{\text{Attention-guided distillation}} + \underbrace{\gamma L_{NLD}}_{\text{Non-local distillation}} \tag{3.30}$$

Experiments In the experimental phase of their study, the authors conducted an extensive evaluation of their proposed methods across diverse datasets and models. The assessment encompassed a range of architectures including Faster RCNN [3], Cascade RCNN [94], Dynamic RCNN [116], Grid RCNN [117], and Retinanet [11], alongside the FSaf Retinanet [118] variant. However, for the sake of comparison with other methods introduced in this study, we focus solely on presenting the results related to Retinanet, as highlighted in Table 3.7 for reference.

Model	Backbone	AP	AP50	AP75	APS	APM	APL	FPS	Params
RetinaNet	ResNet50	37.4	56.7	39.6	20	40.7	49.7	17.7	37.74
Student Retinanet	ResNet50	39.6	58.8	42.1	22.7	43.3	52.5	17.7	37.74
RetinaNet	ResNet101	38.9	58	41.5	21	42.8	52.4	13.5	56.74
Student Retinanet	ResNet101	41.3	60.8	44.3	22.7	46	55.2	13.5	56.74

Table 3.7: Experimental Results Using FKD Distillation Technique as Presented in [10].

Noticeable and consistent increases in Average Precision (AP) are evident across both models.

3.2.3.6 Focal and Global Knowledge Distillation for Detectors (FGD)

FKD technique employs attention masks and the non-local module to guide the student and distill relationships, respectively. However, it distills both the foreground and background jointly. An essential challenge in detection distillation is to pinpoint the valuable

area for the distillation process. Prior methods for distillation treat all pixels and channels uniformly or distill all regions simultaneously. The authors of [13] argue that most existing methods lack the distillation of global context information. They overcome this by employing ground-truth boxes to isolate images, followed by the utilization of the teacher's attention masks to select crucial parts for distillation. Furthermore, they capture the global relationships between diverse pixels and distill them to the student model, introducing another enhancement (replacing non-local distillation in FKD and fine-grained methods).

To prove the logic behind their claims, authors designed experiments wherein the separation of foreground and background was implemented during the distillation process. Remarkably, as evident from Table 1, distilling the foreground and background together yielded the poorest performance, even falling short of using either the foreground or background alone. This observation suggests that disparities in the feature map's distribution can have an adverse impact on distillation outcomes.

	distillation area			mAP
	foreground	background	split	
RetinaNet Resnet101-Resnet50	×	×	-	37.4
	✓	×	-	39.3
	×	✓	-	39.2
	✓	✓	×	38.9
	✓	✓	✓	39.4

Table 3.8: Comparisons of different distillation areas [13].

The attention levels within each channel also vary noticeably. Upon closer examination, it becomes clear that these variations are not only due to interactions between foreground and background but also among individual pixels and channels. In light of this, the concept of focal distillation is introduced in the paper. Apart from the segregation of foreground and background, focal distillation computes the attention garnered by distinct pixels and channels in the teacher's feature set. This permits the student to concentrate on the teacher's critical pixels and channels. However focal distillation causes some missing global information. In order to compensate this global information, GcBlock [119] technique is used to extract the relation between different pixels and consequently then distill them from teachers to students. The method is called e Focal and Global Distillation (FGD) and Figure 3.22 is illustration of it.

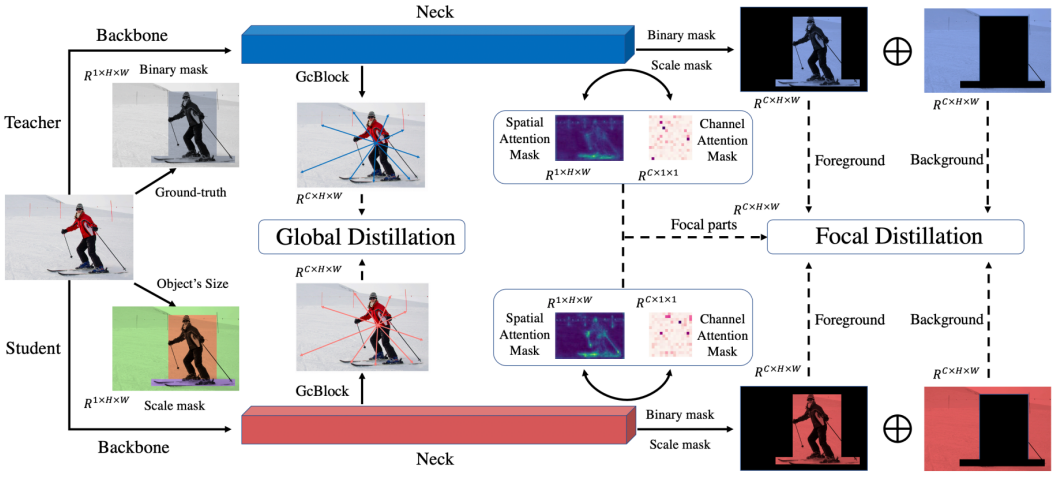


Figure 3.22: FGD process , encompassing both focal distillation and global distillation [11].

When looking at equations 3.22 and 3.30, both fine-grained Feature Imitation and FKD Distillation methods handle all parts of the problem equally and don't capture the global relationships between different pixels. FGD attempts to address these issues.

Focal Distillation In FGD approach, specific pixels and channels are selected, resulting in the acquisition of corresponding attention masks. Average values are subsequently computed for different pixels and channels, respectively. Throughout the training process, the student model is guided using the teacher's masks.

Global Distillation In Focal distillation, the connection between foreground and background information is effectively cut of. Consequently, Global Distillation is utilized to capture the overall relationship among various pixels within an image. This information is taken out of the feature maps and sent from the teacher model to the student model.

To achieve this, the GcBlock technique, introduced in [119] for capturing long-range dependencies within a visual scene, is leveraged. GcBlock allows for the extraction of a global understanding of the entire visual scene in a single image. The process of global distillation is illustrated in Figure 3.23.

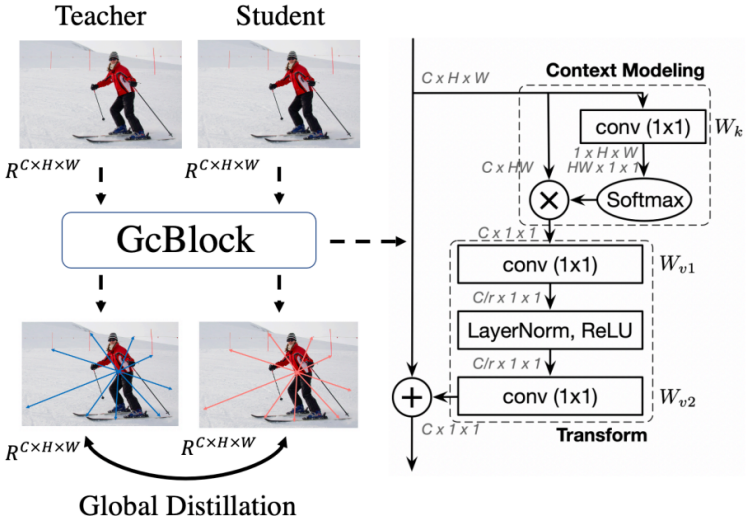


Figure 3.23: Global distillation using GcBlock [11].

Consequently global loss can be calculated:

$$L_{global} = \lambda . \sum (R(F^T) - R(F^S))^2,$$
$$R(f) = F + W_{v2}(ReLU(LN(W_{v1}(\sum_{j=1}^{N_p} \frac{e^{W_k F_j}}{\sum_{m=1}^{N_p} e^{W_k F_M}} F_j)))) \tag{3.31}$$

Finally the student is trained with totall loss:

$$L = L_{original} + L_{focal} + L_{global} \tag{3.32}$$

$L_{original}$ is original loss of detectors. The distillation loss is computed solely based on feature maps, which can be acquired from the neck of the detectors. This makes it straightforward to apply the distillation approach to various detectors.

Experiments In this section, we present the experimental results obtained from the original paper. The method was applied to the COCO dataset [66], which encompasses 80 distinct object classes. The training phase utilized 120,000 images from the training set, with testing conducted on a separate set of 5,000 validation images for all experiments. The performance assessment of various detectors is based on Average Precision. For the training process, all detectors were trained over 24 epochs using the SGD optimizer with a momentum value of 0.9 and a weight decay of 0.0001.

Table 3.9 showcases the results for Retinanet and RCNN models, as we will be draw-

ing upon this technique in Section 4.4.2 for our own experiments on Retinanet. The parameter setup included values of $\alpha = 1 \times 10^{-3}$, $\beta = 5 \times 10^{-4}$, $\gamma = 1 \times 10^{-3}$, and $\lambda = 5 \times 10^{-6}$ for all anchor-based one-stage models. Here, α and β serve as hyper-parameters to balance the loss between foreground and background, while γ is used to balance the attention loss and similarly λ to balance the global distillation loss.

Method	**mAP**
RetinaNet-Res101(T)	38.9
RetinaNet-Res50(S)	37.4
FGD	39
RCNN-Res101(T)	39.8
RCNN-Res50(S)	38.4
FGD	40.4

Table 3.9: Outcomes achieved using the FGD method on RetinaNet and RCNN detection frameworks on the COCO dataset. Here, T and S denote the teacher and student detectors, respectively.

Chapter 4

Methodology

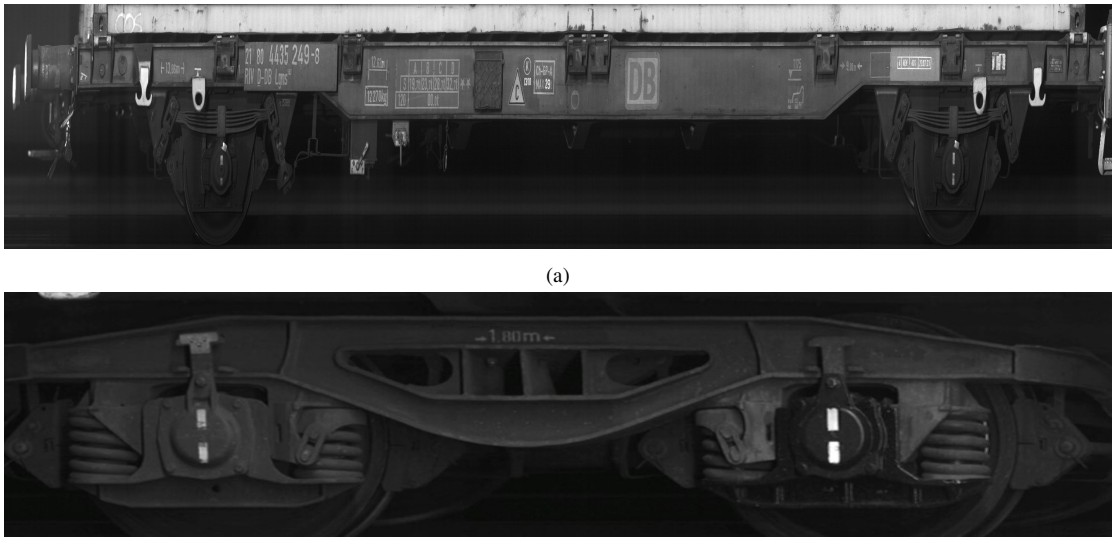
Using a quantitative methodology, this research delves into the domain of model compression techniques and their impact on one-stage object detectors. The inquiry centers on the well-established Retinanet architecture, with the aim of uncovering how two compression methods—Anchor Pruning and Focal and Global Knowledge Distillation for Detectors —can enhance its efficiency. These investigations are facilitated through carefully designed benchmarking experiments, all supported by the adaptable MMDetection library, purpose-built for object detection tasks. At the heart of this research lie benchmarking experiments, meticulously designed to evaluate the efficacy of distinct compression techniques within the context of the Retinanet model. Each experiment is thoughtfully constructed to assess how the model's performance is affected by the application of specific compression methods.

We chose the MMDetection library as our implementation platform because of its exceptional capability to seamlessly execute a wide range of object detection tasks. By harnessing the capabilities of the MMDetection library, both the benchmarking experiments and the groundbreaking Knowledge Distillation framework can be smoothly integrated into the Retinanet architecture, ensuring the research's successful execution.

In this chapter, we will thoroughly discuss the experimental setup. This discussion will include a detailed overview of the datasets used for training and evaluation, an exploration of the performance metrics chosen to measure model effectiveness, and a comprehensive review of the tools employed to conduct these experiments.

4.1 Datasets

Among the array of potential applications, the experiments focus on a specific use case involving object detection in freight wagon images. Illustrated in Figure 4.1 is a side view image of a wagon. The sample image highlights certain challenges, such as image darkness, or hand-written text, both of which pose considerable demands on the detection process.



(a)

(b)

Figure 4.1: Sample images of train wagon from datasets.

The ground truth labels encompass categorization into 18 distinct classes. A subset of these labels is displayed in Figure 4.2

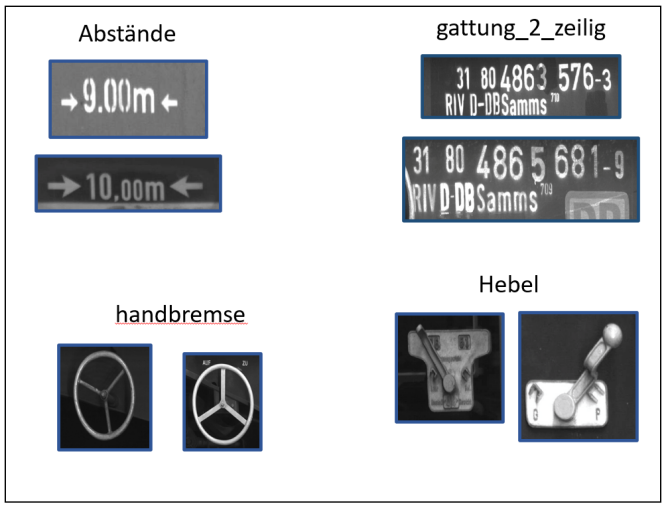


Figure 4.2: Samples from ground truth classes.

The selection of appropriate dataset sizes plays a important role in ensuring the preci-

sion of the developed models. For the model training process, a labeled image dataset of 2022 images is used. This dataset was divided into an 80-20 train-validation split. Type of utilized datasets for experiment is MS COCO [66] dataset.

The MS COCO dataset, or Common Objects in Context, is a prominent reference in the world of object detection and segmentation tasks. Annotations within the dataset are meticulously organized in a json format to provide comprehensive information about object instances in images. These annotations consist of critical attributes such as object category labels, bounding box coordinates, and segmentation masks. Each annotation entry uniquely characterizes an object instance and is structured with the following components:

- Image Information: This contains image information such as the image ID, file name, dimensions (width and height), and other related details.
- Object Annotations: Object instances are characterized by their bounding box coordinates (including the x-coordinate of the top-left corner, y-coordinate of the top-left corner, width, and height), the associated category label, and a unique annotation ID.
- Categories: The dataset includes a broad spectrum of object categories, each assigned a unique category ID and a human-readable label.

For evaluating the models, a separate dataset containing 1072 images was employed. Similar to the training dataset, these images were drawn from the dataset, ensuring consistency in data source and annotation quality.

4.2 Metrics

In the context of model compression, evaluating the performance, accuracy, complexity is crucial. This evaluation helps find the right balance among these factors. To achieve this balance, specific evaluation metrics are chosen. These metrics serve as guides to understand how well model compression techniques work for object detection. This section elaborates on these metrics, explaining their importance and how they contribute to evaluating compressed models.

4.2.1 Mean Average Precision (mAP)

Mean Average Precision (mAP) is a widely used metric to assess the accuracy of object detection models. As discussed in detail in 3.1.3, It provides a comprehensive understanding of a model's performance by considering various levels of precision-recall trade-offs. The concept of mAP revolves around calculating the average precision for each class and then taking their mean.

4.2.2 FLOPs (Floating-Point Operations)

FLOPs is a metric that quantifies the computational complexity of a neural network model. It estimates the number of operations, mainly multiplications and additions, required to process an input sample through the network. It serves as a proxy for the computational demand of a model during both training and inference.

- Model Head FLOPs: In the experiments, the FLOPs of the model's head were measured. The model head contains the layers responsible for object detection, such as the detection heads and related layers. Measuring FLOPs in the model head gives insights into the complexity of the detection process.
- Total FLOPs: Total FLOPs account for all computations within the entire neural network, including both the backbone (feature extraction) and the model head. It provides a holistic view of the computational demand of the entire model.

4.2.3 Number of Parameters

The number of parameters in a neural network indicates its model size and complexity. Generally, more parameters mean a more complex model. However, with techniques like model compression, it's possible to reduce the number of parameters while maintaining performance to some extent. It's anticipated that model compression techniques might lead to a reduction in the number of parameters.

4.3 Tools

Throughout the experiments conducted, a diverse set of tools and programming packages were employed to facilitate the research process. This section focuses on three key tools that held substantial significance within the study.

4.3.1 MMDetection

MMDetection is a robust and versatile open-source software framework designed to facilitate state-of-the-art object detection research and development. MMDetection, short for **Open MMLab Detection Toolbox and Benchmark**, is a project developed by the Open MMLab, a research group at Multimedia Laboratory, The Chinese University of Hong Kong [88]. The project aims to provide a comprehensive toolbox for object detection tasks and evaluation metrics. It is built on top of the PyTorch deep learning framework, leveraging its flexibility and scalability for developing and experimenting with various object detection models. MMDetection is built with a modular and extensible architecture, enabling researchers and practitioners to explore a wide range of object detection methods. It offers a collection of pre-implemented models, backbones, and head architectures, allowing users to assemble and experiment with different configurations. At the core of MMDetection are the following components:

- **Backbones:** These are the foundational feature extractors that process input images and generate feature maps with relevant information. Users can choose from a variety of backbone architectures, including ResNet, VGG, and EfficientNet.
- **Neck Modules:** These modules enhance the features extracted by the backbones by incorporating multi-scale context information and improving the representation power of the model. FPN (Feature Pyramid Network) and PANet (Path Aggregation Network) are popular examples.
- **Detection Heads:** Detection heads generate bounding box predictions and class scores based on the features extracted by the backbone and neck modules. Various heads, such as Faster RCNN, Retinanet, and YOLO, can be utilized based on the specific detection task requirements.
- **Loss Functions:** MMDetection offers a range of loss functions, including standard ones like cross-entropy and smooth L1 loss, as well as more advanced losses tailored to object detection tasks.

MMDetection serves as a valuable asset for this research, offering a user-friendly approach to access its capabilities. By cloning their GitHub repository, it's comprehensive functionalities can be accessed. This toolkit is designed with a modular structure that lends itself to customization, allowing you to shape the model architecture to align with your specific use cases. This involves making adjustments to components such as backbones, neck modules, and detection heads. Through this modular framework,

the model's design can be tailored to suit the research objectives and application requirements. In addition to model configuration, MMDetection's flexibility extends to the training process. Parameters and settings can be adjusted to fine-tune the model's performance on your dataset or within specific contexts. This adaptability ensures that your research outcomes are both relevant and optimized for the intended application. Furthermore, MMDetection empowers the users to customize loss functions. This capability proves advantageous in aligning the training process with the nuances of the research goals or dataset characteristics. By manipulating train detectors, you can effectively steer the learning process towards convergence and desired results.

This study employs MMDetection for developing pruning and quantization algorithms, as well as applying knowledge distillation techniques and theoretical principles. The toolkit is also used to assess the performance of models with the aim of comparison between compressed and baseline model performance.

4.3.2 PyTorch

PyTorch, an open-source machine learning framework developed by Facebook's AI Research lab [120], plays a vital role in this study. PyTorch is highly regarded among researchers and developers for its user-friendly interface and computational graph execution model. In contrast to static graph frameworks, PyTorch empowers developers to dynamically create and adjust their neural network models, streamlining swift experimentation and debugging. Its automatic differentiation capabilities simplify the process of computing gradients, crucial for training models using gradient-based optimization algorithms.

MMDetection, built upon the PyTorch framework, offers a diverse selection of predesigned components, yet it may not encompass every conceivable structure and implementation. For model modifications and tailored feature extraction, the integration of complementary tools like PyTorch becomes exceedingly valuable. This study emphasizes the importance of synergizing MMDetection's capabilities with PyTorch to create models, implement techniques and adjust network architectures in alignment with distinct research goals.

4.3.3 MLFlow

MLFlow, an open-source platform designed by Databricks Inc., is aimed at streamlining and enhancing the machine learning lifecycle [121]. The utility of MLFlow becomes

evident through its four primary components:

- **Tracking:** MLFlow allows users to systematically track experiments, parameters, metrics, and model versions. This feature is particularly valuable when managing various iterations of model training and optimization. By capturing essential information, MLFlow ensures the reproducibility of experiments and provides insights into the performance of different models.
- **Projects:** The Projects component enables the encapsulation of code, data, and environment specifications into a reusable package. This promotes consistency and eliminates discrepancies between development and deployment environments.
- **Models:** MLFlow facilitates the seamless packaging and versioning of trained models. This simplifies the sharing and deployment of models across different stages of development and across various team members.
- **Deployment:** MLFlow also offers model deployment capabilities, aiding in the transition from model development to deployment. This ensures that the models that have been tested and tracked using MLFlow can be efficiently deployed and monitored in production environments.

In the context of this study, MLFlow is utilized to effectively track the training process of models and evaluate optimization efforts. The tracking capability allows for the documentation of different iterations of model training, the parameters used, and the associated performance metrics. This not only fosters accountability but also enables comprehensive comparisons between different approaches.

4.4 Experimental Setup

4.4.1 Anchor Pruning

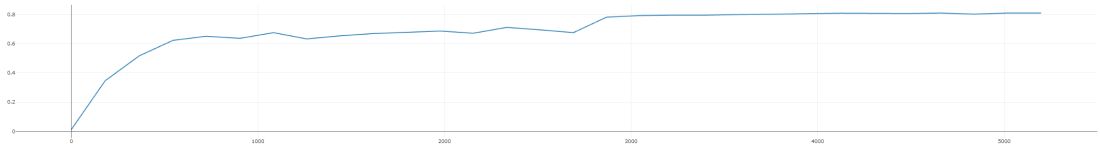
The core of the anchor pruning technique is the implementation of the greedy search algorithm, as elaborated upon in section 3.2.1.2. This algorithm initiates its operation using a trained model. Two foundational models, SSD300 and Retinanet, were chosen to serve as baseline models. The training process involved utilizing a dataset comprising 2022 images, subjecting both models to 30 epochs of training. The specifics of the base model configurations are outlined in Table 4.1.

Baseline Model	train img	val img	Image Scale	num Epochs	score_thr	Optimizer	Backbone
SSD 300	1617	405	3000x500	30	0.05	Adam	Resnet-50
Retinanet	1617	405	2400x400	30	0.05	Adam	VGG16

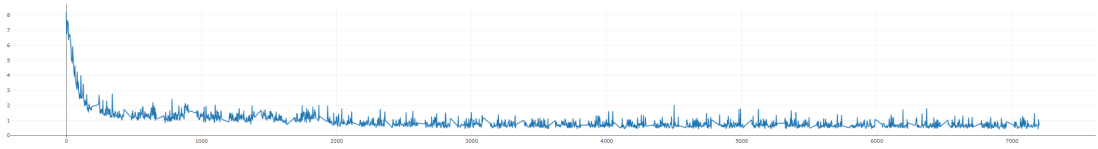
Table 4.1: Baseline Model configuration and parameters.

Figure 4.3 provides a visual representation of the learning curves for both baseline models, offering a deeper insight into their training progress. It is essential to monitor model convergence within the first 30 epochs, particularly when assessing the accuracy of pruned models, and even more so when there is an improvement in accuracy post-pruning.

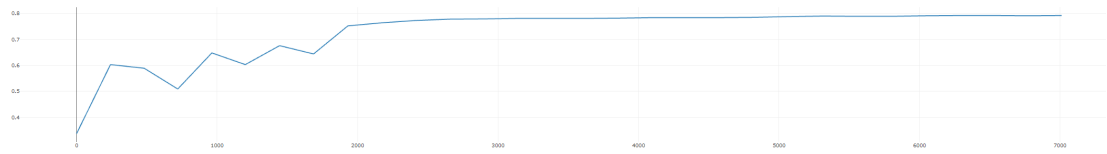
(a) SSD300 training loss curve.



(b) SSD300 validation mAP curve.



(c) Retinanet training loss curve.



(d) Retinanet validation mAP curve.

Figure 4.3: Learning curves for both baseline models.

In anchor pruning experiment we utilized these two pre-trained models checkpoints as an input for anchor search algorithm. These models trained with their original anchor setup. Originally, we have 30 anchors for SSD300 and 45 for Retinanet. Figure 4.4 visually presents how SSD300 and Retinanet begin with their original anchor setups. In each iteration of the search algorithm the algorithm thoroughly examines these anchor sets, considering important mAP and model head Flops.

The algorithm's objective is to achieve a balanced relationship between mAP and Flops. In each iteration, it focuses on enhancing anchor selections, retaining only the Pareto-

optimal ones. This precise anchor search process ensures that the selected anchors strike a balance between accuracy and computational efficiency, ultimately improving the model's overall performance.

```
strides=[8, 16, 32, 64, 100, 300],
scale_ratios=(
    ((0.07, 1.0), (0.07, 2.0), (0.07, 0.5),
    (0.1024, 1.0)),
    ((0.15, 1.0), (0.15, 2.0), (0.15, 3.0),
    (0.15, 0.5), (0.15, 1 / 3), (0.224, 1.0)),
    ((0.33, 1.0), (0.33, 2.0), (0.33, 3.0),
    (0.33, 0.5),(0.33, 1 / 3), (0.4102, 1.0)),
    ((0.51, 1.0), (0.51, 2.0), (0.51, 3.0),
    (0.51, 0.5),(0.51, 1 / 3), (0.5932, 1.0)),
    ((0.69, 1.0), (0.69, 2.0), (0.69, 0.5),
    (0.7747, 1.0)),
    ((0.87, 1.0), (0.87, 2.0), (0.87, 0.5),
     (0.9557, 1.0)),
    ),
anchor_base_size=300,
centers=((4, 4), (8, 8), (16, 16),
         (32, 32), (50, 50), (150, 150))
```

(a)

```
strides=[8, 16, 32, 64, 128],
scale_ratios=(
    ((0.0625, 0.5), (0.0787, 0.5), (0.0992, 0.5),
     (0.0625, 1.0), (0.0787, 1.0), (0.0992, 1.0),
     (0.0625, 2.0), (0.0787, 2.0), (0.0992, 2.0)),
    ((0.125, 0.5), (0.1575, 0.5), (0.1984, 0.5),
     (0.125, 1.0), (0.1575, 1.0), (0.1984, 1.0),
     (0.125, 2.0), (0.1575, 2.0), (0.1984, 2.0)),
    ((0.25, 0.5), (0.315, 0.5), (0.3969, 0.5),
     (0.25, 1.0), (0.315, 1.0), (0.3969, 1.0),
     (0.25, 2.0), (0.315, 2.0), (0.3969, 2.0)),
    ((0.5, 0.5), (0.63, 0.5), (0.7937, 0.5),
     (0.5, 1.0), (0.63, 1.0), (0.7937, 1.0),
     (0.5, 2.0), (0.63, 2.0), (0.7937, 2.0)),
    ((1.0, 0.5), (1.2599, 0.5), (1.5874, 0.5),
     (1.0, 1.0), (1.2599, 1.0), (1.5874, 1.0),
     (1.0, 2.0), (1.2599, 2.0), (1.5874, 2.0))
    ),
anchor_base_size=512
```

(b)

Figure 4.4: (a) SSD300 baseline configuration with 30 anchors (b) Retinanet baseline configuration with 45 anchors.

4.4.2 Focal and Global Knowledge Distillation

In section 3.1.1.3, we have discussed how Retinanet utilizes ResNet [69] as its foundational architecture. Employing more complex ResNets, such as ResNet-101 and Resnet-152, can enhance accuracy, though this comes at the cost of larger model sizes. Using smaller backbones like ResNet-50 and ResNet-18 can provide a more lightweight alternative with faster inference times and lower computational demands. The claim is supported by Figure 4, which offers a comparison of parameter counts between various types of ResNet networks, as obtained from PyTorch's model summary.

```
==========================================================================
Total params: 60,192,808
Trainable params: 60,192,808
Non-trainable params: 0
--------------------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 49.54
Params size (MB): 229.62
Estimated Total Size (MB): 279.20
--------------------------------------------------------------------------
```

(a) Model Summary of ResNet-152.

```
==========================================================================
Total params: 44,549,160
Trainable params: 44,549,160
Non-trainable params: 0
--------------------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 35.10
Params size (MB): 169.94
Estimated Total Size (MB): 205.09
--------------------------------------------------------------------------
```

(b) Model Summary of ResNet-101.

```
==========================================================================
Total params: 25,557,032
Trainable params: 25,557,032
Non-trainable params: 0
--------------------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 23.41
Params size (MB): 97.49
Estimated Total Size (MB): 120.95
--------------------------------------------------------------------------
```

(c) Model Summary of ResNet-50.

```
==========================================================================
Total params: 11,689,512
Trainable params: 11,689,512
Non-trainable params: 0
--------------------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 5.14
Params size (MB): 44.59
Estimated Total Size (MB): 49.78
--------------------------------------------------------------------------
```

(d) Model Summary of ResNet-18.

Figure 4.5: Model summary of diffrent ResNet types using torchvision model summary.

Our experiment is designed not only to address the trade-off between accuracy and model size but also to explore how knowledge distillation can improve the performance of models with smaller backbones. The goal is to distill knowledge from more cumbersome backbones into smaller ones in such a way that the student model achieves higher accuracy than training Retinanet with a smaller backbone from scratch, which serves as our baseline. This experiment focuses on applying FGD knowledge distillation, as detailed in Section 3.2.3.6, to compensate for any reduction in mean Average Precision (mAP) that may occur during this compression process. Based on the chronological description of various distillation strategies in Section 3.2.3, it is evident that FGD is considered to be a more effective approach for addressing issues related to object detection. This is the primary reason for selecting FGD as the method for experimentation in this study.

Knowledge distillation can be examined in terms of how it's incorporated into training methods. It can be categorized into three primary distillation schemes offline-distillation, online-distillation, and self-distillation [7], as illustrated in Figure 4.6.

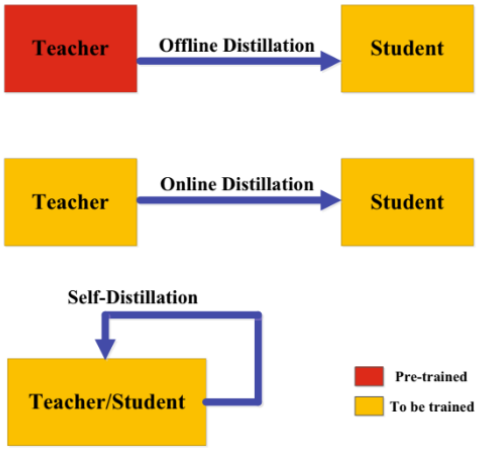


Figure 4.6: Different distillation schemes. Red for pre-trained means the networks are learned before distillation, yellow for to be trained, it means the networks are learned during distillation [7].

In our experiment, we employ both offline-distillation and self-distillation techniques on Retinanet models.

4.4.2.1 Offline-distillation scheme for model compression

The experiment was structured with two distinct distillation setups as follows:

1. **Retinanet Res101-Res50 Distillation:** In this setup, the Retinanet Res101 model served as the teacher model, initially trained for 30 epochs. The knowledge from

the teacher model is distilled into a Rtinanet Res50 student model. Retinanet Res50 model, trained utilizing the same training dataset was employed as a baseline for comparison.

2. **Retinanet Res50-Res18 Distillation:** In this configuration, the Retinanet Res50 model was designated as the teacher model, originally trained for 30 epochs. The knowledge from the teacher model is distilled into a Rtinanet Res18 student model. Similar to the previous setup, the trained Retinanet Res18 model was employed as the baseline model.

Details regarding the parameters of both the teacher, student and the baseline model can be found in Table 4.2.

Experiment	Model		Image Scale	num epoch	score_thr	Optimizer	Backbone
Retinanet Res101-Res50	Teacher	retinanet_r101_fpn_1x_coco	3500x500	30	0.5	Adam	ResNet 101
	Student	retinanet_r50_fpn_1x_coco	-	-	-	-	ResNet50
	Baseline	retinanet_r50_fpn_1x_coco	3500x500	30	0.5	Adam	ResNet50
Retinanet Res50-Res18	Teacher	retinanet_r50_fpn_1x_coco	3500x500	30	0.5	Adam	ResNet50
	Student	retinanet_r18_fpn_1x_coco	-	-	-	-	ResNet18
	Baseline	retinanet_r18_fpn_1x_coco	3500x500	30	0.5	Adam	ResNet18

Table 4.2: Teacher, Student and baseline model configuration and parameters. Student model is trained during distillation.

In all experiments, FGD distillation process was executed utilizing the Adam optimizer with a learning rate of 0.0001. Also we start the student with the same head structure as the teacher.

4.4.2.2 Self-distillation scheme for accuracy improvement

For the next experiment, We setup another scheme to see how the FGD distillation helps improving accuracy of the baseline model. Here Retinanet Res50 model served as both the teacher and student model. The FGD distillation process was executed utilizing the Adam optimizer with a learning rate of 0.0001. Same as offline method, We use the method to start the student with the same head structure as the teacher.

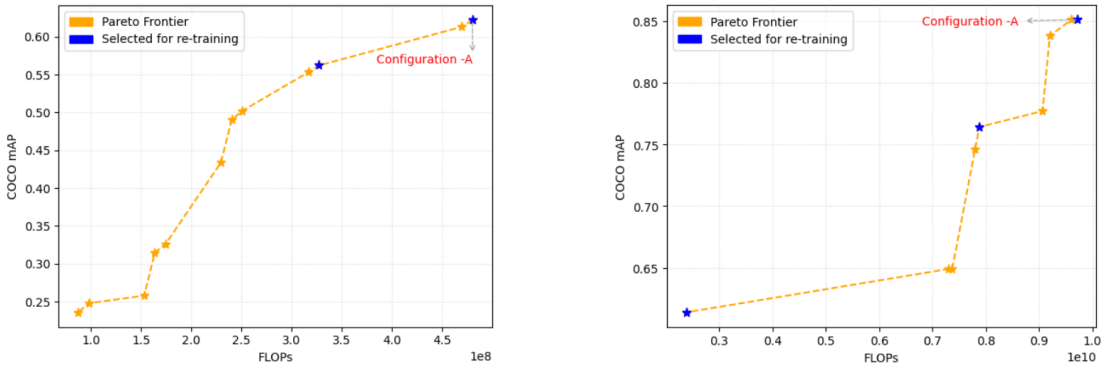
Chapter 5

Experimental Results

5.1 Anchor Pruning

The result of the anchor pruning search algorithm is represented as a Pareto chart. In the context of our baseline models, Figure 5.1 effectively displays these Pareto charts. The blue stars depicted on the Pareto chart symbolize selected anchor configuration for training. We name the configuration resulting in the highest mAP Configuration-A (It is worth mentioning that original paper calls the model with 15% drop in Flops still has same accuracy of the baseline). Correspondingly, the configuration with a lower mAP will be referred to as Configuration-B and C. When examining the SSD300 Pareto chart, Configuration -B appears promising with a 15% reduction in model complexity at the price of 9% drop in mAP.

In the Retina Pareto chart, Configuration-A, while there is no significant reduction in complexity, it shows potential for a mAP increase of 0.06. Anchor Configurations-B and C are expected to result in a 12% and 40% drop in Flops, respectively. However, it's crucial to note that Configuration -C will lead to a significant decrease in mAP ($18\% \downarrow$).



(a) SSD300: 2 configuration selected for re-training.

(b) Retinanet: 3 configuration selected for training.

Figure 5.1: Pareto charts are output of search algorithm. Each point is indicating a pareto-optimal anchor configuration which can be used for re-training of the model.

The Anchor Pruning Greedy Algorithm, detailed in Section 3.2.1.2, operates by iteratively eliminating combinations of anchors. The specific anchors to be removed for configurations in the Pareto chart are outlined below:

SSD300 Configuration-A: {1, 3, 5, 6, 7, 9, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29}

SSD300 Configuration-B: {0, 1, 3, 5, 6, 7, 9, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29}

Retinanet Configuration-A: {32, 33, 34, 35, 6, 7, 42, 43, 15, 16, 24, 25, 27, 28, 29, 30, 31}

Retinanet Configuration-B: {6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 24, 25, 27, 28, 29, 30, 31, 32, 33, 34, 35, 42, 43}

Retinanet Configuration-C: {0, 1, 2, 3, 4, 5, 6, 7, 8, 15, 16, 24, 25, 27, 28, 29, 30, 31, 32, 33, 34, 35, 42, 43}

To better understand the anchor setup and which aspect ratios are kept, check Figure 5.2. It clearly marks the aspect ratios that are removed, helping to grasp the pruning process easily.

```
strides=[8, 16, 32, 64, 100, 300],
scale_ratios=(
    ((0.07, 1.0), (0.07, 2.0), (0.07, 0.5),
    (0.1024, 1.0)),
    ((0.15, 1.0), (0.15, 2.0), (0.15, 3.0),
    (0.15, 0.5), (0.15, 1 / 3), (0.224, 1.0)),
    ((0.33, 1.0), (0.33, 2.0), (0.33, 3.0),
    (0.33, 0.5),(0.33, 1 / 3), (0.4102, 1.0)),
    ((0.51, 1.0), (0.51, 2.0), (0.51, 3.0),
    (0.51, 0.5),(0.51, 1 / 3), (0.5932, 1.0)),
    ((0.69, 1.0), (0.69, 2.0), (0.69, 0.5),
    (0.7747, 1.0)),
    ((0.87, 1.0), (0.87, 2.0), (0.87, 0.5),
     (0.9557, 1.0)),
    ),
anchor_base_size=300,
centers=((4, 4), (8, 8), (16, 16),
         (32, 32), (50, 50), (150, 150))
```

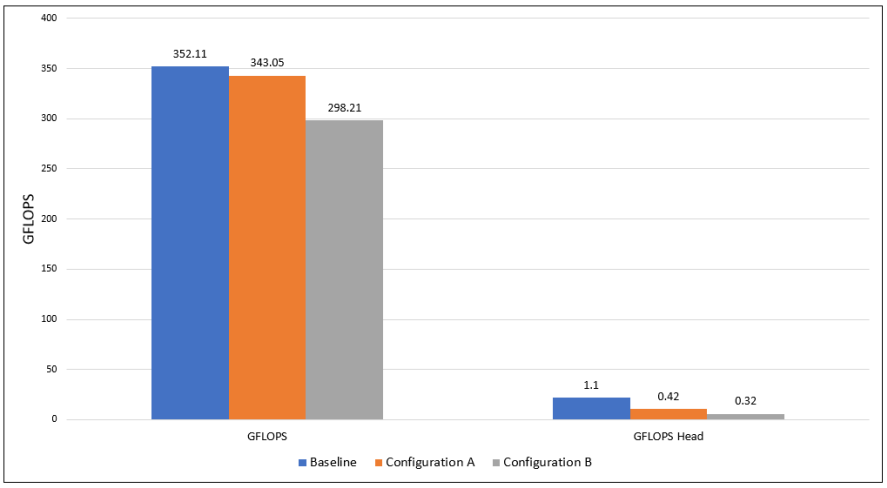
(a)

```
strides=[8, 16, 32, 64, 128],
scale_ratios=(
    ((0.0625, 0.5), (0.0787, 0.5), (0.0992, 0.5),
     (0.0625, 1.0), (0.0787, 1.0), (0.0992, 1.0),
     (0.0625, 2.0), (0.0787, 2.0), (0.0992, 2.0)),
    ((0.125, 0.5), (0.1575, 0.5), (0.1984, 0.5),
     (0.125, 1.0), (0.1575, 1.0), (0.1984, 1.0),
     (0.125, 2.0), (0.1575, 2.0), (0.1984, 2.0)),
    ((0.25, 0.5), (0.315, 0.5), (0.3969, 0.5),
     (0.25, 1.0), (0.315, 1.0), (0.3969, 1.0),
     (0.25, 2.0), (0.315, 2.0), (0.3969, 2.0)),
    ((0.5, 0.5), (0.63, 0.5), (0.7937, 0.5),
     (0.5, 1.0), (0.63, 1.0), (0.7937, 1.0),
     (0.5, 2.0), (0.63, 2.0), (0.7937, 2.0)),
    ((1.0, 0.5), (1.2599, 0.5), (1.5874, 0.5),
     (1.0, 1.0), (1.2599, 1.0), (1.5874, 1.0),
     (1.0, 2.0), (1.2599, 2.0), (1.5874, 2.0))
    ),
anchor_base_size=512
```

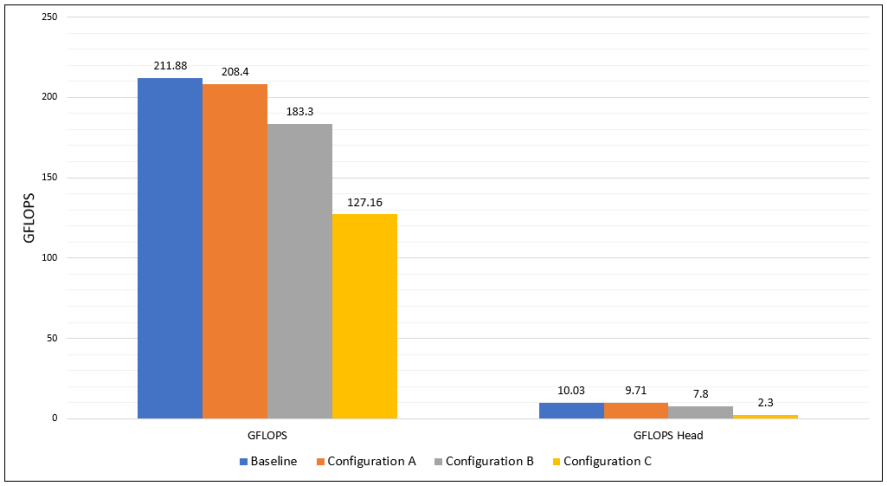
(b)

Figure 5.2: (a) SSD300 configuration-A with 26 anchors removed (b) Retinanet configuration-A with 17 anchors removed.

At this step, as the anchor pruning technique suggests, we need to re-train the models using new anchor configurations. After retraining the models with the chosen configurations and assessing their complexities, a noticeable reduction in model complexities has been observed, both in the overall model and the model's head. This trend is illustrated in Figure 5.3.



(a) SSD300: GFLOPS and GFLOPs in Model Head.



(b) Retinanet: GFLOPS and GFLOPs in Model Head.

Figure 5.3: Model Complexity comparison comparison between Retinanet baseline model and pruned models after re-training

After evaluation using the test dataset, the outcomes reveal notable reductions in the mean Average Precision (mAP) for the pruned models, with the exception of the Retinanet model trained by Configuration A. This observation is visually presented in Figure 5.4, validating the aforementioned findings. As demonstrated in the graph, there is a distinct trade-off between GFLOps and mAP.

Furthermore, Configuration A's distinctive performances might be connected to the shape and size of the training labels, which requires further investigation. Understanding the specifics of Configuration A could help improve pruned models in different

situations.

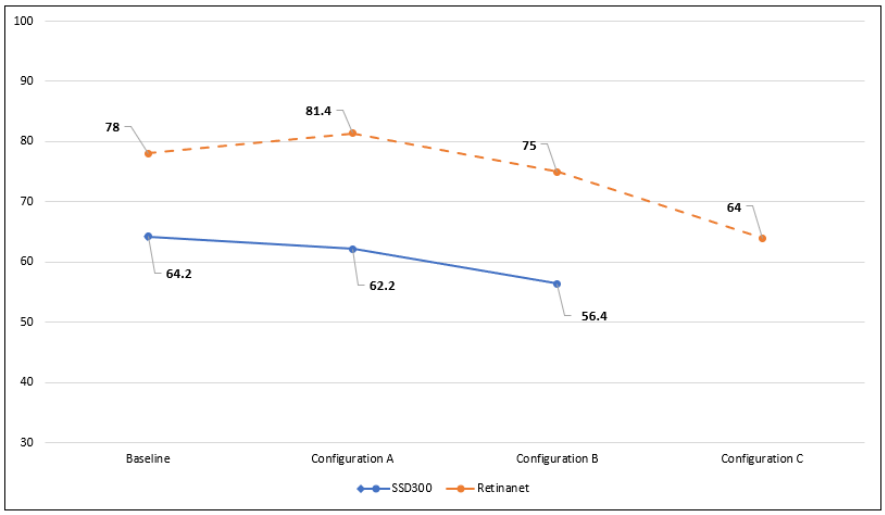


Figure 5.4: Comparative mAP Analysis of Pruned Models Comparison of mean Average Precision (mAP) values across pruned model configurations. The mAP values exhibit notable decreases for most pruned models, except for the Retinanet model trained using Configuration A, which showcases an improved performance.

In conclusion, we are presenting the pruning results in Table 5.1, which provide valuable insights into the influence of pruning on model accuracy and efficiency.

Model	**mAP**	**GFLOPs**	**GFLOPs Head**	**GFLOPs (%)**	**mAP(%)**
SSD300 Baseline	64.2	352.11	1.1	-	-
SSD300 configuration A	62.2	343.05	0.42	2.50% ↓	3% ↓
SSD300 configuration B	56.4	298.21	0.32	15% ↓	12% ↓
Retinanet Baseline	78.1	211.88	10.03	-	-
Retinanet configuration A	81.4	208.4	9.71	1.64% ↓	4% ↑
Retinanet configuration B	75	183.3	7.8	13% ↓	3.60% ↓
Retinanet configuration C	64	127.16	2.3	40% ↓	18% ↓

Table 5.1: Evaluation results for different anchor configurations of SSD300 and Retinanet.

5.2 Focal and Global Knowledge Distillation

In both offline-distillation setups, knowledge distilled from Teacher models to student models within 25 epochs and student models gain more accuracy than the baseline models. The outcomes of offline-distillation are visually represented in Figure 5.5.

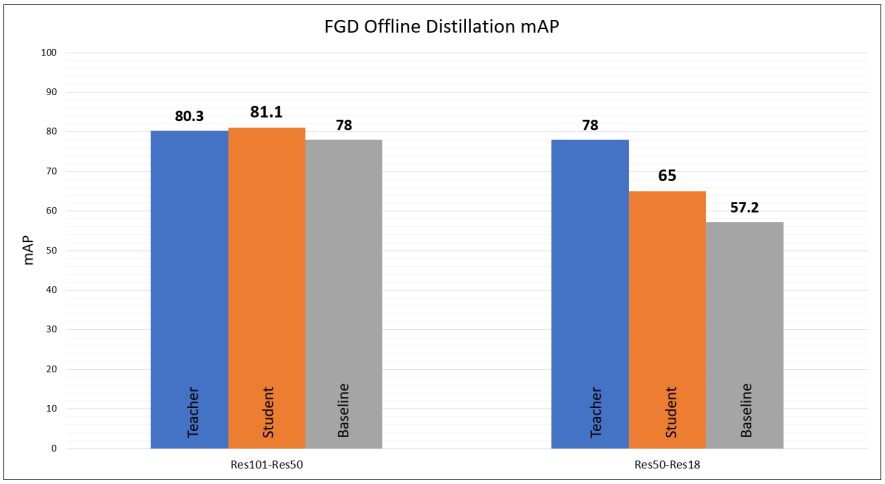
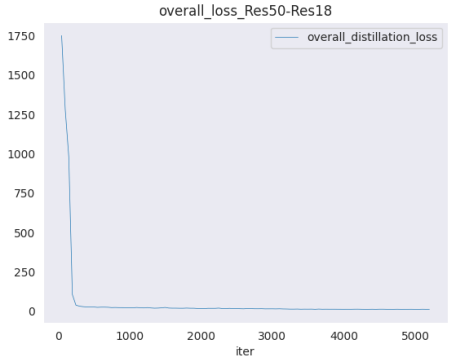


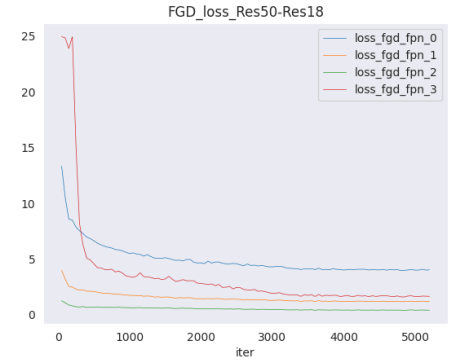
Figure 5.5: Distillation result. Student mAP gained more accuracy than baseline model during distillation.

To gain a more comprehensive understanding of the training process, we provide various graphs depicting different loss optimization and mAP measurements throughout the training phase. This visual representation allows for a clearer insight into the training progress and performance evaluation. Figure 5.6 is illustrating different training losses and Mean Average Precision on validation dataset logged during the distillation process of Retinanet Res50-Res18.

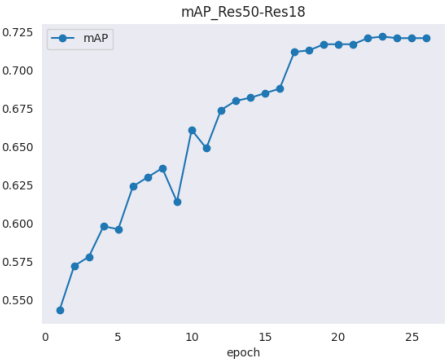


(a) Overall Loss.

(b) bbox and cls loss.



(c) distillation losses.



(d) mAP of validation data.

Figure 5.6: Visualiztion of distillation process for Retinanet Res50-Res18.

The self-distillation result shows 2% improvement in mAP after FGD distillation. Needless to mention that in this experiment Teacher model serves as a baseline for comparison:

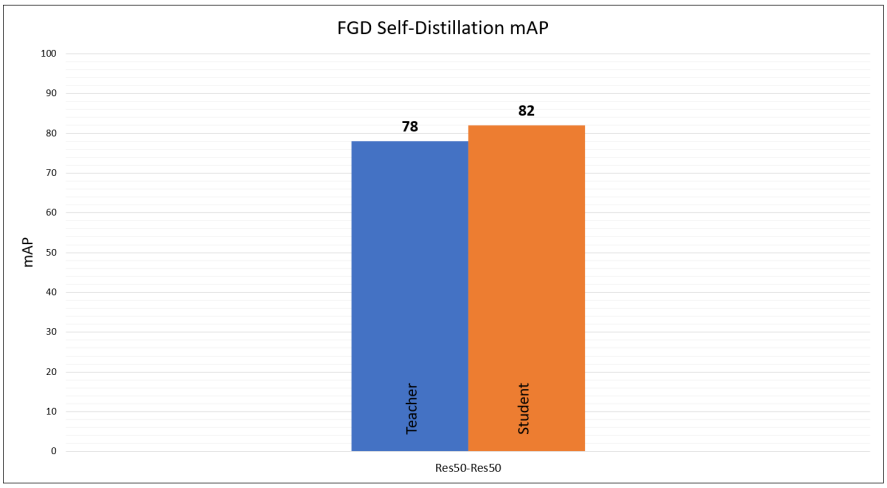
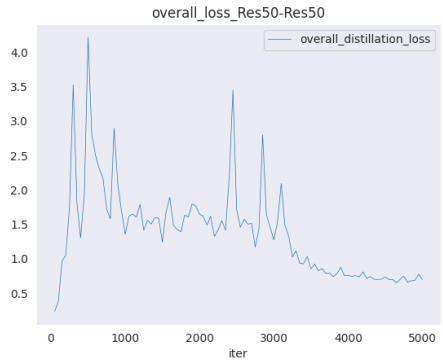


Figure 5.7: Self-distillation result for Retinanet Res50. mAP improved for 3.6% after distillation.

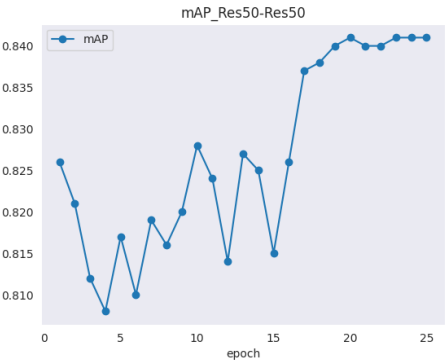
Same as offline-distillation experiment, in figure 5.8 we present training losses and Mean Average Precision on validation data logged during the distillation process for more insight.



(a) Overall Loss.

(b) bbox and cls loss.

(c) distillation losses.



(d) mAP of validation data.

Figure 5.8: Visualiztion of distillation process for self-distillation Retinanet Res50.

Summarizing our findings, Table 5.2 presents the experiment results, shedding light on the effectiveness of the FGD method in our study. These results offer crucial insights into the performance and utility of FGD in our research.

Experiment	Teacher mAP	Student mAP	Baseline mAP	mAP gain	Parameter Reduction (%)
Retinanet Res101-Res50	80.3	81.1	78	4%	40%
Retinane Res50-Res18	78	65	57.2	13.60%	54%
Retinanet Res50-Res50	78	82	78	5.10%	-

Table 5.2: Evaluation Results for Knowledge Distillation Experiments. Promising improvements in the mean average precision (mAP) of student models suggest the effectiveness of FGD method.

Chapter 6

Conclusion

The main objective of this study was to tackle the problem of model compression in object detection models. We explored innovative techniques tailored to address the unique challenges found within this field. The underlying hypothesis guiding our research was founded on the notion that these techniques not only effectively reduce model size but also offer methods to compensate for accuracy reduction, thereby providing a practical solution to the trade-off between model compactness and performance in object detection.

6.1 Contributions and Key Findings

6.1.1 Exploring Compression Techniques for Object Detection

In this reserch we conducted an extensive survey of three primary compression techniques: pruning, quantization, and knowledge distillation. Our initial observation indicates that traditional pruning, quantization, and knowledge distillation techniques are best suited for compressing simpler neural networks. These traditional methods can only be implemented individually on various components of CCN-Based object detection models, particularly on the backbone.

With the aim of finding best-suited techniques for object detection models, our survey extended beyond this initial insight. We did deeper exploration of techniques custom-tailored to the distinct structure of object detection models, encompassing the Backbone, Head, and Neck components. We investigated techniques specifically designed to address the challenge of accuracy loss during compression. We discovered that some of these methods not only improve the trade-off between model accuracy and complex-

ity but also, quite intriguingly, have the potential to enhance model accuracy independently of the compression process. We provided convincing evidence of this in our experiments, which included anchor pruning and FGD methods.

Additionally, This comprehensive survey enhances our understanding of model compression techniques and their applicability in the context of object detection. Moreover, it sets the stage for researchers to enhance and innovate these methods, ultimately advancing the field.

6.1.2 Tackling object detection specific challenges in Knowledge Distillation

For knowledge distillation, this study embarked on a systematic survey of methods. This journey unfolded chronologically, enabling us to gain valuable insights into the evolution and development of techniques designed to enhance the efficiency and performance of these models.

The exploration commenced by establishing a foundational understanding of the technicalities involved in knowledge distillation. We discussed the various types of knowledge distillation, each based on different sources of knowledge in different parts of the NN. This initial groundwork allowed us to build a solid foundation upon which we could investigate more advanced and specialized approaches.

One crucial milestone in our exploration was the in-depth examination of Hint distillation, a prominent feature-based knowledge distillation technique. Understanding Hint distillation was pivotal, as it formed the basis for our subsequent exploration of Learning Efficient Object Detection Models with Knowledge Distillation 3.2.3.3. This section, written based on paper with same name, not only presented a structured approach to distilling object detection models (considering components of detectors) but also introduced methods for designing loss functions tailored to different components of the distillation process. A notable contribution of this method lies in shedding light on a previously unaddressed issue: the imbalance problem between background and foreground images in object detection. the authors highlighted the importance of addressing this imbalance and proposed solutions that involve giving different weightage to foreground and background classes within the loss function, thereby compensating for this inherent issue.

We explored three different feature-based distillation techniques created to tackle these two challenges. Each method aimed to bridge gaps and enhance earlier techniques. As a result, our study on Focal and Global Optimization Distillation (FGD) conclusively

establishes it as the most advanced and state-of-the-art approach in this context.

Finally, in our in-depth exploration of various methods, we deciphered the underlying methodologies of these techniques. What's noteworthy is that all the methods we surveyed share a common approach: Identifying crucial regions within the feature map and adjusting the distillation loss to compensate for the overall loss in the distillation process. This shared approach that provides researchers with a clear path for refining existing techniques, devising novel methods, and making valuable contributions to the field.

6.1.3 Experimental Validation of Hypothesis

In our methodology section, guided by our hypothesis, we meticulously designed and conducted two experiments: one focused on Anchor Pruning and the other on FGD methods. The findings from these experiments strongly support our initial hypothesis, reinforcing the notion that specialized compression techniques for object detection models can effectively reduce model size, providing techniques to to compensate for accuracy reduction.

Anchor Pruning: In Table 5.1, we've compiled a thorough summary of anchor pruning findings. This table helps us assess how adjusting anchor configurations impacts the model's complexity and accuracy, making it valuable for analysis. In the case of SSD300, we achieved a reduction in model complexity by approximately 2.5% and 15%, with a corresponding trade-off of 3% and 12% decrease in mAP.

For Retinanet model, by utilizing anchor configurations B and C during retraining, we observed a reduction in model complexity of around 13% and 40%, while experiencing a trade-off of 3.6% and 18% drop in mAP for the models. It is noteworthy that Configuration A led to a 4% improvement in mean Average Precision (mAP), aligning with our hypothesis that the anchor pruning technique can enhance accuracy in specific use cases.

As shown in Figure 5.1, the anchor pruning technique gives us Pareto-front charts. Also, the anchor pruning search algorithm tells us the exact anchors linked to each configuration. One notable discovery from our experiment is that when we are using the removed anchors for each configuration, combined with the Pareto-front charts, it yields valuable insights. This helps us pinpoint the most important anchors that impact the model's accuracy in each specific use-case. With this strategy, anchor pruning can serve as a tool to spot most prominent anchors affecting the accuracy of model within

the given dataset.

Focal and Global Knowledge Distillation for Detectors (FGD): In our investigation of offline-distillation, we applied a compression technique to the Retinanet model with ResNet-101 backbone, resulting in a more compact model employing a ResNet-50 backbone. Figure 5.5 visually demonstrates the tangible benefits of this approach, showcasing a 4% increase in accuracy when compared to training Retinanet with a ResNet-50 backbone from scratch. This substantial enhancement in model performance is complemented by a significant reduction in parameters. Specifically, the distillation process managed to reduce the parameters of the Retinanet model by 40%, resulting an architecture that reduced the model's size from 169.94 MB to 97.49 MB within the backbone.

Furthermore, we successfully compressed Retinanet with a ResNet-50 backbone, achieving an impressive 54% reduction in parameters, reducing the backbone's size from 97.49 MB to 44.59 MB. Remarkably, this parameter reduction was accompanied by a substantial 14% increase in the mAP, further validating the effectiveness of FGD technique. These results provide robust evidence of the efficacy of FGD distillation, which leverages both focal and global loss optimization strategies to enhance model accuracy in parallel with distilling the knowledge to smaller model.

In addition to our prior experiments, we conducted a self-distillation experiment to further underscore the efficacy of FGD in enhancing accuracy. This experiment was specifically designed to demonstrate the potential of FGD as a viable strategy for improving accuracy in the context of object detection. As shown in Figure 5.7, the self-distillation experiment yielded notable results, showcasing a 5% improvement in the mean Average Precision. This highlights the versatility of FGD technique, as it can be integrated into various object detection pipelines to enhance model performance.

6.2 Future Directions

In the course of our research, we have identified two key areas that hold potential for future contributions to the field of deep learning. These discoveries open up exciting avenues for further exploration and development.

Firstly, in our anchor pruning experiment, we observed a unique performance trend in some configurations. These configurations demonstrated an improvement in mean Average Precision while simultaneously removing specific anchors. This intriguing

finding may be linked to the characteristics of the training labels, particularly their shape and size. To explore this further, additional investigation is necessary. Analyzing the aspect ratios of the removed anchors and comparing them with the shapes of the training labels could yield valuable insights, potentially leading to innovative approaches for optimizing object detection models.

Secondly, as we delved into knowledge distillation, we discovered a commonality among the discussed methods. This shared characteristic offers a chance to create a framework for object detection knowledge distillation techniques. This framework would offer a structured and standardized way for researchers to design and carry out their custom distillation experiments more easily and accurately. This framework holds the potential to simplify the process of developing and assessing new distillation methods, encouraging innovation in the field of deep learning.

References

[1] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, pp. 257–276, 2023.

[2] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, pp. 91–99, 2015.

[4] R. Benenson, S. Popov, and V. Ferrari, "Large-scale interactive object segmentation with human annotators," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 700–11 709.

[5] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[6] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "Zeroq: A novel zero shot quantization framework," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 169–13 178.

[7] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, pp. 1789–1819, 2021.

[8] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," *Advances in neural information processing systems*, vol. 30, pp. 742–751, 2017.

[9] T. Wang, L. Yuan, X. Zhang, and J. Feng, "Distilling object detectors with fine-grained feature imitation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4933–4942.

[10] L. Zhang and K. Ma, "Improve object detection with feature-based knowledge distillation: Towards accurate and efficient detectors," in *International Conference on Learning Representations*, 2020.

[11] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[12] M. Bonnaerens, M. Freiberger, and J. Dambre, "Anchor pruning for object detection," *Computer Vision and Image Understanding*, vol. 221, p. 103445, 2022.

[13] Z. Yang, Z. Li, X. Jiang, Y. Gong, Z. Yuan, D. Zhao, and C. Yuan, "Focal and global knowledge distillation for detectors," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4643–4652.

[14] L. Liu, S. Zhang, Z. Kuang, A. Zhou, J.-H. Xue, X. Wang, Y. Chen, W. Yang, Q. Liao, and W. Zhang, "Group fisher pruning for practical network compression," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7021–7032.

[15] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, pp. 137–154, 2004.

[16] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. Ieee, 2005, pp. 886–893.

[17] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *2008 IEEE conference on computer vision and pattern recognition*. Ieee, 2008, pp. 1–8.

[18] 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*, 2014, pp. 580–587.

[19] ——, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 1, pp. 142–158, 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[22] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[23] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 734–750.

[24] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[25] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*. Springer, 2020, pp. 213–229.

[26] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable detr: Deformable transformers for end-to-end object detection," *arXiv preprint arXiv:2010.04159*, 2020.

[27] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, pp. 598–605, 1989.

[28] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in neural information processing systems*, vol. 5, pp. 164–171, 1992.

[29] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, pp. 1135–1143, 2015.

[30] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," *Advances in neural information processing systems*, vol. 29, pp. 1379–1378, 2016.

[31] A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, and H. Li, "Learning n: m fine-grained structured sparse neural networks from scratch," *arXiv preprint arXiv:2102.04010*, 2021.

[32] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," *arXiv preprint arXiv:2104.08378*, 2021.

[33] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE transactions on information theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[34] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[35] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[36] C. E. Shannon *et al.*, "Coding theorems for a discrete source with a fidelity criterion," *IRE Nat. Conv. Rec*, vol. 4, no. 142-163, p. 1, 1959.

[37] C. Baskin, B. Chmiel, E. Zheltonozhskii, R. Banner, A. M. Bronstein, and A. Mendelson, "Cat: Compression-aware training for bandwidth reduction," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 12 356–12 375, 2021.

[38] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[39] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[40] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 2019, pp. 3009–3018.

[41] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," *Advances in Neural Information Processing Systems*, vol. 32, pp. 7950–7958, 2019.

[42] R. Zhao, Y. Hu, J. Dotzel, C. De Sa, and Z. Zhang, "Improving neural network quantization without retraining using outlier channel splitting," in *International conference on machine learning*. PMLR, 2019, pp. 7543–7552.

[43] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1325–1334.

[44] M. Haroush, I. Hubara, E. Hoffer, and D. Soudry, "The knowledge within: Methods for data-free model compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8494–8502.

[45] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[46] J. Kim, S. Park, and N. Kwak, "Paraphrasing complex network: Network compression via factor transfer," *Advances in neural information processing systems*, vol. 31, pp. 2760–2769, 2018.

[47] J. Ba and R. Caruana, "Do deep nets really need to be deep?" *Advances in neural information processing systems*, vol. 27, pp. 2654–2662, 2014.

[48] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh, "Improved knowledge distillation via teacher assistant," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 5191–5198.

[49] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[50] Z. Huang and N. Wang, "Like what you like: Knowledge distill via neuron selectivity transfer," *arXiv preprint arXiv:1707.01219*, 2017.

[51] S. Ahn, S. X. Hu, A. Damianou, N. D. Lawrence, and Z. Dai, "Variational information distillation for knowledge transfer," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 9163–9171.

[52] B. Heo, M. Lee, S. Yun, and J. Y. Choi, "Knowledge transfer via distillation of activation boundaries formed by hidden neurons," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3779–3787.

[53] N. Komodakis and S. Zagoruyko, "Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer," in *ICLR arXiv:1612.03928*, 2017.

[54] D. Chen, J.-P. Mei, Y. Zhang, C. Wang, Z. Wang, Y. Feng, and C. Chen, "Cross-layer distillation with semantic calibration," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 8, 2021, pp. 7028–7036.

[55] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proceedings of the*

IEEE conference on computer vision and pattern recognition, 2017, pp. 4133–4141.

[56] S. Lee and B. C. Song, "Graph-based knowledge distillation by multi-head attention network," *arXiv preprint arXiv:1907.02226*, 2019.

[57] Y. Liu, J. Cao, B. Li, C. Yuan, W. Hu, Y. Li, and Y. Duan, "Knowledge distillation via instance relationship graph," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7096–7104.

[58] F. Tung and G. Mori, "Similarity-preserving knowledge distillation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1365–1374.

[59] L. Yu, V. O. Yazici, X. Liu, J. v. d. Weijer, Y. Cheng, and A. Ramisa, "Learning metrics from teachers: Compact networks for image embedding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2907–2916.

[60] Q. Li, S. Jin, and J. Yan, "Mimicking very efficient network for object detection," in *Proceedings of the ieee conference on computer vision and pattern recognition*, 2017, pp. 6356–6364.

[61] R. Sun, F. Tang, X. Zhang, H. Xiong, and Q. Tian, "Distilling object detectors with task adaptive regularization," *arXiv preprint arXiv:2006.13108*, 2020.

[62] X. Dai, Z. Jiang, Z. Wu, Y. Bao, Z. Wang, S. Liu, and E. Zhou, "General instance distillation for object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 7842–7851.

[63] J. Guo, K. Han, Y. Wang, H. Wu, X. Chen, C. Xu, and C. Xu, "Distilling object detectors via decoupled features," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2154–2164.

[64] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[65] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, pp. 154–171, 2013.

[66] X. Chen, H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollár, and C. L. Zitnick, "Microsoft coco captions: Data collection and evaluation server," *arXiv preprint arXiv:1504.00325*, 2015.

[67] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer, 2014, pp. 818–833.

[68] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[69] 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*, 2016, pp. 770–778.

[70] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, pp. 303–338, 2010.

[71] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.

[72] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer, 2014, pp. 740–755.

[73] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov *et al.*, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *International Journal of Computer Vision*, vol. 128, no. 7, pp. 1956–1981, 2020.

[74] S. Shao, Z. Li, T. Zhang, C. Peng, G. Yu, X. Zhang, J. Li, and J. Sun, "Objects365: A large-scale, high-quality dataset for object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 8430–8439.

[75] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 304–311.

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

[77] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1389–1397.

[78] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2018.

[79] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[80] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," *arXiv preprint arXiv:1810.05270*, 2018.

[81] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv preprint arXiv:1902.09574*, 2019.

[82] T. Suzuki, H. Abe, T. Murata, S. Horiuchi, K. Ito, T. Wachi, S. Hirai, M. Yukishima, and T. Nishimura, "Spectral pruning: Compressing deep neural networks via spectral analysis and its generalization error," *arXiv preprint arXiv:1808.08558*, 2018.

[83] L. Theis, I. Korshunova, A. Tejani, and F. Huszár, "Faster gaze prediction with dense networks and fisher pruning," *arXiv preprint arXiv:1801.05787*, 2018.

[84] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84–90, 2017.

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

[86] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[87] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[88] K. Chen, J. Wang, J. Pang, Y. Cao, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Xu *et al.*, "Mmdetection: Open mmlab detection toolbox and benchmark. arxiv 2019," *arXiv preprint arXiv:1906.07155*, 2019.

[89] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[90] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, pp. 3123–3131, 2015.

[91] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[92] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *arXiv preprint arXiv:1510.03009*, 2015.

[93] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," *arXiv preprint arXiv:2008.05000*, 2020.

[94] Z. Cai and N. Vasconcelos, "Cascade r-cnn: High quality object detection and instance segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 5, pp. 1483–1498, 2019.

[95] J. Fang, A. Shafiee, H. Abdel-Aziz, D. Thorsley, G. Georgiadis, and J. H. Hassoun, "Post-training piecewise linear quantization for deep neural networks," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer, 2020, pp. 69–86.

[96] S. Garg, A. Jain, J. Lou, and M. Nahmias, "Confounding tradeoffs for neural network quantization," *arXiv preprint arXiv:2102.06366*, 2021.

[97] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Improving post training neural quantization: Layer-wise calibration and integer programming," *arXiv preprint arXiv:2006.10518*, 2020.

[98] J. H. Lee, S. Ha, S. Choi, W.-J. Lee, and S. Lee, "Quantization for rapid deployment of deep neural networks," *arXiv preprint arXiv:1810.05488*, 2018.

[99] E. Meller, A. Finkelstein, U. Almog, and M. Grobman, "Same, same but different: Recovering neural network quantization error through weight factorization," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4486–4495.

[100] G. Shomron, F. Gabbay, S. Kurzum, and U. Weiser, "Post-training sparsity-aware quantization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 17 737–17 748, 2021.

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

[102] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[103] Y. Wei, X. Pan, H. Qin, W. Ouyang, and J. Yan, "Quantization mimic: Towards very tiny cnn for object detection," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 267–283.

[104] R. Li, Y. Wang, F. Liang, H. Qin, J. Yan, and R. Fan, "Fully quantized network for object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2810–2819.

[105] Z. Dong, Z. Yao, D. Arfeen, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq-v2: Hessian aware trace-weighted quantization of neural networks," *Advances in neural information processing systems*, vol. 33, pp. 18 518–18 529, 2020.

[106] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 293–302.

[107] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-bert: Hessian based ultra low precision quantization of bert," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 8815–8821.

[108] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 535–541.

[109] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[110] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 473–480.

[111] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *Artificial intelligence and statistics*. PMLR, 2009, pp. 153–160.

[112] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3354–3361.

[113] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," *arXiv preprint arXiv:1511.06530*, 2015.

[114] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2921–2929.

[115] H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei, "Relation networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3588–3597.

[116] H. Zhang, H. Chang, B. Ma, N. Wang, and X. Chen, "Dynamic r-cnn: Towards high quality object detection via dynamic training," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV 16*. Springer, 2020, pp. 260–275.

[117] X. Lu, B. Li, Y. Yue, Q. Li, and J. Yan, "Grid r-cnn," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7363–7372.

[118] C. Zhu, Y. He, and M. Savvides, "Feature selective anchor-free module for single-shot object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 840–849.

[119] Y. Cao, J. Xu, S. Lin, F. Wei, and H. Hu, "Gcnet: Non-local networks meet squeeze-excitation networks and beyond," in *Proceedings of the IEEE/CVF international conference on computer vision workshops*, 2019, pp. 0–0.

[120] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.

[121] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, "Accelerating the machine learning lifecycle with mlflow." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.