

DOCTORAL THESIS

Efficient Deep Learning Model Optimization for Resource Constrained Devices

Olutosin Ajibola Ademola

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
37/2025

Efficient Deep Learning Model Optimization for Resource Constrained Devices

OLUTOSIN AJIBOLA ADEMOLA



TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems

**The dissertation was accepted for the defence of the degree of Doctor of Philosophy in
Computer Systems Engineering, on 5th March 2025**

Supervisors: Tenured Full Professor Eduard Petlenkov, Ph.D.
Department of Computer Systems, School of Information Technologies
Tallinn University of Technology
Tallinn, Estonia

Mairo Leier, Ph.D.
Department of Computer Systems, School of Information Technologies
Tallinn University of Technology
Tallinn, Estonia

Opponents: Professor Andrei Lobov, Ph.D.
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

Professor Sergio Galvez Rojas, Ph.D.
Universidad de Málaga
Malaga, Spain

Defence of the thesis: 13th June 2025, Tallinn

Declaration:

Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Olutosin Ajibola Ademola

signature

Copyright: Olutosin Ajibola Ademola, 2025
ISSN 2585-6898 (publication)
ISBN 978-9916-80-319-6 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9916-80-320-2 (PDF)
DOI <https://doi.org/10.23658/taltech.37/2025>
Printed by Koopia Niini & Rauam

Ademola, O. A. (2025). *Efficient Deep Learning Model Optimization for Resource Constrained Devices* [TalTech Press]. <https://doi.org/10.23658/taltech.37/2025>

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
37/2025

Tõhus süvaõppe mudeli optimeerimine piiratud ressurssidega seadmete jaoks

OLUTOSIN AJIBOLA ADEMOLA

Contents

Abstract.....	8
Kokkuvõte	9
LIST OF PUBLICATIONS	10
AUTHOR'S CONTRIBUTION TO THE PUBLICATIONS	11
Abbreviations.....	12
1 INTRODUCTION.....	13
1.1 Background and Motivation	13
1.2 Literature Review.....	14
1.2.1 Bit Reduction	14
1.2.2 Knowledge Distillation	15
1.2.3 Low Rank Tensor Decomposition	16
1.2.4 Pruning.....	16
1.2.5 Microarchitecture	17
1.2.6 Ensemble of Compression Methods.....	18
1.3 Research Gaps and Research Questions	19
1.4 Objectives and Contributions of the Thesis	20
1.5 Thesis Structure	22
2 PRELIMINARIES	23
2.1 Model Compression Methods	23
2.1.1 Quantization	23
2.1.1.1 Binarization: An Extreme Case of Quantization.....	25
2.1.1.2 Challenges and Considerations	25
2.1.1.3 8-bit Quantization Impact on CNN Models	26
2.1.2 Knowledge Distillation	27
2.1.2.1 Challenges and Considerations	28
2.1.3 Low-Rank Tensor Decomposition	29
2.1.3.1 Tensor Train Decomposition	30
2.1.3.2 Challenges and Considerations	31
2.1.4 Pruning.....	31
2.1.4.1 Challenges and Considerations	32
2.1.5 Microarchitecture	33
2.1.5.1 Challenges and Considerations	34
2.1.6 Ensemble of Compression Methods.....	35
2.1.6.1 Challenges and Considerations	36
3 COMPRESSION METHODS EVALUATION USING NUMERICAL-BASED MATHEMATICAL APPROACH	37
3.1 Evaluation of Deep Neural Network Compression Methods for Edge Devices Using Weighted Score-Based Ranking Scheme	37
3.2 Evaluation Metrics	37
3.3 Limitations of Current Metrics	38
3.4 The Numerical-Based Mathematical Approach	38
3.4.1 Mathematical Formulation	38

3.4.1.1	Definition of Parameters	38
3.5	Weighted Score-Based Ranking Scheme	39
3.5.1	Scaling.....	39
3.5.2	Scoring	40
3.5.2.1	Adapting Scores for Metric Characteristics	40
3.5.3	Scoring Formula	40
3.5.4	Weighting	41
3.5.5	Weighted Score	41
3.5.6	Weight Score Average	42
3.5.7	Ranking	42
3.5.8	Application of Weighted Score-Based Ranking Scheme	42
3.5.8.1	Benchmark Model Description	42
3.5.8.2	Compression Methods Evaluated	42
3.5.8.3	Scoring and Weighting Process	44
3.5.8.4	Weights Assignment Strategy	45
3.6	Results and Discussion	46
3.7	Summary	49
3.8	Conclusion	50
4	ENSEMBLE OF COMPRESSION METHODS	51
4.1	Limitations of Single Compression Methods.....	51
4.2	Ensemble of Tensor Train Decomposition and Quantization	51
4.2.1	Architecture of the Ensemble Compression Pipeline	52
4.2.2	Challenges of the Ensemble Method	53
4.3	Experimental Setup and Results	53
4.3.1	Architecture of the Baseline Model	53
4.3.2	Ensemble End-to-End Trainable Pipeline	53
4.3.3	Training and Optimization Process	54
4.3.4	Tensor Train Configuration	55
4.3.5	Tensor Trained Model Quantization	56
4.3.6	Performance Comparison and Evaluation of Key Metrics	56
4.4	Summary and Conclusion	58
4.4.1	Summary.....	58
4.4.2	Conclusion	58
5	PRACTICAL USE CASE: OPTIMIZATION LIMITATION OF SCENE TEXT DETECTION AND RECOGNITION MODELS	59
5.1	Resource-Aware Scene Text Recognition Using Learned Features, Quantization, and Contour-Based Character Extraction	59
5.2	Scene Text Recognition In Embedded Systems	59
5.3	Resource Constraints in Embedded Hardware.....	59
5.4	Challenges and Research Objectives	60
5.4.1	Challenges in Optimizing Scene Text Detection and Recognition Models	60
5.4.2	Research Objectives	60
5.5	Scene Text Detection and Recognition	61
5.5.1	Text Detection	61
5.5.2	Text Recognition	62
5.6	Proposed Methodology	62
5.6.1	System Architecture Overview.....	62

5.6.2	Text Detection: Modified EAST Architecture	63
5.6.3	Quantization for Integer-Only Hardware	64
5.6.4	Text Recognition: Contour-Based Character Extraction	65
5.7	Experiments	67
5.7.1	Experimental Setup	67
5.7.2	Evaluation Metrics.....	68
5.8	Results and Discussion	69
5.9	Summary and Conclusion	70
5.9.1	Summary.....	70
5.9.2	Conclusion	70
6	CONCLUSION	71
	List of Figures	72
	List of Tables	73
	References.....	74
	Acknowledgements	80
	Appendix 1.....	81
	Appendix 2	101
	Appendix 3	109
	Curriculum Vitae	121
	Elulookirjeldus.....	122

Abstract

Efficient Deep Learning Model Optimization for Resource Constrained Devices

This thesis addresses critical challenges in efficient deep learning (DL) model compression and optimization, which are essential for deploying models on resource-constrained hardware. Despite significant advancements, several key research gaps remain: the lack of a standardized quantitative method for evaluating and comparing compression techniques, the difficulty of achieving deep compression without significant performance loss, and the challenge of optimizing models that are highly sensitive to compression. This thesis explores these gaps through three primary research questions.

To tackle these challenges, the thesis first develops and validates a robust numerical-based method for objectively evaluating and ranking model compression techniques. This method, introduced in Publication 1, uses a weighted score-based ranking system to assess the efficiency, effectiveness, and suitability of each technique across various applications. It provides a systematic and adaptable method for selecting optimal compression strategies, enabling informed decision-making tailored to diverse use cases.

The second research objective focuses on achieving a deep compression ratio of 32x while maintaining acceptable performance. This was accomplished through an innovative ensemble technique combining Tensor Train Decomposition (TTD) and 8-bit quantization, as detailed in Publication 2. This approach significantly exceeded expectations, achieving a remarkable 57x compression ratio, demonstrating the feasibility of high-ratio compression for real-world deployment without substantial loss in accuracy.

The final research objective addresses the optimization of models that are highly sensitive to compression and quantization. Publication 3 introduces a novel quantization offset technique, known as quantization bias, to mitigate the sensitivity of state-of-the-art scene text detection and recognition models to quantization. This technique enabled efficient deployment on integer-only hardware with minimal performance loss. Additionally, an integrated text orientation detection module enhances the model's capability to handle diverse orientations, broadening its applicability across different scenarios.

The cumulative contributions of this thesis provide a comprehensive methodology for evaluating, compressing, and optimizing DL models for deployment on low-power, low-cost hardware platforms. The proposed solutions not only advance the state-of-the-art in model compression but also ensure that even architectures sensitive to compression can be effectively optimized without compromising performance. These findings have significant implications for real-world applications, enabling high-performance DL models to operate efficiently in environments with stringent computational and memory constraints.

Kokkuvõte

Tõhus süvaõppe mudeli optimeerimine piiratud ressurssidega seadmete jaoks

See lõputöö käsitleb kriitilisi väljakutseid tõhusa sügava õppimise (DL) mudelite tihendamisel ja optimeerimisel, mis on hädavajalikud mudelite juurutamiseks piiratud ressurssidega riistvarale. Vaatamata märkimisväärtetele edusammudele on endiselt mitmeid olulisi uurimislünki: standardiseeritud kvantitatiivse meetodi puudumine tihendustehnikate hindamiseks ja võrdlemiseks, raskused sügava tihendamise saavutamisel ilma märkimisväärt jõudluse vähenemiseta ja tihendamise suhtes ülitundlike mudelite optimeerimise väljakutse. See uuring uurib neid lünki kolme peamise uurimisküsimuse kaudu.

Nende väljakutsetega toimetulemiseks töötatakse lõputöös esmalt välja ja valideeritakse tugev numbripõhine meetod mudeli tihendamise tehnikate objektiivseks hindamiseks ja järjestamiseks. See meetod, mida tutvustati, valideeritakse I publikatsioonis, kasutab kaalutud skooripõhist järjestussüsteemi, et hinnata iga tehnika tõhusust, tulemuslikkust ja sobivust erinevates rakendustes. See pakub süstemaatilist ja kohandatavat raamistikku optimaalsete tihendusstrateegiate valimiseks, võimaldades teha teadlikke otsuseid, mis on kohandatud erinevatele kasutusjuhtudele.

Teine uurimiseesmärk keskendub sügava tihendusastme 32-kordsele saavutamisele, säilitades samal ajal vastuvõetava jõudluse. See saavutati uuendusliku ansamblitehnika abil, mis ühendas Tensor Train Decomposition (TTD) ja 8-bitise kvantimise, nagu on üksikasjalikult kirjeldatud II publikatsioonis. See lähenemine ületas oluliselt ootusi, saavutades märkimisväärt 57-kordse tihendusastme, mis näitab suure tihendussuhte teostatavust reaalseks kasutuselevõtuks ilma täpsuse olulise vähenemiseta.

Lõplik uurimiseesmärk käsitleb tihendamise ja kvantiseerimise suhtes ülitundlike mudelite optimeerimist. III publikatsioon tutvustab uutset kvantimise nihketehnikat, mida tuntakse kvantimise eelarvamusega, et leevendada tipptasemel stseeni tekstituvastus- ja tuvastusmudelite tundlikkust kvantimise suhtes. See tehnika võimaldas tõhusat juurutamist ainult täisarvudega riistvaras minimaalse jõudluskahuga. Lisaks suurendab integreeritud teksti orientatsiooni tuvastamise moodul mudeli võimet käsitleda erinevaid orientatsioone, laiendades selle rakendatavust erinevates stsenaariumides.

Käesoleva teadustöö kokkuvõtvad panused pakuvad põhjalikku metoodikat süvaõppe mudelite hindamiseks, tihendamiseks ja optimeerimiseks, et neid kasutada madala võimsusega ja odavatel riistvaraplatvormidel. Pakutud lahendused mitte ainult ei edenda mudeli tihendamise tipptasemel tehnoloogiat, vaid tagavad ka selle, et isegi tihendamise suhtes tundlikke arhitektuure saab tõhusalt optimeerida ilma jõudlust vähendamata. Need tulemused mõjutavad märkimisväärselt reaalmaailma rakendusi, võimaldades suure jõudlusega süvaõppe mudelitel tõhusalt töötada keskkondades, kus on ranged arvutus- ja mälupiirangud.

LIST OF PUBLICATIONS

The present Ph.D. thesis is based on the following publications that are referred to in the text by Roman numbers.

- I Ademola, O.A.; Leier, M.; Petlenkov, E. Evaluation of Deep Neural Network Compression Methods for Edge Devices Using Weighted Score-Based Ranking Scheme. *Sensors* 2021, 21, 7529. <https://doi.org/10.3390/s21227529>
- II O. A. Ademola, E. Petlenkov, and M. Leier, "Ensemble of Tensor Train Decomposition and Quantization Methods for Deep Learning Model Compression", 2022 International Joint Conference on Neural Networks (IJCNN), Padua, Italy, 2022, pp. 1-6, doi: 10.1109/IJCNN55064.2022.9892626
- III O. A. Ademola, E. Petlenkov, and M. Leier, "Resource-Aware Scene Text Recognition Using Learned Features, Quantization, and Contour-Based Character Extraction," in *IEEE Access*, vol. 11, pp. 56865-56874, 2023, doi: 10.1109/ACCESS.2023.3283931.

AUTHOR'S CONTRIBUTION TO THE PUBLICATIONS

- I In **Publication I**, as the first author, I developed a numerical-based mathematical method that utilizes a weighted score-based ranking system to evaluate deep learning model compression methods. This approach allows for the ranking of each compression method based on their weighted scores, tailored to the specific requirements of the application. I also authored the manuscript.
- II In **Publication II**, as the first author, I developed an ensemble technique for deep compression of deep learning models that maintains a modest model accuracy. This technique combines tensor train decomposition with 8-bit quantization, informed by the findings from **Publication I**. The ensemble method achieved a 57x compression ratio, significantly surpassing the 4x ratio, which was the highest achieved by any single compression method. I also authored the manuscript.
- III In **Publication III**, as the first author, I addressed the sensitivity issue associated with quantizing state-of-the-art scene text detection and recognition models by introducing a quantization offset (also referred to as a quantization bias). This innovation enhanced the model robustness and sensitivity, enabling efficient quantization for integer-only hardware with minimal performance degradation. Additionally, I developed a text orientation detection module to support both portrait and landscape text orientations. I authored the manuscript.

Abbreviations

Acc	Accuracy
AI	Artificial Intelligence
ANN	Artificial Neural Networks
BN	Batch Normalization
CNN	Convolutional Neural Network
CONV2D	2D Convolution
CP	CANDECOMP/PARAFAC Decomposition
CR	Compression Ratio
CV	Computer Vision
DNN	Deep Neural Network
DL	Deep Learning
EAST	Efficient and Accurate Scene Text
FC	Fully Connected (Layer)
FLOP	Floating Point Operations
FLOPS	Floating Point Operations Per Second
IT	Inference Time
KN	Knowledge Distillation
ML	Machine Learning
MAXPOOL2D	2D Max Pooling
NAS	Neural Architecture Search
NN	Neural Network
NLP	Natural Language Processing
PMF	Peak Memory Footprint
QAT	Quantization-Aware Training
QuanTT	Quantized Tensor-Trained Model
RQ	Research Question
RO	Research Objective
R-CNN	Region-based Convolutional Neural Network
RNN	Recurrent Neural Network
RPN	Region Proposal Network
STE	Straight-Through Estimator
TD	Tensor Decomposition
TT	Tensor Train
TTD	Tensor Train Decomposition
TTP	Tensor Train Representation
VDS	Variational Dropout Sparsification

1 INTRODUCTION

1.1 Background and Motivation

In an increasingly connected world, the proliferation of smart devices has generated an unprecedented demand for technologies capable of processing complex data in real-time at the edge. Marco et al. [1] highlighted adaptive model selection techniques to optimize deep learning inference on embedded systems, addressing the limitations of limited computational power. Han et al. [2] proposed MCDNN, a framework for efficient stream processing under resource constraints, utilizing approximation-based methods to enhance the performance of the deep learning model. Liu et al. [3] advanced our understanding of usage-driven model selection for mobile devices, emphasizing dynamic adjustments to model compression for improved efficiency.

Deep neural networks (DNNs) have demonstrated remarkable success in driving advancements across various domains. Lukas [4] examined efficient online processing methodologies for neural networks, emphasizing their role in real-time applications. Zhang et al. [5] investigated compilation and optimization strategies to deploy machine learning models in embedded systems, providing information about overcoming resource limitations. Cai et al. [6] presented an extensive review of enabling deep learning on mobile devices, discussing methods such as neural architecture search and model compression to improve applicability. Nagarhalli et al. [7] reviewed the impact of machine learning in natural language processing (NLP), underlining the contributions of DNNs to tasks such as speech recognition and sentiment analysis. Saptarshi et al. [8] surveyed recent trends in deep learning architectures and applications, demonstrating their transformative potential in various fields. Minar et al. [9] provided an overview of recent advances in deep learning, contextualizing its applications in modern computing.

Deep learning (DL), a subset of machine learning (ML), functions as a computational approach inspired by the structure and function of the brain's neural networks (NNs). Zhai et al. [10] explored the integration of AI in education, providing a broader perspective on its potential applications. Panch et al. [11] discussed the synergy between artificial intelligence and health systems, highlighting the pivotal role of deep learning in decision-making processes. Ademola et al. [12] focused on resource-aware scene text recognition, detailing how learned features and quantization techniques enhance performance on constrained devices. Ademola et al. [13] further explored tensor train decomposition and quantization methods, showcasing an ensemble approach for deep learning model compression. Cai et al. [6] emphasized the hierarchical nature of neural network feature extraction, which enables powerful processing of unstructured data such as text and images.

DL applications extend across computer vision (CV), NLP, healthcare, finance, and entertainment. In CV, Wu et al. [14] demonstrated the application of DL in tasks such as object detection and facial recognition. Rawat et al. [15] provided a comprehensive review of convolutional neural networks for image classification, highlighting their robustness in complex visual tasks. In NLP, Alshahrani et al. [16] evaluated DL's capabilities for language translation and sentiment analysis, while Pattanayak et al. [17] emphasized the utility of recurrent neural networks for text processing. Hirschberg et al. [18] discussed DL advancements in NLP, offering insight into state-of-the-art methodologies. Li et al. [19] reviewed the integration of deep learning in signal processing, including its applications in speech and text analysis.

In healthcare, Miotto et al. [20] reviewed the transformative role of deep learning in disease diagnosis and personalized treatments. Chensi et al. [21] examined its applications in bioinformatics and genomics, demonstrating the adaptability of DL to various

biomedical challenges. Khader et al. [22] explored the contributions of machine learning to cardiovascular medicine, highlighting its clinical implications. Liu et al. [23] conducted a meta-analysis comparing DL's diagnostic performance against healthcare professionals, establishing its potential in medical imaging. Sendak et al. [24] critically evaluated the challenges and opportunities of integrating ML into healthcare systems.

In finance, Eunsuk et al. [25] demonstrated the use of DL for analysis and prediction of the stock market. Sohangir et al. [26] presented a framework for financial sentiment analysis, leveraging deep learning for better forecasting. Ahmet et al. [27] surveyed the financial applications of DL, including algorithmic trading and risk management. Polat et al. [28] examined ML algorithms for stock trading strategies, underlining DL's advantages in detecting complex patterns.

Entertainment platforms widely leverage DL for personalization and user engagement. Zhang et al. [29] conducted a survey on DL-based recommender systems, providing a foundation for their use on platforms such as Netflix and Spotify. Dellal-Hedjazi et al. [30] explored the role of collaborative filtering in recommendation systems, emphasizing its synergy with DL techniques. Liu et al. [31] presented a survey of DL applications in recommendation systems, describing their ability to effectively predict user preferences. Anantrasirichai et al. [32] reviewed the role of AI in creative industries, including its impact on content generation and recommendation systems. Ying et al. [33] highlighted the scalability of graph convolutional neural networks for web-scale recommendation systems.

Despite these successes, deploying DL models on resource-constrained devices remains challenging. Zhang et al. [5] analyzed the trade-offs in adapting ML to embedded systems, offering strategies for model optimization. Cai et al. [6] emphasized lightweight architectures to improve efficiency on mobile platforms. Mujtaba et al. [34] optimized CNN dataflow to streamline computations and applied lightweight kernel transformations, ensuring efficiency without compromising accuracy. Liang et al. [35] provided a detailed survey of pruning and quantization, emphasizing their importance in reducing computational demands while maintaining model performance.

In conclusion, the integration of DL in resource-constrained environments requires balancing computational efficiency, memory usage, and energy consumption. This thesis contributes to advancing compression techniques tailored for these challenges, ensuring DL's scalability and effectiveness in diverse applications.

1.2 Literature Review

Model compression for resource-constrained devices has been a vital area of research to bring AI to the edge. The state-of-the-art methods for model compression can be categorized as follows —bit reduction, knowledge distillation, low-rank tensor decomposition, pruning, microarchitecture (i.e. compact model design) and ensemble of methods. Each offers distinct advantages and trade-offs, contributing to the development of resource-aware and efficient models for resource-constrained devices.

1.2.1 Bit Reduction

Bit reduction (i.e., quantization, ternarization, binarization) is a prominent technique in model compression, primarily aimed at reducing the size and computational complexity of DNNs. It involves approximating 32-bit full-precision weights and activations with lower-bit representations (e.g., 8-bit, 2-bit),

Jacob et al. [36] quantized the weights and activations using 8 bit, but maintained the bias vector in 32 bit integer. Banner et al. [37] represents the weights and activations using 4-bit with a minimal decrease in accuracy.

Liu et al. [38] trained a ternary network using 2-bit. Thus, reducing the memory footprint when compared to using binary weights representation.

To address the decrease in accuracy associated with post-training quantization, Jacob et al. [36] simulated the effect of quantization during training by rounding up full precision weights and activations. This approach is referred to as quantization-aware training. Courbariaux et al. [39] and Mohammad et al. [40] restricted the weights to binary values and reported near-state-of-the-art results.

Yongxin et al. [41] introduce a novel approach to overcome representation collapse in Vector Quantization (VQ) models by reparameterizing code vectors using a learnable linear transformation layer, enabling optimization of the entire codebook space and enhancing scalability, adaptability, and performance across various applications.

Bit reduction has demonstrated significant potential for compressing large DL models, making it ideal for resource-constrained hardware. However, it presents challenges, such as accuracy degradation, especially with aggressive bit reduction (e.g., ternary, binary networks).

Another issue is hardware heterogeneity, as different devices support varying levels of quantization. Hardware efficiency also varies across different hardware architectures, with some platforms unable to fully utilize integer-only precision.

Finally, quantization sensitivity varies by model and tasks. For instance, models with large weight variations, such as scene text detection and recognition, are more sensitive to quantization compared to models for image classification. This requires task-specific optimization, which can be very complex and challenging.

1.2.2 Knowledge Distillation

Knowledge distillation (KD) is a model compression technique that has gained prominence in DL research due to its simplicity and effectiveness. The idea of KD, initially proposed by Geoffrey et al. [42], involves transferring the knowledge from a large and complex model (ie, teacher) to a smaller and compact model (ie, student) with small computational and memory footprints. The process allows the student model to approximate the performance of the teacher model.

The concept of KD was rooted in the idea that large models often capture intricate patterns and relationships in data that smaller models struggle to learn directly. Geoffrey et al. [42] proposed training a student model by mimicking the soft predictions (probability distributions) of the teacher model, rather than just learning from hard labels. This approach takes advantage of the dark knowledge embedded in the teacher's output probabilities, allowing the student model to generalize better than it would by relying only on labeled data. Their work demonstrated that KD could achieve the predictive capacity of ensembles using smaller and compact models.

The researchers explored various strategies to improve the effectiveness of KD. One such strategy is collaborative teacher-student mutual learning, where both models learn simultaneously, and the knowledge transfer is bidirectional [43]. Zhang et al. [44] proposed a self-distillation framework in which a model distills the knowledge of itself over multiple iterations or layers, effectively refining its own predictions without requiring a separate teacher model. This approach has shown potential for further improving model performance without increasing complexity.

KD still presents some challenges. One major issue is the performance gap between the teacher and student models. Also, KD does not guarantee that the student model will reach the teacher's performance level. This is particularly evident when the difference in model expressivity is large.

Another challenge is the effectiveness of distillation across different domains. Although KD has shown remarkable success in image classification, its application in more complex tasks, such as natural language processing and scene detection & recognition, has been less straightforward.

1.2.3 Low Rank Tensor Decomposition

Low-rank tensor decomposition (TD) is a mathematical technique that has gained traction in compression of the DL model. It extends the concept of matrix decomposition to higher-dimensional data representations (tensors). Low-rank TD leverages to reduce the number of parameters of the models through a low-rank approximation.

Vadim et al. [45] decomposed a 4D convolutional kernel into a sum of vector products using CP decomposition (CANDECOMP/PARAFAC decomposition). CP decomposes a tensor into a sum of rank one tensors. It is one of the simplest forms of tensor decomposition and is widely used due to its simplicity. However, CP may not always provide the best approximation for complex tensors present in DL models.

Yong-Deok et al. [46] applied Tucker decomposition (TD) to deep CNNs, demonstrating significant reductions in both the number of parameters and computational cost, while maintaining competitive accuracy. TD generalizes CP decomposition by allowing the tensor to be decomposed into a core tensor and multiple factor matrices. TD offers more flexibility in the approximation of tensors and has been shown to achieve better compression ratios with less loss of accuracy compared to CP decomposition.

Novikov et al. [47] employed the decomposition of the tensor train (TT) in the NNs, reducing the number of parameters in the fully connected layers by orders of magnitude without significant accuracy loss. TT represents a tensor as a sequence of low-rank matrices. This approach is particularly efficient for very large tensors, as it scales linearly with the tensor dimensions.

Low-rank TD offers significant advantages; however, several challenges still exist. One key issue is the accuracy trade-offs, as the compression ratio scales with the tensor rank. Another challenge is computational complexity during the process of decomposing a tensor, especially for high-dimensional data, which can be computationally expensive. This complexity can offset some of the gains achieved through model compression, particularly during training.

Choosing the appropriate rank for decomposition is also critical. A lower rank may result in more compression, but at the cost of accuracy degradation, while a higher rank may preserve accuracy but limit the compression benefits. Automated methods for rank selection are still an area of active research.

Lastly, decomposed models may not generalize well across different tasks or datasets, particularly if the decomposition was tuned for a specific application. Ensuring that the compressed model retains its generalization capabilities is a key challenge.

1.2.4 Pruning

Pruning is another compression technique employed in DL, primarily aimed at reducing the size and computational footprint of NNs. By eliminating redundant or less significant parameters, pruning creates efficient models that require less memory and computational power, making them suitable for deployment on resource-constrained devices such as mobile phones, embedded systems, and IoT devices. The method has gained significant attention because of its ability to retain the accuracy of the models while reducing the model size and inference time.

Early work on NN pruning dates back to the 1980s and 1990s, with studies by LeCun

et al. [48] and Hassibi et al. [49] introducing the concept of optimal brain damage and optimal brain surgeon, respectively. These methods aimed to prune network connections based on their contribution to the loss function, effectively removing those that had minimal impact on the model performance.

These early approaches laid the groundwork for more sophisticated pruning techniques, emphasizing the potential of pruning, not just for reducing the model size but also for improving generalization by eliminating unnecessary complexity. Song et al. [50] combined pruning, quantization, and Huffman coding and achieved significant reductions in model size.

Hao et al. [51] introduced a method to prune convolutional filters based on their L1 norm, demonstrating that all filters could be removed with minimal impact on the accuracy of the model. Jonathan et al. [52] introduced the "Lottery Ticket Hypothesis," suggesting that within large networks there exist small, randomly initialized subnetworks that, when trained in isolation, can achieve performance comparable to the original network. This hypothesis has inspired numerous studies on iterative pruning during training.

Lin et al. [53] present the design of a sparsity-aware deep learning hardware accelerator that takes advantage of both data and weight sparsity in CNN models. Louizos et al. [54] exploit L0 regularization, which yielded inconsistent compression in larger datasets, while simple magnitude pruning achieves comparable or better results.

Network pruning presents some challenges. One major challenge is accuracy loss, particularly when aggressive pruning strategies are employed. Although fine-tuning can often recover lost accuracy, there is a limit to how much pruning can be done before the model's performance deteriorates significantly.

Another challenge is the trade-off between sparsity and efficiency. Unstructured pruning, while effective at reducing the number of parameters, can lead to sparse weight matrices that are difficult to optimize on standard hardware, limiting the potential speedup. Structured pruning, although more hardware-friendly, may require more sophisticated criteria to determine which filters or neurons to prune, as these components often capture complex, high-level features.

Generalization and overfitting are also concerns in network pruning. As pruning reduces the number of parameters, it can lead to overfitting, particularly if the remaining parameters are not sufficient to capture the underlying patterns in the data. This is particularly problematic in small datasets or when transferring pruned models to new domains or tasks.

1.2.5 Microarchitecture

Microarchitectures (i.e., compact models) involve the design of small, compact, and efficient model architectures. This approach is based on residual domain knowledge of the blocks needed in the design of NN architectures. Microarchitecture differs from the other compression methods (i.e., KD, pruning, quantization, TD etc.), because it does not rely on any external compression.

Forrest et al. [55] achieved a similar accuracy with 50x fewer parameters obtained by Krizhevsky et al. [56]. Andrew et al. [57] proposed MobileNets that utilize depth-wise separable and point-wise convolutions to reduce the number of multiplications. These CNN microarchitectures have become the state-of-the-art for image classification and object detection base models. Other proposed compact architectures include SqueezeNet[55], ShuffleNet [58], EfficientNet [59], and TinyYOLO [60]. All efficient for edge deployments.

Despite the success of compact model design, several challenges persist, including balancing the accuracy-efficiency trade-off, where compact models often struggle to achieve

the performance of larger models on complex tasks. Furthermore, ensuring generalization across diverse tasks and datasets remains difficult, particularly when models are heavily optimized for specific applications. Scalability also presents a challenge, as adapting compact models for more powerful hardware without losing efficiency gains requires further research.

1.2.6 Ensemble of Compression Methods

Ensemble combines multiple model compression techniques to achieve a better balance between model size, inference time, memory, and accuracy. Using the strengths of various compression techniques, such as pruning, quantization, TD, and KD, it mitigates the limitations/weakness of the individual method. By integrating these methods, the ensemble technique can effectively compress models while preserving performance, making them particularly useful for resource-constrained devices.

One common ensemble method is the combination of pruning and quantization. Pruning reduces the number of parameters in a model by removing less important/contributing weights, while quantization reduces the precision of the remaining weights. Song et al. [50] demonstrated that combining this technique can significantly reduce the size of neural networks with minimal impact on accuracy. Their work on the "Deep Compression" framework shows that pruning followed by quantization leads to more efficient models without significant performance loss.

Another effective ensemble technique involves combining KD and quantization. KD transfers knowledge from a large and complex model to a compact model. The compact model is then further compressed by quantization. Mishra et al. [61] explored this combination, showing that quantization-aware distillation helps maintain accuracy while allowing lower bit quantization of the student model.

More recent research explores the integration of all three methods —KD, pruning, quantization. Li et al. [62] proposed a comprehensive approach that first prunes the model, distills the knowledge from the pruned model, and then applies quantization. This multistep process helps maintain the model accuracy while reducing the model size and improving inference speed. Their results indicate that the ensemble method can outperform individual techniques in terms of compression efficiency and model performance.

Lin et al. [63] introduced an approach that integrates pruning, quantization, and search for neural architecture (NAS). This method allows for automatic model design tailored to specific hardware constraints, achieving the state-of-the-art compression rates with minimal accuracy degradation.

The ensemble of compression methods, while very powerful, face challenges such as increased complexity and implementation overhead, as integrating multiple techniques (e.g., pruning, quantization, knowledge distillation, low-rank tensor decomposition) requires careful coordination and consideration. Compatibility issues can also arise when different methods interfere with each other, and the sequential application of these techniques often leads to longer training times.

Task-specific tuning is also needed to optimize the ensemble for different models, which can be time consuming. Hardware constraints may limit the benefits of compression, and there is a risk of over-compression, leading to accuracy loss.

Lastly, evaluating and benchmarking ensemble methods is difficult due to the lack of standardized metrics that capture the trade-offs between compression, accuracy, and performance across different hardware platforms.

Table 1 provides a summary of the state-of-the-art of model compression methods. Each method is described in terms of its key contributions and the main challenges asso-

ciated with it, along with relevant references.

Table 1: Summary of key literature on various model compression methods, describing their contributions and challenges.

Technique	Year	Key Contributions	Challenges
Bit Reduction	2015–2024	Enabled reduction in model size by approximating 32-bit full-precision weights and activations with lower-bit representations (e.g., 16-bit, 8-bit, 4-bit) [36]. Extreme cases such as binarization and ternarization further reduce memory requirements and inference speed [39, 40]. Techniques like post-training quantization [37] have been explored for efficient deployment. Vector quantization was introduced to improve inference time of the model by reparameterizing code vectors using a learnable linear transformation layer [41].	Balancing accuracy and compression, especially for extremely low-bit formats, is challenging. Additionally, hardware platforms often lack robust support for efficient low-bit operations, complicating deployment.
Knowledge Distillation	2015–2021	Facilitated knowledge transfer from larger, complex teacher models to smaller, simpler student models to maintain accuracy while reducing size [42]. Techniques such as collaborative teacher-student learning [43] and self-distillation [44] further optimize the distillation process.	Designing effective teacher-student architectures is critical. Optimizing the knowledge transfer process to ensure student models retain performance remains a challenge.
Pruning	1989–2019	Reduced network size by removing less significant weights, neurons, or filters using unstructured pruning [48, 49] or structured pruning [50, 51]. Techniques like the lottery ticket hypothesis [52] and variational dropout [54] provided new methods to identify redundant parameters.	Identifying redundant parameters and balancing compression with accuracy remain significant challenges. Efficient pruning algorithms that generalize across different architectures are needed.
Low Rank Tensor Decomposition	2015–2019	Applied matrix factorization techniques to decompose tensors in convolutional and fully connected layers, reducing parameters [45]. Hybrid tensor decomposition approaches [46] and Tensor Train Decomposition frameworks [47] have been developed to optimize computational efficiency.	Selecting appropriate decomposition methods for specific architectures, handling non-linear layers, and maintaining performance are key challenges.
Microarchitecture	2016–2020	Designed lightweight CNN architectures (e.g., MobileNet [57], SqueezeNet [55], ShuffleNet [58], and EfficientNet [59]) that achieve high accuracy with fewer parameters. These architectures often reduce dependency on external compression techniques [55, 60].	Balancing model complexity, accuracy, and generalization is challenging. Adapting models for diverse tasks and hardware environments adds further complexity.
Ensemble of Model Compression	2016–2020	Combined multiple compression techniques such as pruning, quantization, and knowledge distillation for greater reductions in model size while maintaining accuracy [50, 61]. Approaches like dynamic knowledge distillation [62] have demonstrated improved compression outcomes.	Coordinating and managing interactions between different compression techniques is complex. Ensuring overall performance in varied deployment scenarios is a major challenge.

1.3 Research Gaps and Research Questions

Efficient model compression has emerged as a pivotal area of research, driven by the growing demand for resource-aware AI solutions capable of operating on low-cost, low-power embedded hardware. According to the surveyed literature, several key research gaps have been identified.

One significant gap is the lack of a standardized, quantitative methodology for assessing the efficiency, effectiveness, reliability, and suitability of DL compression methods. This absence leaves researchers and practitioners without a clear framework for objectively comparing various compression techniques. As a result, the selection of the most suitable method often becomes subjective, relying on inconsistent benchmarks or incomplete performance metrics.

Another critical challenge lies in achieving deep compression without loss of performance. Fitting large models onto extremely resource-constrained hardware while maintaining acceptable accuracy levels is a very challenging task. There is a pressing need to develop compression techniques that are both resource- and performance-aware, enabling efficient operation on devices with extreme storage and memory limitations without significant degradation in performance.

Balancing model optimization and accuracy also remains a persistent challenge. Optimizing deep learning models often introduces trade-offs, where reductions in computational or memory footprints can negatively impact model performance. For highly sensitive models, even minor adjustments may result in substantial performance degradation. Developing strategies that minimize resource usage while maintaining robust accuracy is crucial, ensuring that optimized models remain effective across diverse tasks.

These challenges highlight the gaps in current research and emphasize the need for new approaches to effectively solve them.

To address these gaps, this thesis aims to explore the following research questions.

Research Question 1 (RQ1): Can a numerically based mathematical method be developed to quantitatively evaluate state-of-the-art deep learning (DL) model compression techniques for diverse application requirements?

RQ1 focuses on developing a standardized numerical approach to objectively evaluate and rank compression methods. The goal is to establish a systematic evaluation framework that quantifies the impacts of various methods on the characteristics of the model, providing a clear basis for selecting the most appropriate techniques.

Research Question 2 (RQ2): Can a deep model compression ratio of 32x be achieved while maintaining an acceptable level of accuracy for practical deployment?

RQ2 examines the feasibility of achieving deep compression while preserving performance. It aims to explore the practical limits of compression techniques and their ability to maintain acceptable accuracy levels, offering insights into their application for resource-constrained hardware.

Research Question 3 (RQ3): Can efficient model optimization be achieved without significantly compromising performance, particularly for compression-sensitive architecture?

RQ3 investigates strategies for optimizing models that are particularly prone to performance degradation under compression. The focus is on balancing efficiency gains, such as reduced memory and computational requirements, with the need to preserve accuracy and generalizability, especially in sensitive architectures.

1.4 Objectives and Contributions of the Thesis

This thesis aims to address the research gaps outlined in the preceding section by exploring the three key research questions. This thesis is structured around three research objectives (**RO1, RO2, and RO3**), each designed to address respective highlighted research questions (**RQ1, RQ2, and RQ3**).

The objectives have resulted in contributions to efficient model compression for resource-constrained hardware, which have been validated through publications in peer-reviewed scientific journals and conference presentation. The cumulative findings and contributions are thoroughly highlighted and discussed in detail throughout this thesis.

Research Objective 1 (RO1): Develop and validate a robust, numerical-based mathematical method for objectively evaluating and ranking state-of-the-art model compression methods. This method will assess each method's impact on efficiency, effectiveness, and

suitability across a wide range of applications, providing a standardized, quantitative approach to guide informed decision-making in selecting optimal compression strategies.

This objective explores the feasibility of developing a standardized, robust, numerical-based method to objectively evaluate compression techniques based on their impact on various model characteristics, such as model size, inference time, memory, accuracy, etc.

The goal was to establish a systematic evaluation method that provides a clear and objective basis for evaluating the compression technique for different application requirements.

Contribution 1 (addressing RQ1): In **Publication I**, a numerical-based mathematical method utilizing a weighted score-based ranking system was developed to evaluate compression methods. Each method was meticulously evaluated and ranked according to a weighted scoring scheme, allowing for a nuanced evaluation that is finely tuned to the unique demands of various application contexts.

This innovative approach not only offers a systematic and objective means of assessing compression methods but also provides an adaptable framework that can be tailored to optimize performance for diverse application requirements. Using this ranking system, a more informed selection of compression techniques is enabled, ensuring that the most suitable and efficient methods are used for specific use cases.

Research Objective 2 (RO2): Achieve a deep model compression ratio of 32x while preserving acceptable levels of accuracy, thereby demonstrating the practicality and feasibility of high-ratio compression techniques for deep learning models.

This objective aims to explore the practical limits of model compression by evaluating whether a 32x compression ratio can be achieved without a significant loss of performance, particularly for critical edge applications. It seeks to assess the impact of extreme compression techniques on overall model accuracy, providing valuable insights into the potential to achieve highly efficient model compression.

Contribution 2 (addressing RQ2): In **Publication II**, I developed an ensemble technique that integrates TT decomposition with 8-bit quantization, building upon the findings from **Publication I**. This innovative approach achieved an impressive 57x compression ratio, significantly surpassing the 4x compression ratio achieved by individual compression methods. This demonstrates the superior efficacy of ensemble techniques in pushing the boundaries of model compression.

Research Objective 3 (RO3): Develop and implement advanced optimization techniques specifically designed for models with high sensitivity to compression and quantization. The objective is to achieve substantial reductions in computational and memory requirements while ensuring minimal to no degradation in accuracy and other critical performance metrics.

This research objective addresses the challenge of optimizing deep learning models that are inherently sensitive to precision changes due to their architecture. It seeks to push the limits of what is achievable in model compression and optimization, ensuring that even the most sensitive models can be efficiently deployed without sacrificing performance integrity.

Contribution 3 (addressing RQ3): In **Publication III**, the inherent sensitivity of cutting-edge scene text detection and recognition models to quantization was effectively mitigated by the introduction of a novel quantization offset technique, called quantization bias. This innovation significantly strengthened the robustness of the model, enabling efficient quantization for hardware with integer only with acceptable performance loss. In addition, a comprehensive text orientation detection module was incorporated, improving the model's ability to accurately process text in portrait and landscape orientations,

thus broadening its applicability in diverse scenarios.

1.5 Thesis Structure

The organization of this thesis is summarized as follows:

Chapter 1 provides an introduction to the background, motivation, research gaps, research questions, and objectives of the thesis. The chapter also outlines the contributions of the thesis and reviews the state of the art in DL compression methods for resource-constrained devices.

Chapter 2 establishes the foundation for understanding efficient optimization of the DL model in resource-constrained environments. The chapter discusses key compression techniques, examining their principles, strengths, and limitations within the context of these environments.

Chapter 3 details the development of a numerical-based mathematical method for evaluating model compression techniques. The chapter addresses **RO1**, which investigates the feasibility of creating a standardized, quantitative approach to objectively assess and rank compression methods based on their impact on various model characteristics. The chapter discusses the proposed method, the experimental setup, and the results.

Chapter 4 focuses on **RO2**, examining the practical limits of achieving extreme model compression for embedded devices with extremely limited memory resources. The chapter evaluates the feasibility of achieving a deep compression ratio of approximately 32x while maintaining acceptable accuracy levels. The chapter presents the methods used and the results of the evaluation.

Chapter 5 addresses the challenge of optimizing DL models that are inherently sensitive to precision changes due to their architecture. It seeks to push the limits of what is achievable in model compression and optimization, ensuring that even the most sensitive models can be efficiently deployed. The chapter addresses **RO3**.

Chapter 6 concludes the thesis by summarizing the key findings, contributions, and recommendations for future research directions.

2 PRELIMINARIES

This chapter provides a concise foundation for understanding the core concepts and techniques underlying efficient optimization of the DL model for resource-constrained devices.

Key compression methods such as quantization, pruning, tensor decomposition, knowledge distillation, and lightweight architecture design are explored in detail. Their principles, strengths, and limitations are analyzed to provide insight into their practical applicability and effectiveness in addressing the challenges.

2.1 Model Compression Methods

Compression methods for DL are designed to minimize the size of the model and computational complexity while preserving acceptable levels of accuracy. Each technique employs a different approach to achieve this objective. Primary methods include quantization, knowledge distillation (KD), pruning, low-rank tensor decomposition (TD), compact model design, and the integration of multiple compression strategies through ensemble approaches.

Quantization is one of the most widely used techniques in model compression, particularly in resource-constrained environments where memory and computational resources are limited. The core idea behind quantization is to reduce the precision of the numerical values used to represent a model's parameters and activations, thereby decreasing the overall model size and improving computational efficiency.

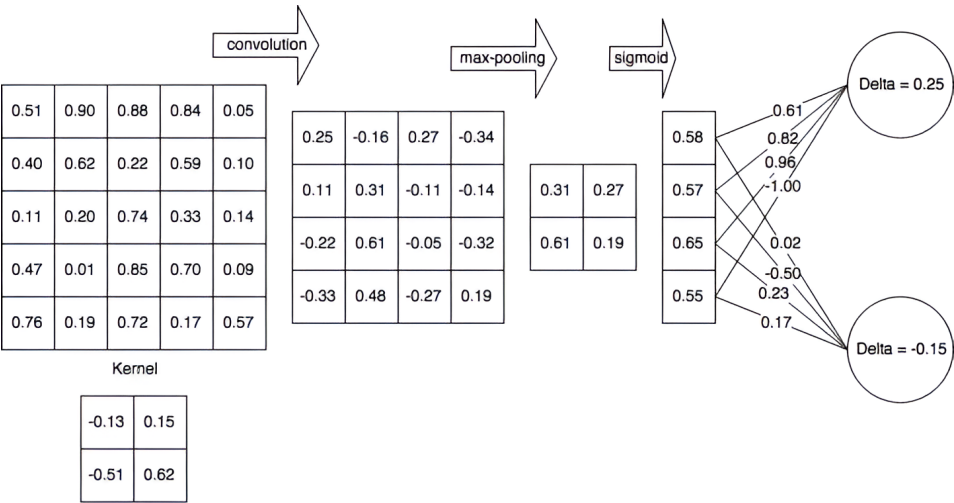


Figure 1: A typical CNN architecture with a normalized (5×5) input image convolved with a normalized filter (2×2 kernel) showing convolution, pooling, and weights matrix multiplication operations in a deep learning network. This low-level abstraction shows the internal computation performed on the network parameters (i.e., the input tensor (5×5) image). weights, and activations. Appendix 1, Figure 4.

2.1.1 Quantization

In DL models, parameters (weights, biases, and activations) are typically represented using full-precision 32-bit floating point numbers (FP32) as shown in Figure 1. While the full precision allows for accurate representation of values, it also results in models that are memory intensive and computationally demanding, particularly when targeting IOT de-

vices or embedded hardware. Quantization addresses this challenge by converting the full-precision 32-bit floating-point values into lower-bit representations (8-bit, 4-bit, or 2-bit).

Quantization techniques can be categorized into various types, each offering different trade-offs between precision, model size, and computational efficiency. Symmetric quantization maps floating point values into uniformly spaced intervals, making it computationally efficient and straightforward to implement. In contrast, non-uniform quantization adjusts interval sizes based on data distribution, allocating more precision to areas with higher variance, thereby reducing quantization errors at the expense of greater complexity.

Post-training quantization (PTQ) [37] applies quantization to pre-trained models without additional training. Calibration on a small dataset helps fine-tune the quantization parameters, making PTQ a quick and resource-efficient solution. On the other hand, quantization-aware training (QAT) [12] incorporates quantization effects directly into the training process, allowing the model to adapt to low bit precision. QAT often achieves better accuracy than PTQ, especially for very low-bit representations.

Integer-only quantization [12] ensures that all operations use integer arithmetic, making it an optimal choice for hardware lacking floating-point support. This approach enables highly efficient inference, particularly on resource-constrained platforms. Each type of quantization offers unique advantages and challenges, catering to specific application needs and hardware constraints.

The quantization process involves several key steps. First, the range of values for weights and activations is determined, typically by calculating the minimum and maximum values. This range is then divided into discrete levels based on the target bit-width, with a scale factor and zero-point calculated to map the floating-point values to integer representations. Using these parameters, the weights and activations are quantized into integers, effectively reducing their precision.

For inference, the quantized values are dequantized back to floating-point equivalents using the same scale and zero-point, enabling compatibility with downstream computations. The model then performs inference with these quantized values, resulting in significantly reduced memory requirements and faster computation, especially on hardware optimized for integer arithmetic. This streamlined process ensures efficient deployment while maintaining acceptable performance.

A 32-bit float range (i.e., weights, biases, activations) maps to an n-bit quantized range. Considering an 8-bit quantizer, the mapping function maps the input float tensor range to the 8-bit quantized output. The function is defined in Eq. (1):

$$q_{8bit} = \text{round}(m_f i_f) \quad (1)$$

where q_{8bit} is the 8-bit quantizer, m_f is the multiplier (scale factor), and i_f is the input float tensor. The multiplier is the quantization constant that is multiplied by the float input tensor, as expressed in Eq. (2):

$$m_f = \frac{\frac{2^7-1}{2}}{\max(|i_f|)} \quad (2)$$

Quantization has been widely adopted in various applications, particularly for deploying state-of-the-art models on resource-constrained edge devices, such as using object detection models for real-time surveillance on drones, enabling natural language processing on mobile assistants, and powering image classification in wearable health monitoring devices.

For example, MobileNet [57], a family of lightweight convolutional neural networks (CNNs), is based on quantization to reduce computational demands, making it suitable for mobile devices. Similarly, BERT [64], a transformer-based language model for NLP tasks, benefits from quantization to enable efficient deployment on mobile platforms without significant loss of accuracy. These examples highlight the critical role of quantization in bridging the gap between high-performance models and the practical constraints of edge computing.

2.1.1.1 Binarization: An Extreme Case of Quantization

Binarization is a bit reduction technique that is considered an extreme case of quantization in which the weights and/or activations are encoded using a single bit (i.e., 1 bit) [39, 40]. A single bit can be considered the atomic bit level of a number system; therefore, a significant decrease in the model accuracy is imminent due to loss of information during the binarization process.

In the binarization process, updating the weights during backward pass using the standard gradient descent approach is impossible because computing the loss gradient would result in zero in almost all conditions. A Straight-Through-Estimator (STE) pseudo-function that has been proven to solve this limitation.

The binarization function (a non-zero sign function) b_{1bit} takes the float tensor as input and returns a binary output ($-1, +1$), as shown in Eq. (3):

$$o_{1bit} = b_{1bit}(i_f), o_{1bit} \in \{-1, 1\}, i_f \in \{R\} \quad (3)$$

where b_{1bit} is the binarization function, o_{1bit} is the binary output generated, and i_f is the input float tensor. During the backward pass, the loss gradient is calculated using the STE function, which takes the output tensors as input and returns a binary output, which is constrained to the threshold value, as expressed in (Eq. (4)):

$$loss_{gradient} = \begin{cases} 1 & \text{abs}(i_f) \leq threshold_{value} \\ 0 & \text{abs}(i_f) > threshold_{value} \end{cases} \quad (4)$$

where the $threshold_{value}$ is the float value that controls the $loss_{gradient}$ and i_f is the float tensor processed by the STE pseudo gradient function.

2.1.1.2 Challenges and Considerations

As established in the surveyed literature (Section 1.2), bit reduction in DL models offers significant benefits in terms of reducing model size and improving computational efficiency. However, its implementation poses several challenges and considerations that must be addressed to maintain the integrity and performance of the model.

One critical challenge is managing accuracy trade-offs. Reducing the precision of weights and activations can lead to performance degradation, particularly in DL models where high precision significantly affects output quality. The balance of model size, computational speed, and accuracy is essential. Another key consideration is bit-width selection, which requires careful experimentation to determine the optimal level of quantization for different layers. Incorrect bit-width choices can result in excessive performance losses.

Hardware compatibility further complicates quantization. The effectiveness of low-precision arithmetic depends heavily on the target hardware's ability to execute such operations efficiently. Aligning quantization strategies with hardware capabilities is crucial

to achieving performance gains. Additionally, model sensitivity can be an issue, as quantization may make models more prone to performance instability, especially in critical applications. Rigorous validation of quantized models is necessary to ensure they meet accuracy and performance standards.

Although quantization has advanced significantly, several research areas remain open. Adaptive quantization methods that dynamically adjust parameters during inference could improve accuracy without sacrificing efficiency. Using higher bit precision in critical layers while aggressively quantizing others is another promising avenue. Furthermore, as new neural network architectures emerge, tailored quantization techniques are needed to ensure that these architectures can fully benefit from the advantages of quantization. Addressing these challenges and exploring these research directions will further enhance the applicability and effectiveness of quantization in deep learning models.

2.1.1.3 8-bit Quantization Impact on CNN Models

In CNNs, most of the computations and memory usage are attributed to the manipulation of large matrices, such as weight matrices and activation tensors. Reducing the precision of these matrices from 32-bit floats to 8-bit integers not only reduces the memory required to store the parameters, but also accelerates the computation by using integer arithmetic instead of floating-point operations.

Quantization is applicable to various layers of CNN —convolutional layers, fully connected layers, and activation functions. However, it is common practice to exclude the input and output layers from quantization to avoid a significant loss in accuracy, as these layers are highly sensitive to precision reduction. The quantization process is mathematically defined in Eq. (1) and Eq. (2). The transformation allows weights, activations, and other network parameters to be represented as 8-bit integers, reducing the overall memory footprint and computational complexity.

8-bit quantization offers significant advantages for optimizing DL models, particularly in resource-constrained environments. One of the primary benefits is the reduction in model size, achieved by representing parameters in an 8-bit integer format instead of 32-bit floats. This transformation reduces the size of the model by approximately 4x [13], which is particularly valuable for applications running on mobile or embedded systems with limited memory resources.

Another key advantage of quantization is faster computation. Integer arithmetic is generally faster than floating-point arithmetic, especially on hardware optimized for low-precision operations. This leads to reduced inference times, making quantized models well-suited for real-time applications where low latency is critical.

Energy efficiency is also a notable benefit of quantization. Lower-precision operations consume less power, which is essential for devices with constrained power budgets, such as edge devices and Internet of Things (IoT) platforms. By combining these advantages, quantization enables the deployment of efficient and effective deep learning models in environments with stringent computational, memory, and power limitations.

However, quantization comes with trade-offs that need careful consideration to ensure its effectiveness. One major challenge is accuracy loss. Reducing the precision of parameters can degrade a model's accuracy, with the impact being particularly pronounced in sensitive layers, such as input and output layers. These layers are critical to preserving feature integrity and overall model performance, and aggressive quantization can lead to significant reductions in their effectiveness. To mitigate this, quantization is often applied selectively, targeting less sensitive layers while leaving critical layers at higher precision.

Another challenge relates to layer sensitivity. Certain network architectures and spe-

cific layers are more vulnerable to lower-precision representations. Layers that capture fine-grained details or make high-level decisions are particularly susceptible to performance degradation under aggressive quantization. This sensitivity requires a nuanced approach, in which compression techniques are applied judiciously to ensure a balance between efficiency and performance. By understanding and addressing these sensitivities, it is possible to achieve effective model compression without compromising essential functionality.

2.1.2 Knowledge Distillation

KD is a model compression technique that involves transferring the learned information from a large model (teacher) to a small, compact model (student). The objective is for the student model to learn the expressive capacity of the teacher model, thus ensuring the preservation of performance.

KD uses the output of the teacher model (usually logits or soft targets) to train the student model as described in Figure 2. This allows the student to learn the approximation function represented by the teacher, which is richer and potentially more informative than learning directly from the hard labels.

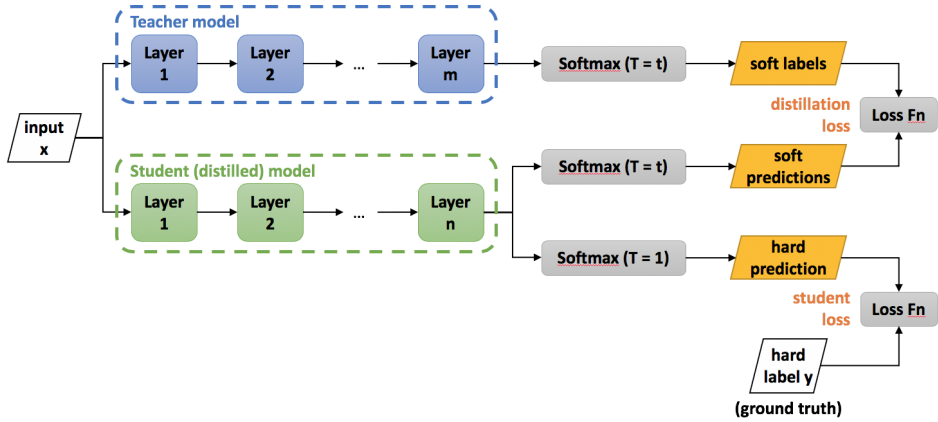


Figure 2: The teacher-student model based on a temperature-based softmax function. [42].

Knowledge distillation encompasses several variants designed to cater to diverse scenarios and training setups. In self-distillation [44], the same model serves as both the teacher and the student in different phases of training. This approach allows the model to refine its own predictions iteratively, enhancing its performance without requiring an external teacher model.

On-line distillation, on the other hand, involves multiple models being trained simultaneously. During this process, the models share knowledge with each other [43], learning collaboratively to improve their overall performance. This variant is particularly effective in distributed training environments where multiple models can leverage shared insights dynamically.

Mutual distillation takes a slightly different approach by allowing two models to distill knowledge together. This bidirectional exchange helps improve the accuracy and generalization capabilities of both models, making it a powerful strategy for scenarios where model collaboration can yield significant performance gains. These variants highlight the versatility of knowledge distillation in addressing various training requirements and re-

source constraints.

Knowledge distillation (KD) involves a series of well-defined steps, each contributing to the effective transfer of knowledge from the teacher model to the student model.

First, the teacher model, a large and highly accurate neural network, is trained on a given dataset to achieve superior performance. This model serves as the reference point for distillation. Next, the teacher model generates soft labels from its output logits. These soft labels, representing the probability distribution over classes, encapsulate richer information about the data compared to standard hard labels. This additional information helps the student model to understand the nuances of the dataset.

The student model is then trained using a combination of the soft labels and the original hard labels. This dual training approach enables the student to closely approximate the teacher's function while retaining essential features of the original dataset. To enhance the distillation process, temperature scaling is applied during teacher output generation. A temperature parameter is introduced in the softmax activation function to soften the probability distribution, providing the student model with more informative gradients.

These steps collectively ensure that the student model effectively learns from the teacher, achieving a balance between compactness and performance.

The teacher-student model uses a temperature-based softmax function at the output layer as shown in (Eq. (5)):

$$\begin{aligned} Loss_{total} = \alpha * H(y, \sigma(z_s), T = 1) + \beta * \\ H(\sigma(z_t; T = t), \sigma(z_s, T = t)) \end{aligned} \quad (5)$$

$Loss_{total}$ is the total loss, which is the combination of student and distillation losses. The loss of the student is calculated using the standard loss function by making the temperature parameter ($T = 1$). The temperature parameter controls the amount of information that can be distilled for the student. However, we need to remember that the student has a threshold that limits the amount of information that he can retain from the teacher. The α and β are constants associated with the individual loss function taking the respective unnormalized log probabilities (z_s, z_t) for each class label.

2.1.2.1 Challenges and Considerations

Knowledge distillation (KD) provides a practical approach for the deployment of deep learning models on resource-constrained devices, offering a balance between performance and efficiency. As research advances, it is expected that improvements in KD techniques will address existing limitations, expanding their applicability across a broader range of tasks and domains.

Despite its advantages, KD presents several challenges as established in the surveyed literature (Section 1.2) that require careful consideration. One critical factor is its dependency on the quality of the teacher model. The student's performance is inherently limited by the teacher's capabilities, making the selection of an effective teacher model a crucial step in the process. Additionally, the training complexity of KD arises from the need to carefully tune parameters such as the temperature and loss functions. The temperature parameter, which softens the output probabilities to facilitate knowledge transfer, must be optimized to achieve effective distillation, which adds to the intricacy of the process.

While the student model is designed to be resource efficient, the training phase can be resource intensive because it involves running both the teacher and student models simultaneously. This dual execution doubles computational overhead, posing challenges in scenarios with limited training resources. Moreover, balancing model compression and

performance retention is a delicate task. Excessive compression may degrade the student model's capabilities, while insufficient compression may fail to deliver the desired efficiency gains.

Another challenge is ensuring the student model's generalization to unseen data. Overfitting to the teacher's specific characteristics can hinder the student's ability to perform well on new tasks. Furthermore, KD must be adaptable to different tasks and domains. This adaptability often requires modifying the distillation process to account for varying data characteristics, label granularity, or operational constraints.

These challenges highlight the need for careful design and optimization in KD application, to ensure its effectiveness while maintaining a balance between efficiency and performance in diverse applications.

These challenges strengthen the need for ongoing research, with a focus on creating more adaptable, efficient, and compact models that can leverage the full capabilities of teacher networks.

2.1.3 Low-Rank Tensor Decomposition

A tensor is a multidimensional array that generalizes matrices and vectors to higher dimensions. While a matrix is a two-dimensional array, a tensor is a higher-dimensional array (3 or more). For DNNs, tensors are used to represent weights, activations, and input data. Compressing these tensors can lead to significant reductions in memory and computational requirements.

TD is a technique that divides a tensor into smaller components, which can be represented in compressed format. Among various TD techniques, *Tensor Train Decomposition* (TTD) is particularly effective for NN compression. TTD represents the original high-dimensional tensor as a sequence (i.e., tensor train) of smaller low-rank tensors, called *cores*, which are linked in a chain-like structure.

Low-rank TD is another model compression technique that reduces the parameter count and computational complexity of NNs by approximating the weight tensors with lower-rank structures. This method is particularly effective for CNNs and fully connected (FC) layers, where the weights can be represented as high-dimensional tensors.

TD decomposes a high-dimensional tensor into its low-rank components, leveraging the fact that weight tensors in neural networks often contain redundant information that can be compactly represented without significantly impacting performance. It capitalizes on the linear dependencies within the tensor to achieve compression.

Tensor decomposition (TD) plays a pivotal role in deep learning by enabling efficient representation and computation. The most widely used TD techniques include CP decomposition, Tucker decomposition, and Tensor Train decomposition, each offering unique advantages for model compression and optimization.

CP decomposition (CANDECOMP / PARAFAC) [45] decomposes a tensor \mathcal{X} into a sum of rank-one tensors. The mathematical representation is as follows:

$$\mathcal{X} \approx \sum_{r=1}^R \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r \quad (6)$$

where \mathbf{a}_r , \mathbf{b}_r , and \mathbf{c}_r are the factor vectors in each mode, and R is the rank of the decomposition.

Tucker Decomposition [46] decomposes a tensor \mathcal{X} into a core tensor \mathcal{G} multiplied by a matrix along each mode. The equation is as follows:

$$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \quad (7)$$

where \mathbf{A} , \mathbf{B} , and \mathbf{C} are the factor matrices corresponding to each mode, and \times_n denotes the n -mode product of the tensor with a matrix.

Tensor train decomposition [47] represents a tensor \mathcal{X} in the form of a sequence of three-dimensional tensors, expressed as:

$$\mathcal{X}(i_1, i_2, \dots, i_N) \approx \mathbf{G}_1[i_1] \cdot \mathbf{G}_2[i_2] \cdots \mathbf{G}_N[i_N] \quad (8)$$

where $\mathbf{G}_k[i_k]$ are the k -th core tensors of the Tensor Train, and i_k are the indices corresponding to each dimension of the tensor.

Tensor decomposition (TD) in DL involves several key steps, ensuring that the model benefits from reduced computational and memory requirements while maintaining accuracy.

The first step is to select the appropriate decomposition type based on the model architecture and specific requirements. Options include CP decomposition for simplicity, Tucker decomposition for flexibility, or Tensor Train decomposition for high-dimensional tensors. Each method offers unique advantages, depending on the context.

Once the decomposition type is chosen, the next step involves the decomposition of the weight tensors of the neural network. This process transforms high-dimensional tensors into lower-dimensional structures, significantly reducing the model's complexity and memory footprint. The decomposition is typically applied to layers with the largest parameter counts, such as fully connected or convolutional layers.

Finally, during inference, the decomposed tensors are efficiently multiplied according to the chosen decomposition rules to reconstruct the approximate weight tensors. This step ensures that the model operates seamlessly with the compressed representation, maintaining performance while benefiting from the reduced computational load.

2.1.3.1 Tensor Train Decomposition

Consider a weight matrix $W \in \mathbb{R}^{I \times J}$, where I and J are the dimensions of the matrix. To apply TTD, the weight matrix is first reshaped into a high-dimensional tensor \mathcal{W} . The tensor is then factorized into a sequence of smaller tensors G_1, G_2, \dots, G_N , where each G_n is a core tensor of the decomposition. These cores are connected through shared indices, forming a TTR. Formally, the TTP of a tensor \mathcal{W} can be written as:

$$\mathcal{W}(i_1, i_2, \dots, i_N) = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{N-1}=1}^{R_{N-1}} G_1(i_1, r_1) G_2(r_1, i_2, r_2) \cdots G_N(r_{N-1}, i_N) \quad (9)$$

Here, i_1, i_2, \dots, i_N are the indices of the tensor elements, and R_1, R_2, \dots, R_{N-1} are the TT-ranks of the decomposition, which control the level of compression. A lower TT rank leads to a more compressed tensor but could degrade the performance of the model, while a higher TT rank maintains precision but offers less compression.

In the context of DL models, particularly CNNs, TTD is applied to compress the weight matrices of the FC and convolutional layers. These layers are often memory-intensive and computation-intensive, and compressing them will lead to significant reductions in model size and inference time.

The dense layers in CNNs typically account for the majority of the model's parameters, making them ideal candidates to target. By applying TTD to these layers, the weight matrices are transformed into their TT representations, reducing the memory required to store the weights and accelerating computations during both training and inference.

Tensor Train Decomposition (TTD) provides significant advantages for compressing convolutional neural networks (CNNs). By decomposing large-weight matrices into smaller tensor cores, TTD greatly reduces the memory footprint, making it ideal for deploying models on resource-constrained devices such as mobile phones and embedded systems.

The compact tensor representation also improves computational efficiency by minimizing the number of operations required during forward and backward passes. Additionally, TTD's scalability, which grows linearly with the size of the input tensor, makes it well-suited for large networks with high-dimensional weight matrices.

However, the compression achieved through TTD is governed by the Tensor Train rank (TT rank). Selecting an appropriate TT-rank is critical for balancing compression efficiency and maintaining model performance, as overly aggressive compression can lead to a loss of accuracy.

2.1.3.2 Challenges and Considerations

Low-rank tensor decomposition (TD) is a powerful method for compressing deep neural networks, particularly in scenarios where reducing the size of the model and improving computational efficiency are critical. This technique has significant potential to advance the deployment of deep learning models in resource-constrained environments. As research continues, it is expected that low-rank TD methods will become more robust and adaptable, extending their applicability across diverse architectures and use cases.

Despite its advantages, the implementation of low-rank TD comes with several challenges, as established in the surveyed literature (Section 1.2). A primary consideration is the selection of rank, which directly affects the balance between compression and performance retention. Determining the optimal rank is a complex task that often requires extensive empirical testing and validation. The choice must align with the application's specific requirements and the data's complexity to ensure that the model retains its predictive power.

Managing the trade-offs between accuracy and compression is another critical factor. Excessive rank reduction can result in the loss of vital information, leading to significant performance degradation. Careful evaluation is needed to determine the extent of compression that the model can sustain while maintaining acceptable accuracy levels. Additionally, while the goal of low-rank TD is to reduce computational demands during inference, initial decomposition and reconstruction processes can introduce notable computational overhead, which must be managed effectively.

Hardware compatibility also plays a pivotal role. Decomposed models must align with the capabilities of the target hardware, particularly specialized AI accelerators such as TPUs and FPGAs. Ensuring that the optimized models can be efficiently deployed on the intended devices is crucial for leveraging the full benefits of low-rank TD.

These challenges emphasize the need for a strategic approach to implementing low-rank TD, balancing the goals of compression and efficiency with the preservation of essential model performance.

2.1.4 Pruning

Pruning is a popular model compression technique that systematically removes less important neurons or connections (weights) to reduce a network's complexity and size. This process not only decreases the number of parameters and computational demands, but also enhances model generalization by mitigating overfitting.

Pruning operates on the premise that not all weights in a neural network contribute

equally to its performance. By eliminating redundant or low-impact weights, the model becomes more efficient without significantly compromising accuracy.

Deep learning pruning techniques are designed to reduce the size and computational complexity of neural networks by selectively removing certain components. Each approach has its own strategy and implications for network performance.

Weight pruning focuses on removing individual weights from the weight matrix. The weights are ranked on the basis of their magnitudes, as shown in the equation.

$$rank_{weight} = |w_i|, w_i \in W \quad (10)$$

Weights with the smallest magnitudes are removed under the assumption that they contribute less to the activation of subsequent layers. This approach leads to a sparser network while retaining its core functionality.

Node pruning takes a broader perspective by removing entire neurons or filters from the network. This involves eliminating all incoming and outgoing connections associated with those nodes. Node pruning is particularly effective in convolutional layers, where it results in significant computational reductions and a more streamlined architecture.

Structured pruning adopts a systematic method by targeting larger elements, such as entire channels, layers, or other structures within the network. This approach is advantageous for hardware acceleration, as it simplifies memory access patterns and reduces fragmentation, making it suitable for deployment on modern hardware platforms.

Each of these pruning strategies offers unique benefits and trade-offs, requiring careful consideration based on the specific application and network architecture.

The process of implementing pruning typically involves three key steps. First, the neural network is trained to convergence, ensuring that it achieves high performance on the training data set. This initial step provides a robust baseline model with which to work.

Next, a pruning criterion is applied to determine which weights, neurons, or other components to remove. Common criteria include the magnitude of weights, their contribution to output variation, or specific heuristics tailored to the network architecture. This step effectively reduces the network size and complexity by identifying and eliminating less significant elements.

Finally, the pruned network undergoes fine-tuning, a critical phase in which the model is retrained on the original dataset to recover any accuracy lost during the pruning process. This step ensures that the network retains its predictive performance while benefiting from the reduced computational and memory demands introduced by pruning.

2.1.4.1 Challenges and Considerations

Pruning, a widely used method for reducing the computational and memory demands of neural network models, presents several challenges that require thoughtful consideration for successful implementation as established in the surveyed literature (Section 1.2).

One of the primary challenges is selecting the appropriate pruning technique, as options such as weight pruning, unit pruning, and structured pruning offer different trade-offs in terms of model performance and computational efficiency. The choice of method must align with the specific requirements of the application and the network's architecture to maximize benefits without compromising critical performance aspects.

The impact of pruning on model performance is another significant consideration. Although pruning can reduce model size, excessive pruning can lead to a loss of accuracy and generalization capabilities, particularly if essential connections are removed. Striking the right balance between compression and performance retention is crucial for maintaining the model's predictive power.

Pruning is often an iterative process that involves cycles of pruning followed by fine-tuning to restore or stabilize performance. This iterative approach can be computationally intensive and requires careful monitoring to prevent overfitting and to ensure the model remains stable during successive adjustments.

Hardware compatibility further complicates pruning implementation. Structured pruning, for example, is often more advantageous for deployment on GPUs and specialized AI accelerators due to its ability to create regular memory access patterns and better hardware utilization. However, unstructured pruning can lead to sparse matrices that are less efficiently handled by many hardware platforms.

In dynamic environments, where input data characteristics may change over time, static pruning strategies may not be ideal. Adaptive pruning approaches, capable of modifying the network architecture in response to evolving data distributions or task requirements, may be necessary to maintain optimal performance.

These considerations highlight the complexity of pruning neural networks, underscoring the need for a strategic, context-aware approach to fully leverage its benefits while mitigating potential drawbacks.

2.1.5 Microarchitecture

Microarchitecture, or lightweight architecture, in NN refers to the design of compact network structures with a focus on minimizing storage and memory footprints. Unlike other compression techniques such as pruning, quantization, KD, or low-rank TD, microarchitecture involves building efficient models from the ground up, rather than modifying existing ones.

Microarchitecture directly influences the initial model design, with the aim of efficiency and compactness without compromising performance. Examples include models like MobileNet and EfficientNet, which are designed inherently to be lightweight and suitable for deployment on resource-constrained devices.

The design of efficient microarchitectures typically adheres to a set of foundational principles aimed at balancing performance and resource efficiency. A critical aspect is layer optimization, where each network layer is crafted to minimize computational cost and parameter count. Techniques such as depth-wise separable convolutions [57] and point-wise convolutions [57] exemplify this principle by breaking down complex operations into simpler, less resource-intensive components, significantly improving efficiency in convolutional neural networks (CNNs).

Another key principle is the use of neural architecture search (NAS) [63]. This approach employs machine learning techniques to automate the discovery of optimal network architectures. By exploring various configurations within defined resource and performance constraints, NAS enables the creation of architectures that are finely tuned for specific deployment scenarios, maximizing efficiency and effectiveness. These principles collectively drive the development of microarchitectures that are both high-performing and well-suited for resource-constrained environments.

Several prominent models demonstrate the effective application of microarchitecture principles to achieve efficiency and performance. MobileNets leverage depthwise separable convolutions to create lightweight architectures that are highly suitable for deployment on mobile and edge devices with constrained computational resources. SqueezeNet employs a unique design strategy centered on 1x1 convolutions, significantly reducing the parameter count without sacrificing accuracy, making it an ideal choice for memory-constrained applications.

EfficientNet [59] introduces a compound scaling method that uniformly scales the

depth, width, and resolution of the network. This approach allows the model to achieve state-of-the-art performance while maintaining computational and parameter efficiency. These models exemplify how thoughtful architectural design can address the challenges of deploying deep learning in resource-constrained environments.

2.1.5.1 Challenges and Considerations

The microarchitecture approach provides substantial advantages in designing efficient neural network models. However, several challenges and considerations must be addressed to optimize its application as established in the surveyed literature (Section 1.2). Developing microarchitectures is a complex process that requires sophisticated design strategies. It often involves iterative trial-and-error procedures, which can be resource-intensive. Employing a Neural Architecture Search (NAS), although promising, adds another layer of complexity. The computational demands of numerous iterations needed to identify optimal architectures make this approach time-consuming and costly.

Scaling microarchitectures to handle more complex tasks or larger datasets presents another significant challenge. Maintaining efficiency gains while ensuring the scalability of the model can be difficult. This process requires careful adjustments to prevent the trade-offs between performance and resource optimization from diminishing returns. Furthermore, creating optimized microarchitectures often leads to extended development and testing cycles. Each iteration requires extensive simulations, refinements, and validations, which can delay the deployment of the model in practical applications.

Balancing computational efficiency with model accuracy is a persistent challenge in microarchitecture design. Although efficiency is vital in resource-constrained environments, overly aggressive optimizations can impair the model's ability to handle complex tasks effectively. Achieving this balance requires meticulous adjustments to ensure that the model remains both efficient and capable.

These challenges emphasize the importance of employing flexible design strategies and frameworks. Such approaches enable iterative refinement and optimization, ensuring that microarchitectures deliver balanced performance, scalability, and timely deployment across various applications.

2.1.6 Ensemble of Compression Methods

An ensemble of compression methods is a sophisticated strategy that combines multiple techniques, such as pruning, quantization, low-rank tensor decomposition, and knowledge distillation, to maximize model efficiency and performance. Using the unique strengths of each approach, this method achieves superior results compared to using any single compression technique in isolation [50].

The rationale for using an ensemble of compression methods is to achieve deep compression of NNs, targeting substantial reductions in model size and computational complexity while preserving performance. Each technique addresses specific inefficiencies, and their combined effect enables an unprecedented level of compression that surpasses the capabilities of any single method.

Pruning focuses on reducing redundancy within the NN. By eliminating unnecessary or less important neurons and connections, pruning helps reduce model size and computational overhead, which enhances the network's operational efficiency during inference.

Quantization aims to reduce the precision of the weights and activations from floating-point to lower bit representations, such as 8-bit, 4-bit, 2-bit, or 1-bit. This reduction significantly decreases the model storage and memory footprints. Hence, speed up the computation, particularly on hardware that supports integer operations.

Low-rank tensor decomposition targets the computational complexity of the network. By approximating the weight matrices with lower-rank versions that maintain most of the original matrix's significant information, this method can reduce the number of parameters and the computational cost of matrix multiplications involved in neural computations.

Knowledge distillation works by transferring the knowledge from a large model (teacher) to a small, compact model (using fewer layers). KD aims to mimic the performance of the large network by ensuring that the compact model achieves similar accuracy.

By integrating these methods, this approach optimally enhances model efficiency, making it highly adaptable for deployment on extremely resource-constrained hardware. This strategy ensures that the model meets stringent requirements for memory, computation, and power consumption.

Implementing ensemble of compression techniques involves a strategic and iterative approach to achieve optimal efficiency and performance. The process begins with identifying compatible techniques, as not all methods work harmoniously across every network architecture. Careful consideration is given to the specific requirements of the target application and the characteristics of the network to select a complementary set of techniques.

The selected methods are then applied sequentially to maximize their individual and collective impact. Typically, the process starts with pruning to eliminate redundancies, followed by quantization to reduce precision and memory requirements. Tensor decomposition may be employed to simplify complex weight structures, and finally, knowledge distillation is used to fine-tune and optimize the model's overall performance.

After each compression step, iterative optimization is performed to evaluate and fine-tune the model. This ensures that compression does not negatively impact accuracy or generalizability. Each phase is followed by rigorous testing to maintain the model's reliability and robustness for its intended application.

2.1.6.1 Challenges and Considerations

Ensemble methods for DL model compression, while highly effective, introduce significant complexity and unique challenges as established in the surveyed literature (Section 1.2).

Integrating multiple compression techniques requires strategic management to handle dependencies and interactions. Each method, such as pruning, quantization, or distillation, can introduce changes that affect subsequent layers, adding complexity to the optimization pipeline. Ensuring that these methods complement each other without conflicts is crucial.

A major concern with ensemble methods is the cumulative error introduced by each technique. Although individual methods may only introduce minor inaccuracies, they can add up across the layers of compression, potentially leading to a noticeable degradation in model performance. Addressing this requires a delicate balance between achieving high compression rates and maintaining model accuracy.

The ensemble approach also requires extensive hyperparameter tuning to optimize the interactions between different compression methods. This tuning process can be resource intensive, often requiring significant computational power and time to achieve optimal results. Moreover, maintaining and scaling models compressed with multiple techniques can be challenging, particularly when deploying them on different hardware platforms or adapting them to new datasets. These challenges highlight the need for robust frameworks and methodologies to maximize the benefits of ensemble compression while minimizing its inherent complexities.

The ensemble of compression methods represents a sophisticated strategy to achieve a very deep compressed neural network models suitable for deployment on resource-constrained devices. This approach requires a deep understanding of each compression method and careful management of their integration and tuning to ensure optimal performance and efficiency.

3 COMPRESSION METHODS EVALUATION USING NUMERICAL-BASED MATHEMATICAL APPROACH

This chapter addresses RQ1: "Can a numerically based mathematical method be developed to quantitatively evaluate state-of-the-art deep learning (DL) model compression techniques for diverse application requirements?" by presenting and validating a novel methodology for objectively evaluating and ranking compression methods. This approach quantifies the impact of various techniques, allowing informed comparisons and selections tailored to specific needs.

Publication I: (Section 3.1)

Ademola, O.A.; Leier, M.; Petlenkov, E. Evaluation of Deep Neural Network Compression Methods for Edge Devices Using Weighted Score-Based Ranking Scheme. *Sensors* 2021, 21, 7529. <https://doi.org/10.3390/s21227529>

3.1 Evaluation of Deep Neural Network Compression Methods for Edge Devices Using Weighted Score-Based Ranking Scheme

To develop a numerical approach for evaluating and ranking DL model compression methods, it is essential first to establish a conceptual framework that outlines the key metrics and criteria involved in the evaluation process. This framework will serve as the foundation for the subsequent mathematical formulation and implementation. In addition, understanding the underlying mechanisms of each compression method is crucial for a comprehensive evaluation.

3.2 Evaluation Metrics

DL model compression methods are assessed through metrics that capture their impact on both performance and efficiency. These key metrics - compression ratio, accuracy, inference time, maximum memory footprint, and computational cost - offer a holistic view of the strengths and limitations of different approaches.

The compression ratio provides a measure of how much the model size is reduced after compression compared to the original. This metric is a direct indicator of the storage efficiency achieved. Meanwhile, accuracy evaluates the extent to which the compressed model preserves the predictive performance of its uncompressed counterpart. Maintaining accuracy is essential to ensure that compression does not compromise the model's utility, especially in critical applications.

Inference time, which measures the time required for the model to make predictions, is another vital metric. Reducing the inference time enhances the speed and responsiveness of the model, making it suitable for real-time and interactive applications. Closely related to this is the computational cost, which is reflecting the processing resources needed to execute the model. Lower computational costs result in faster processing and reduced energy consumption, which is critical for devices with limited power or processing capabilities.

The peak memory footprint measures the maximum memory required during model execution. This metric is especially important in resource-constrained environments, where memory is limited. A reduced memory footprint ensures that the model can be efficiently deployed on hardware with stringent memory restrictions.

Each of these metrics plays a crucial role in understanding the benefits and drawbacks of different model compression techniques, providing insight into their practical applica-

tions.

Together, these metrics form a comprehensive framework for evaluating model compression methods, helping researchers and developers identify suitable techniques for specific use cases while balancing performance, efficiency, and resource constraints.

3.3 Limitations of Current Metrics

According to the literature surveyed (Section 1.2), while the aforementioned metrics are invaluable for evaluating model compression methods, they exhibit certain limitations that hinder a comprehensive assessment of compression techniques. These shortcomings emphasize the need for more nuanced and integrative evaluation methods.

One significant limitation is the lack of comparative analysis. Current metrics do not facilitate direct comparisons between different types of compression methods. For example, a method that excels in improving inference speed might sacrifice accuracy [37], making it challenging to evaluate overall performance and rank the methods holistically.

Another issue is the absence of a holistic perspective. Many metrics focus on isolated aspects of compression, such as compression ratio or accuracy, without considering their interdependencies. For example, achieving a high compression ratio at the cost of a drastic drop in accuracy might render a compression method ineffective, yet existing metrics often fail to adequately balance these trade-offs.

Application-specific variability further complicates evaluations. Metrics typically do not account for the diverse requirements of different application domains. A compression method optimized for image classification may not perform as well for natural language processing or time series analysis. This variability requires metrics that can adapt to the unique constraints and objectives of each domain.

Lastly, user and environmental factors are often overlooked. An effective evaluation should consider user expectations for responsiveness and the computational resources available in deployment environments. For example, mobile users prioritize fast responses, while IoT devices may operate under stringent memory and energy constraints. Current metrics do not sufficiently reflect these real-world factors, limiting their practical utility.

These limitations underscore the importance of developing a unified evaluation methodology that integrates multiple metrics into a cohesive numerical-based mathematical approach. Such a methodology would enable more comprehensive and objective assessments, facilitate meaningful comparisons across methods, and better account for the diverse conditions and requirements of deployment environments.

3.4 The Numerical-Based Mathematical Approach

3.4.1 Mathematical Formulation

This subsection develops a mathematical method that integrates the key parameters essential for evaluating model compression methods. These parameters —Compression Ratio (CR), Accuracy (Acc), Peak Memory Footprint (PMF), Computational Cost (FLOP) and Inference Time (IT) are chosen for their critical impact on the performance and efficiency of compressed models in practical applications.

3.4.1.1 Definition of Parameters

The definition of key parameters is critical for evaluating and ranking model compression methods, particularly in resource-constrained environments. These parameters capture the trade-offs between model efficiency and performance, ensuring that compressed

models meet the specific demands of applications. By focusing on aspects such as compression ratio, inference time, computational cost, model accuracy, and peak runtime memory footprint, a comprehensive understanding of the impact of compression techniques can be achieved.

The compression ratio (*CR*) quantifies the reduction in model size achieved after compression. It is defined as the ratio of the size of the original model to the size of the compressed model, providing a measure of the efficiency in reducing storage requirements. Inference time (*IT*) represents the time required for the model to process input and produce output. It directly impacts the responsiveness of the model, particularly in real-time applications, where lower *IT* values signify improved performance.

The computational cost (*FLOPs*) reflects the resources needed to train and execute the model. This parameter includes considerations of energy and processing power, making it critical for scenarios involving devices with limited computational capabilities or energy constraints. The accuracy (*Acc*) evaluates the predictive performance of the compressed model, ensuring that its functionality is retained after compression. Accuracy is typically measured against a validation dataset and compared to the original model.

The maximum memory footprint (*PMF*) assesses the maximum memory usage during model execution. This parameter is especially important for the deployment of models in environments with stringent memory restrictions, such as mobile devices or IoT devices, where minimizing *PMF* improves the feasibility and efficiency of the deployment.

These parameters collectively form a comprehensive framework for evaluating model compression methods, ensuring a balance between efficiency and functionality tailored to resource-constrained environments.

3.5 Weighted Score-Based Ranking Scheme

The weighted score-based ranking scheme is a comprehensive method developed to evaluate and compare different model compression techniques. It consists of six primary components — Scaling, Scoring, Weighting, Weighted Score, Weighted Score Average, and Ranking. Each component plays a crucial role in ensuring that the final ranks are accurate and reflect the models' performance across the various metrics.

3.5.1 Scaling

The first step in the ranking scheme involves scaling the results of each compression method to a uniform scale, ensuring that all metrics are comparable and weighted appropriately. This is achieved using the following scaling function.

$$n_{scaled} = \frac{n - n_{min}}{n_{max} - n_{min}} \times (r_{max} - r_{min}) + r_{min} \quad (11)$$

where:

- n_{scaled} denotes the scaled value, transforming the raw metric into a uniform scale.
- n is the original unscaled metric value to be transformed.
- n_{min} and n_{max} represent the minimum and maximum observed values for the metric in all models, respectively.
- $r_{min} = 1$ and $r_{max} = c$ are the endpoints of the target scale range, chosen to standardize the score within a defined and manageable range. Here, c represents the count of compression methods, adapting the scale to accommodate the number of methods that are evaluated.

The scaling function linearly transforms the original metric values into a standardized scale that ranges from 1 to c . The adaptation of r_{max} to c ensures that the scoring range dynamically adjusts to the total number of compression methods that are being evaluated, allowing for a direct and objective comparison among a variable number of methods. This approach facilitates equitable scoring and enhances the granularity of the evaluation by scaling the maximum possible score to reflect the number of competing methods.

3.5.2 Scoring

After the scaling process, each scaled metric undergoes a scoring process, where it is assigned a score based on its performance relative to other results. The scores are calibrated on a scale from 1 to c , where c represents the count of the compression methods and also the highest possible score indicating the best performance.

3.5.2.1 Adapting Scores for Metric Characteristics

The scoring mechanism for evaluating model compression metrics must account for the nature of each parameter. For metrics like compression ratio and accuracy, where higher values indicate better performance, the scoring directly corresponds to the scaled values, with higher scaled values receiving higher scores. However, for metrics such as inference time, peak memory footprint, and computational cost, where lower values are preferable, the scoring is inverted to ensure that lower scaled values, reflecting better performance, receive higher scores. This balanced approach ensures that the scoring system accurately reflects the desired outcomes across different metrics.

3.5.3 Scoring Formula

The scoring formula translates the scaled values into the final scores based on the specific performance criteria of each metric. The formula accounts for whether a higher or lower metric value signifies better performance, as illustrated below.

$$score_i = \begin{cases} r_{max} - (n_{scaled} - 1) & \text{if lower values are better} \\ n_{scaled} & \text{if higher values are better} \end{cases} \quad (12)$$

where:

- $score_i$ is the score assigned to the i -th metric, reflecting its performance relative to the desired result.
- n_{scaled} is the scaled value of the metric, adjusted to a range that standardizes all metrics for a fair comparison.
- $r_{max} = c$ is the maximum value on the scoring scale, corresponding to the highest possible score, which is set based on the count of compression methods evaluated (c).

The scoring rules for evaluating compression metrics depend on whether higher or lower values indicate better performance. These rules ensure that the scoring system appropriately reflects the desired outcomes for each metric.

For metrics where lower values indicate better performance, such as inference time or error rates, scores are calculated by subtracting the scaled value from r_{max} . This approach ensures that lower values, representing better results, correspond to higher scores. The scoring is expressed mathematically as:

$$score_i = r_{max} - (n_{scaled} - 1) \quad (13)$$

In contrast, for metrics where higher values are better, such as compression ratio or accuracy, the scaled value is directly assigned as the score. This approach aligns the scoring with the goal of achieving higher values for these metrics. The corresponding formula is as follows:

$$score_i = n_{scaled} \quad (14)$$

These equations provide a standardized method for scoring, facilitating objective and consistent evaluation of different compression techniques.

3.5.4 Weighting

Weighting is a crucial step in the evaluation process, as it reflects the relative importance of each metric in the overall assessment. This subsection explains how weights are assigned to each metric and discusses the rationale behind these decisions.

The weights are determined on the basis of the significance of each metric in achieving the desired outcomes of the compression process. Factors influencing weight assignment include the specific requirements of the application, the performance objectives, and the potential impact of each metric on the overall utility of the compressed model. The weighting process involves the following steps:

The process of assigning weights to evaluate compression methods involves several structured steps to ensure alignment with specific application goals and priorities.

First, all relevant metrics are identified that influence the performance and utility of compression methods, such as compression ratio, accuracy, and inference time. This initial step establishes the foundation for a comprehensive evaluation.

Next, these metrics are prioritized on the basis of their relative importance to the specific objectives of the model compression. For instance, reducing inference time might be more critical in real-time applications, whereas minimizing model size could take precedence in environments with severe storage constraints.

Once the priorities are established, numerical weights are assigned to each metric. These weights reflect their relative importance and ensure that the evaluation process emphasizes the most critical aspects of performance. For example, in scenarios where minimizing inference time is the primary goal, a higher weight is assigned to that metric. In contrast, when model size is the most critical factor, the compression ratio receives greater weight.

This systematic weighting process ensures that the evaluation aligns with the application's needs, providing a scoring system that accurately reflects the desired performance trade-offs and objectives.

3.5.5 Weighted Score

The weighted score for each metric is calculated by multiplying the score by its respective weight. This step integrates the importance of the metric with its performance.

$$composite_score = \sum_{i=1}^n (score_i \times weight_i) \quad (15)$$

where:

- *composite_score* is the weighted score for each model, indicating its overall performance.

- $score_i$ is the score of the i -th metric, which has been adjusted according to the scale and importance of the metric.
- $weight_i$ is the normalized weight for the i -th metric, reflecting its relative importance in the overall assessment.
- n is the total number of metrics used in the evaluation.

3.5.6 Weight Score Average

The weighted score average is computed by averaging the composite scores of all metrics for each compression method. This average provides a single score that represents the overall performance of the method.

$$weighted_score_average = \frac{composite_score}{\sum_{i=1}^n weight_i} \quad (16)$$

3.5.7 Ranking

Finally, the compression results are ranked based on their weighted score averages (i.e., the mean weighted scores). The method with the highest average receives the highest rank, and the rank proceeds in descending order of the average scores. This ranking method highlights the method that performs best across all considered metrics and aligns with the application's specific requirements.

$$\text{Rank} = \text{order by } weighted_score \text{ descending} \quad (17)$$

These components collectively form a robust approach for evaluating and ranking model compression methods, ensuring that the final decisions are grounded in a thorough quantitative analysis.

3.5.8 Application of Weighted Score-Based Ranking Scheme

This subsection details the practical application of the weighted score-based scheme to assess the efficacy of different model compression methods applied to a benchmark CNN model (Table 2). The evaluation considers key performance metrics —CR, IT, FLOPs, Acc, and PMF.

3.5.8.1 Benchmark Model Description

The benchmark model, designed for an image classification task, serves as the base model for the experiment. Performance metrics were initially recorded to establish a baseline for comparison before the application of any compression methods.

The architecture of the CNN benchmark is described in Table 3. The architecture takes a three-channel input image of size (64, 6, 3) as input, and (3, 3) convolutional filters (kernels) were used throughout the entire network, resulting in a total of 1,106,209 parameters. CNN includes a stack of two sets of CONV2D, RELU, BN, and POOL layers, followed by a set of CONV, RELU, and BN. The final block is the dense block, which consists of layers of FC, BN, FC, and SOFTMAX, as shown in Table 2.

3.5.8.2 Compression Methods Evaluated

The study evaluates the following state-of-the-art compression methods:

The 8 bit quantization (Section 2.1.1.3) reduces the precision of the weights from floating point to a full 8 bit integer. The full precision base model was quantized using the

Table 2: A table showing a summary of all the layers of the baseline model architecture.

Layer Type	Output Size	Parameters
CONV2D	(None, 64, 64, 32)	864
BN	(None, 64, 64, 32)	96
MAXPOOL2D	(None, 32, 32, 32)	0
CONV2D	(None, 32, 32, 64)	18,432
BN	(None, 32, 32, 64)	192
MAXPOOL2D	(None, 16, 16, 64)	0
CONV2D	(None, 16, 16, 64)	36,928
BN	(None, 16, 16, 64)	192
FLATTEN	(None, 16384)	0
DENSE	(None, 64)	1,048,576
BN	(None, 64)	192
DENSE1	(None, 11)	704
ACTIVATION	(None, 11)	0
TOTAL PARAMETERS		1,106,209

symmetric mode 8-bit signed full integer quantizer defined in Eq. (1) and Eq. (2). The scaling function transforms the input float tensors (weights and activations) of the base model to a quantized 8 bit output.

Binarization (Section 2.1.1.1) minimizes the precision of the weights to binary values, significantly reducing the size of the model. The base model was binarized using Larq, an open-source Binary Neural Network library built on Keras. The binarization function of the non-zero sign function b_{1bit} transforms the float tensor of the model to a binary output $(-1, +1)$ as shown in Eq. (3).

The weight pruning (Section 2.1.4) removes insignificant weights from the model to decrease its complexity and size. The base model was pruned using magnitude-based weight pruning as opposed to the neuron-based method, because it was observed that magnitude-based weight pruning does not affect model accuracy significantly. The weights of the pruned base model were selected using rank-based criteria, calculated using the absolute value of the individual weight in Eq. (10).

The decomposition of low-rank tensors (Section 2.1.3.1) decomposes the model tensors into low-rank approximations to reduce computational demands. The dense layer of the base model was transformed into TT matrices as described in Eq. (8). The transformation parameters used include a TT-rank of 4, an input dims of (16,16,8,8), and an output dims of (4,4,2,2). The decomposition transforms the dense layer into a TT layer with fewer parameters while maintaining the expressiveness of the layer.

Knowledge distillation (Section 2.1.2) Transfers knowledge from the base model to a smaller, more compact model. The class probabilities vector of the base model for each data point was computed and stored. These probabilities vectors, also called soft labels, were distilled to a compact model. The compact model was trained using both the soft and hard labels, and the overall losses generated by the model were combined and weighted, as defined in Eq. (5). The architecture of the compact model is described in Table 3.

Table 3: A table showing a summary of all the layers of the compact model architecture.

Layer Type	Output Size	Parameters
CONV2D	(None, 64, 64, 16)	432
BN	(None, 64, 64, 16)	48
MAXPOOL2D	(None, 32, 32, 16)	0
CONV2D	(None, 32, 32, 32)	4,640
BN	(None, 32, 32, 32)	96
MAXPOOL2D	(None, 16, 16, 32)	0
CONV2D	(None, 16, 16, 32)	9,248
BN	(None, 16, 16, 32)	96
FLATTEN	(None, 8192)	0
DENSE	(None, 32)	262,208
BN	(None, 32)	96
DENSE1	(None, 11)	363
ACTIVATION	(None, 11)	0
TOTAL PARAMETERS		277,227

3.5.8.3 Scoring and Weighting Process

The scoring and weighting process for evaluating model compression methods involves several key steps, designed to ensure a robust and objective assessment of each method's effectiveness. This process transforms raw metric data into a normalized score that reflects both the performance of the models and their relative importance, as determined by the weighting system.

1. **Scaling (Normalization of Raw Metric Values):** The first step involves scaling the raw performance metrics to a normalized scale from 1 to c (where $c = 5$ in our study, corresponding to the number of evaluated compression methods). This normalization adjusts the metric values to a uniform scale, allowing for equitable comparison across different metrics and models.

$$n_{scaled} = \frac{n - n_{min}}{n_{max} - n_{min}} \times (c - 1) + 1$$

where n is the raw metric value, n_{min} and n_{max} are the minimum and maximum values for that metric, respectively, and c is the top score value.

2. **Scoring (Adapting Metric Scores):** Once metric values are scaled, each is scored on the adjusted scale from 1 to c . This scoring reflects whether higher or lower values indicate better performance, with adjustments made accordingly to ensure that higher scores always reflect better performance.
3. **Weight Assignment:** Weights are assigned to each metric based on their importance to the overall effectiveness of the compression method. This importance is determined through stakeholder analysis and the specific operational context in which the model operates.

Weight for each metric = Assigned based on strategic importance

4. **Calculation of Weighted Score Summation:** The mean weighted score for each model is calculated by taking the weighted average of the scored metrics.

$$composite_score = \sum_{i=1}^n (score_i \times weight_i) \quad (18)$$

where:

- *composite_score* is the weighted score for each model, indicating its overall performance.
- *score_i* is the score of the *i*-th metric, which has been adjusted according to the metric's scale and importance.
- *weight_i* is the normalized weight for the *i*-th metric, reflecting its relative importance in the overall assessment.
- *n* is the total number of metrics used in the evaluation.

5. **Weighted Score Average:** To normalize the mean weighted score, it is divided by the total sum of the weights, providing a score average that allows for comparison across models with varying numbers of metrics and weight distributions.

$$weighted_score_average = \frac{composite_score}{\sum_{i=1}^n weight_i} \quad (19)$$

where:

- *weight_i* is the normalized weight for the *i*-th metric, reflecting its relative importance in the overall assessment.
- *n* is the total number of metrics used in the evaluation.

6. **Ranking:** Finally, models are ranked based on their weighted score averages. The model with the highest score average is ranked highest, indicating it is the most effective according to the evaluated metrics and assigned weights.

$$Rank = \text{order by } weighted_score \text{ descending} \quad (20)$$

Each step in this process is meticulously designed to ensure that the evaluation of compression methods is not only comprehensive and systematic, but also aligned with the specific needs and constraints of the application domain. This methodical approach provides a robust framework for making informed decisions about which compression method best meets the operational requirements.

3.5.8.4 Weights Assignment Strategy

In the evaluation of the compression methods of the model applied to the benchmark model detailed in Table 3, the weights were objectively assigned to each performance metric. These assignments were based on different optimization objectives, each tailored to meet specific application requirements. To accommodate the comparative analysis, sets of weights were systematically generated such that each weight is an integer between 1 and *c* - with *c* = 5, representing the total number of compression methods evaluated. This ensures that the impact of each metric is scaled appropriately according to its importance in various application contexts. The following outlines the simulation of different weighting scenarios, corresponding to five distinct objectives: performance-oriented, efficiency-oriented, balanced approach, memory reduction focus, and cost-sensitive model.

Each set of weights is designed to reflect the intended application requirements, ensuring that the evaluation metrics align closely with the performance goals shown in Table 4. This method allows for a structured and quantifiable approach to evaluating the

effectiveness of different compression methods with respect to the requirements of the application.

Table 4: Summary of Weight Assignments for Different Optimization Goals

Goals	CR ^a	IT ^b	FLOPs ^c	Acc. ^d	PMF ^e
Performance-Oriented	2	3	3	5	2
Efficiency-Oriented	4	4	3	2	2
Balanced Approach	3	3	3	3	3
Memory Reduction Focus	2	2	1	3	5
Cost-Sensitive Model	4	3	4	2	2

- ^a CR: Compression Ratio
- ^b IT: Inference Time
- ^c FLOPs: Floating Point Operations
- ^d Acc.: Accuracy
- ^e PMF: Peak Memory Footprint

3.6 Results and Discussion

The baseline CNN model, prior to any compression, serves as the benchmark for the experiments. Table 5 shows the performance metrics of the baseline model in several key metrics — Compression ratio, inference time, computational cost, accuracy, and peak memory footprint.

Table 5: Benchmark Model Metrics

Metrics	Baseline
Compression Ratio	4429.61
Infer. Time (ms)	22.88
Comp. Cost (MFLOPs)	66.44
Acc. (%)	77.23
PMF (KB)	8907.81

- ^a CR: Compression Ratio
- ^b Infer. Time: Inference Time
- ^c FLOPs: Floating Point Operations
- ^d Acc.: Accuracy
- ^e PMF: Peak Memory Footprint

Table 6 presents the raw metric values obtained from the compression methods, providing initial insight into their individual impacts.

Table 6: Raw Metric Values.

Metric	Quant.	Bin.	Prun.	Distil.	Tensor-Tr.
Compression Ratio	3.96	41.99	3.38	4.00	18.23
Infer. Time (ms)	13.65	5.40	22.64	12.55	18.53
Comp. Cost (MFLOPs)	7.29	6.96	66.44	35.12	64.34
Acc. (%)	76.95	67.10	74.64	72.05	72.91
PMF (KB)	3705.47	1775.78	8900.78	2300.48	3617.19

To facilitate an objective comparison across the compression methods, each metric was scaled using Eq. (11) as shown in Table 7. Each metric was scaled from 1 to 5 (cor-

responding to the total number of compression methods evaluated) based on their performance relative to others within the same metric category. The normalization process allows to objectively score each method.

Table 7: Scaled Metric Values.

Metric	Quant.	Bin.	Prun.	Distil.	Tensor-Tr.
Compression Ratio	1.06	5.00	1.00	1.06	2.54
Infer. Time (ms)	2.91	1.00	5.00	2.66	4.05
Comp. Cost (MFLOPs)	1.02	1.00	5.00	2.89	4.86
Accuracy (%)	5.00	1.00	4.06	3.01	3.36
PMF (KB)	2.08	1.00	5.00	1.29	2.03

After the scaling process, each scaled metric undergoes a scoring process by applying Eq. (12), where it is assigned a relevance score is assigned based on its performance relative to other results.

The scores were transformed on a scale from 1 to c , where $c = 5$ represents the count of compression methods and also the highest possible score indicating the best performance. The scoring translates the scaled values into final scores based on the specific performance criteria of each metric as shown in Table 8.

Table 8: Scored Metric Values.

Metric	Quant.	Bin.	Prun.	Distil.	Tensor-Tr.
Compression Ratio	3.00	5.00	1.00	3.00	4.00
Infer. Time (ms)	3.00	5.00	1.00	4.00	2.00
Comp. Cost (MFLOPs)	4.00	5.00	1.00	3.00	2.00
Acc. (%)	5.00	1.00	4.00	2.00	3.00
PMF (KB)	2.00	5.00	1.00	4.00	3.00

In Table 4, each set of weights is strategically designed to reflect distinct optimization objectives—**performance**, **efficiency**, **balanced**, **memory-reduction**, and **cost-sensitive**. These categories were tailored to meet specific application requirement contexts, demonstrating the flexibility in evaluating compression methods to align with varying operational requirements. This differentiation in weighting profiles underscores the adaptability of the proposed evaluation method, allowing it to cater to diverse application needs and highlight the most suitable compression strategies based on targeted optimization goals.

Table 9: Weighted Score of the Performance-Based Weight Set Profile.

Metric	Weight	Quant.	Bin.	Prun.	Distil.	Tensor-Tr.
Compression Ratio	2.00	6.00	10.00	2.00	6.00	8.00
Infer. Time (ms)	3.00	9.00	15.00	3.00	12.00	6.00
Comp. Cost (MFLOPs)	3.00	12.00	15.00	3.00	9.00	6.00
Acc. (%)	5.00	25.00	5.00	20.00	10.00	15.00
PMF (KB)	2.00	4.00	10.00	2.00	8.00	6.00
Weighted Score		56.00	55.00	30.00	45.00	41.00

To illustrate the application of these weight set profiles, the performance-oriented profile was specifically chosen for detailed analysis. The weighted scores and their corresponding averages for each compression method, using the performance-based weight

set, were calculated according to Equations (18) and (19). The results of these calculations are presented in Table 9, providing a quantified demonstration of how each compression method performs under a performance-centric evaluation criterion.

The weighted score averages were ranked using Eq. (18) as shown in Table 10. The results of the performance-based weight set profile for the evaluated compression methods —quantization, binarization, pruning, KD, and TD were summarized in Table 10. Each method was assessed in five key metrics: Compression Ratio, Inference Time, Computational Cost (MFLOP), Accuracy, and Peak Memory Footprint (PMF), with corresponding weights reflecting the relative importance of these metrics in performance-oriented applications.

Quantization is highest with a weighted score average of 3.73, demonstrating a strong balance between maintaining the accuracy of the model (76.95%) and improve the inference time (13.65 ms) & computational cost. Although quantization did not lead to the most significant reduction in model size, its overall performance across the metrics evaluated makes it the most effective compression method under the performance-based profile.

Binarization follows closely with a weighted score average of 3.67. This method excels in compression ratio (41.99) and inference time (5.40 ms), but its relatively lower accuracy (67.10%) reduces its overall score. Although binarization significantly reduces the model size and speeds up inference, the trade-off in accuracy makes it less favorable for applications that require high predictive performance.

Pruning ranks lowest with a weighted score average of 2.00, primarily due to its poor performance in inference time (22.64 ms) and computational cost (66.44 MFLOPs), despite showing improvements in model accuracy (74.64%). Although pruning can effectively remove redundant parameters, its computational overhead and slower inference time hinder its overall performance in this evaluation.

KD and TD rank third and fourth, respectively. KD offers a balanced approach with a weighted score average of 3.00, combining acceptable accuracy (72.05%) and inference time (12.55 ms). TD, while effective in reducing model size (Compression Ratio: 18.23), exhibits moderate performance on most metrics, leading to a weighted score average of 2.73.

Table 10: Rank of the Performance-Based Weight Set Profile.

Metric	Weight	Quant.	Bin.	Prun.	Distil.	Tensor-Tr.
Compression Ratio	2.00	6.00	10.00	2.00	6.00	8.00
Infer. Time (ms)	3.00	9.00	15.00	3.00	12.00	6.00
Comp. Cost (MFLOPs)	3.00	12.00	15.00	3.00	9.00	6.00
Acc. (%)	5.00	25.00	5.00	20.00	10.00	15.00
PMF (KB)	2.00	4.00	10.00	2.00	8.00	6.00
Weighted Score Avg.		3.73	3.67	2.00	3.00	2.73
Rank		1	2	5	3	4

3.7 Summary

This chapter addressed a critical gap in the evaluation of DL model compression methods by developing a numerically based mathematical method. The primary objective was to establish a standardized approach that allows for the systematic and objective evaluation and ranking of various compression techniques based on their impact on key performance metrics, such as compression ratio, inference time, computational cost, accuracy and peak memory footprint.

The core objective of this chapter was to answer the research question **RQ1**: "*Can a numerically based mathematical method be developed to quantitatively rank the efficiency, effectiveness and suitability of model compression methods?*" The presented approach not only quantifies the effects of different compression techniques on model characteristics, but also provides a clear, objective framework for comparing and selecting the most appropriate compression method based on specific application requirements.

To demonstrate the practical application of this method, a performance-based weight set profile was selected and applied to five different compression methods: **quantization**, **binarization**, **pruning**, **knowledge distillation**, and **tensor decomposition**. Each method was evaluated based on its ability to balance trade-offs between reducing model size, minimizing computational complexity, and maintaining acceptable accuracy & inference times.

The results of the evaluation, summarized in Table 10, showed that **quantization** ranked highest with the best weighted score average. This method demonstrated an optimal balance between maintaining model accuracy and improving computational efficiency, making it particularly suitable for performance-critical applications. **Binarization** ranked second, excelling in compression ratio and inference time but suffering from reduced accuracy, limiting its overall effectiveness. **Pruning**, although effective in parameter reduction, was hampered by higher computational costs and slower inference times, which led to its lowest rank in this performance-based profile.

The use of the performance-based profile allowed for a focused evaluation of how each method performs under strict performance requirements, where accuracy, inference time, and computational efficiency are paramount. The weighted score averages were calculated using the methodology outlined in Equations (18) and (19), providing an objective basis to rank the compression methods.

This chapter makes a significant contribution to the field by introducing a numerical-based mathematical approach for systematically evaluating model compression methods. Provides a standardized yet flexible methodology that adapts to diverse application needs through customizable weight set profiles. By offering a quantitative framework for ranking the efficiency, effectiveness, and suitability of compression techniques, this chapter equips researchers and practitioners with a robust tool to make informed decisions when optimizing deep learning models for specific use cases.

Although the performance-based profile was used in this demonstration, flexibility allows the application of other optimization objectives, such as **efficiency**, **memory-reduction**, **balanced**, or **cost-sensitive** profiles, depending on the operational context and requirements of the application. For instance, in scenarios where memory usage is a critical factor, the *memory-reduction profile* could be applied, where peak memory footprint is assigned a higher weight relative to other metrics. Similarly, in cost-sensitive environments, the *cost-sensitive profile* can be used to prioritize compression ratio and computational cost over other performance metrics.

This flexibility underscores the utility of the proposed methodology in adapting to different operational constraints and priorities, thereby ensuring its broad applicability

across a range of deep learning applications. The results also revealed several trade-offs inherent in the use of different compression methods. For example, while binarization excels in reducing model size and improving inference time, it significantly compromises accuracy, making it less suitable for applications where predictive performance is a top priority. Conversely, knowledge distillation offers a more balanced approach but is less effective in reducing computational complexity compared to other techniques such as pruning or tensor decomposition.

3.8 Conclusion

The development of this numerically based mathematical method for evaluating and ranking model compression methods provides a powerful tool for addressing the challenges associated with the optimization of the DL model. By offering a standardized and quantitative approach, this method fills a critical gap in the field, enabling systematic comparison of compression techniques and supporting the selection of methods based on the specific requirements of real-world applications.

In addition to the performance-based profile demonstrated in this chapter, the flexibility to apply different weight set profiles ensures that the methodology can be adapted to suit a variety of optimization goals. This enables researchers and practitioners to focus on optimizing models for specific applications, whether that involves minimizing memory usage, improving computational efficiency, or balancing between multiple objectives.

Future work could focus on further refining the profiles of the weight set by incorporating additional metrics or exploring specialized applications, such as edge computing or the Internet of Things (IoT), where resource constraints are even more stringent. Additionally, this methodology could be extended to evaluate the impact of compression methods on more complex DL architectures, such as large language transformers or recurrent neural networks.

In conclusion, this chapter has successfully developed and validated a comprehensive approach to model compression evaluation, providing theoretical and practical contributions to the optimization of DL models for performance-critical applications. The flexibility of the weight-based evaluation method ensures that it can continue to evolve, providing valuable information for a wide range of applications.

4 ENSEMBLE OF COMPRESSION METHODS

This chapter addresses the research question (RQ2): ***"Can a deep model compression ratio of 32x be achieved while maintaining an acceptable level of accuracy for practical deployment"***? It explores the practical limits of model compression techniques, assessing the feasibility of attaining such a significant compression ratio while ensuring that performance levels remain suitable for application requirements.

Publication II: (Section 4.2)

O. A. Ademola, E. Petlenkov and M. Leier, "Ensemble of Tensor Train Decomposition and Quantization Methods for Deep Learning Model Compression", 2022 International Joint Conference on Neural Networks (IJCNN), Padua, Italy, 2022, pp. 1-6, doi: 10.1109/IJCNN55064.2022.9892626.

4.1 Limitations of Single Compression Methods

Single compression methods as evaluated in Section 1.3 often struggle to strike an optimal balance between key model metrics —CR, Acc, IT, FLOPs, and PMF. Each method has its advantages, but also has inherent limitations that restrict its effectiveness when applied individually.

To overcome these limitations that individual methods pose, combining two or more compression methods into an ensemble (Section 2.1.6) can provide deeper compression while preserving the performance of the model. The ensemble method leverages the strengths of each technique and compensates for their weaknesses, resulting in a more efficient and compact model suitable for deployment in resource-constrained environments.

Although deep model compression using ensemble offers the potential to deploy compressed models on resource-limited devices, several challenges arise when attempting to compress models at high ratios, such as 32x or more, without significantly impacting accuracy. The possibility is explored in this chapter.

4.2 Ensemble of Tensor Train Decomposition and Quantization

DL models, particularly CNNs, are often too large and computationally expensive to deploy directly on edge devices or embedded systems with limited resources. Although individual compression techniques such as TTD and 8-bit quantization offer substantial improvements in reducing the size and inference time of the model, each has its own limitations when applied independently (Table 6). TTD achieves an impressive compression ratio, while quantization offers computational speedup and reduced precision storage.

The choice of combining tensor TTD and 8-bit quantization was inspired by the results of my first publication, ***Evaluation of Deep Neural Network Compression Methods for Edge Devices Using Weighted Score-Based Ranking Scheme*** in Appendix 1 (also discussed in Section 3.1). In this publication, different DNN compression methods were ranked based on a weighted score-based system.

The motivation behind combining these methods is to leverage the strengths of both techniques, achieving deeper model compression than what either method could achieve individually. By first applying TTD to reduce the memory and computation demands of dense layers and then quantizing the resulting model, a significant compression was achieved without sacrificing too much accuracy. This two-stage approach allows for greater flexibility in optimizing both memory and computational efficiency.

4.2.1 Architecture of the Ensemble Compression Pipeline

The proposed ensemble compression pipeline integrates Tensor Train Decomposition (TTD) and 8-bit integer quantization in a sequential process to achieve effective compression of deep learning models, as described in Figure 3. Each step contributes uniquely to optimizing the model, resulting in a compact but efficient version suitable for deployment on resource-constrained devices.

The pipeline begins with a CNN base model (Table 2), which comprises convolutional and fully connected (FC) layers. TTD is then applied specifically to the dense layer, which typically contains the majority of the model's parameters. By decomposing the weight tensors of this layer into low-rank tensor cores, TTD substantially reduces the memory footprint while retaining the model's structural integrity. Following TTD, 8-bit integer quantization is performed on the weights, biases, and activations of the tensor-trained model. This step reduces the parameter precision from 32-bit floating point to 8-bit integers, effectively decreasing the model size and computational demands. Careful calibration ensures minimal performance loss during this quantization process.

The final output of the pipeline is a highly optimized, quantized tensor-trained model that combines the advantages of TTD and 8-bit quantization, offering reduced memory and computational requirements while maintaining accuracy. This makes the model particularly well suited for applications on low-resource platforms such as embedded systems and mobile devices.



Figure 3: An illustration of the trainable pipeline for model compression using the ensemble of Tensor Train Decomposition (TTD) and 8-bit quantization. The pipeline begins with a trained base model, followed by TTD applied to dense layers, and concludes with 8-bit quantization of the resulting tensor trained model. Appendix 2, Figure 3

The integration of TTD and 8-bit quantization offers several advantages over using either technique individually, creating a synergistic effect that enhances the compression and efficiency of deep learning models.

One key benefit is the ability to achieve deep compression ratios by leveraging the strengths of both methods. TTD significantly reduces the memory requirements of fully connected layers, while 8-bit quantization further compresses the model by reducing parameter precision. This combination delivers a level of compression that neither method could achieve independently (Table 6).

Furthermore, the ensemble method provides a balanced reduction in both memory usage and computational demands (Table 6). TTD minimizes memory footprints, particularly in dense layers, and quantization accelerates computations through the use of low-precision 8-bit integers. Together, these methods enable real-time inference on resource-constrained devices, making the compressed model well suited for edge and embedded applications.

The ensemble approach is also designed to retain accuracy despite aggressive compression. Careful selection of TT-ranks and the selective application of quantization ensure that the resulting quantized tensor-trained model maintains acceptable performance levels. In most cases, the accuracy degradation is kept within 10% of the original uncom-

pressed model, offering a practical trade-off between the compression efficiency and the model performance.

4.2.2 Challenges of the Ensemble Method

While the ensemble method offers substantial benefits, its implementation poses certain challenges that require careful consideration as established in the surveyed literature (Section 1.2). One critical factor is selecting the appropriate TT-ranks during tensor train decomposition. Lower TT-ranks achieve higher compression but may significantly impact model performance. Striking the right balance requires meticulous tuning and experimentation to ensure optimal performance for the specific model and data set.

Another challenge lies in managing accuracy loss during 8-bit quantization. Although quantization substantially reduces the size of the model and improves speed, it can degrade accuracy, especially in sensitive architectures and layers. The ensemble method must carefully calibrate the quantization to maintain the overall performance of the model within acceptable limits.

By addressing these challenges through careful tuning and applying both techniques in tandem, the ensemble method delivers a compact and efficient model that meets the demands of the application.

4.3 Experimental Setup and Results

4.3.1 Architecture of the Baseline Model

The experiments in this study were carried out using a baseline CNN model, which was trained on a custom data set. The architecture of the baseline model is shown in Table 2. The model consists of several convolutional layers followed by max-pooling layers and an FC layer. The FC layer contributes to the majority of the model parameters (weights), making it an ideal target for TTD.

The performance of the model was evaluated using these model metrics — **model size, peak memory footprint, accuracy, model speed, and computational cost** as evaluated in Section 3.2. The objective of the experiments was to evaluate the effects of applying TTD and 8-bit quantization on this baseline model in terms of the model metrics. This is to fulfill the goal of exploring the practical limits of model compression and to evaluate the feasibility of achieving a very deep compression ratio while maintaining acceptable performance limits required by applications.

4.3.2 Ensemble End-to-End Trainable Pipeline

To fully exploit the advantages of TTD and 8-bit quantization, I developed an end-to-end trainable pipeline that compresses the CNN model of benchmark using an ensemble of these two techniques, as shown in Figure 4. The pipeline was designed to be trainable from scratch, meaning that the model was optimized with compression methods applied during training, ensuring that the compressed model maintains an acceptable level of performance while achieving significant reductions in size and computational complexity.

The pipeline integrates three core components to achieve efficient model compression and performance optimization.

The pipeline begins with a trained baseline CNN model, comprising convolutional layers and a fully connected (FC) layer. The FC layer is the primary target for compression as it typically contains the majority of the model parameters. By focusing on this layer, the pipeline aims to address the significant memory and computational demands associated with dense connections in deep learning models.

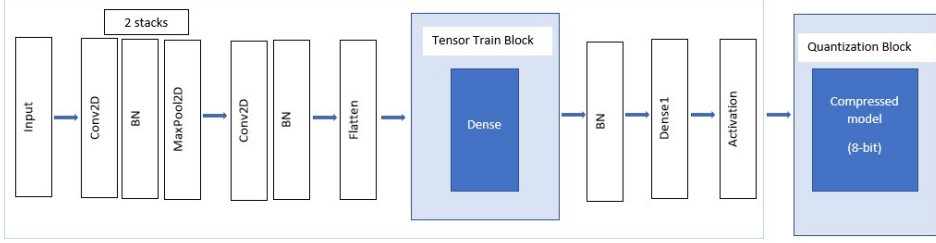


Figure 4: A block representation of the end-to-end trainable pipeline for model compression using an ensemble of Tensor Train Decomposition (TTD) and 8-bit quantization. The pipeline first applies TTD to compress the dense layers, followed by 8-bit quantization to further reduce the model size and computational cost. Appendix 2, Figure 4

TTD (Section 2.1.3.1) is then applied to the FC layer during training. This process factorizes large weight matrices into smaller tensor cores, significantly reducing the memory footprint of the model. TTD enables the network to maintain its performance by learning an efficient low-rank representation, ensuring that the reduced parameter set retains the critical features required for effective predictions.

Finally, 8-bit integer quantization (Section 2.1.1.3) is used to further compress the model. In this step, the weights, biases, and activations of the tensor-trained model are converted from 32-bit floating point to 8-bit integers. By incorporating quantization-aware training, the model adjusts to the lower-precision format, ensuring minimal accuracy degradation while achieving substantial reductions in computational and memory requirements.

The final compressed model generated by the pipeline is called the quantized tensor trained model (*QuanTT*). This model combines the strengths of TTD, which reduces the number of parameters in memory, and 8-bit quantization, which accelerates inference by using low-precision integer arithmetic.

4.3.3 Training and Optimization Process

The pipeline was designed for end-to-end training, incorporating both TTD and quantization-aware during the forward and backward passes. This approach allows the model to adapt to the compressed structure and reduced precision of the weights and activations throughout the training. Although actual quantization was applied post-training, quantization-aware induction ensures that the model learns to operate effectively under the constraints of lower precision, thereby minimizing potential accuracy loss that could occur.

The optimization process includes the following.

- **Backpropagation with TTD:** As the model trains, the tensor cores are updated through backpropagation, ensuring that the compressed tensor representations are optimized to minimize the loss function. The TT-ranks, which control the degree of compression, are also tuned during this process.
- **Quantization-Aware Training:** 8-bit quantization was integrated into the training process, allowing the model to learn how to operate effectively with low precision weights and activations. Quantization-aware training helps the model mitigate the accuracy degradation that typically accompanies post-training quantization.

The end-to-end trainable pipeline enables deep learning model compression for deployment in resource-constrained environments. By integrating TTD with 8-bit quantization, this approach produces highly compressed models that maintain strong performance, making them ideal for real-world applications with limited computational and memory resources.

4.3.4 Tensor Train Configuration

TTD was applied to the dense layer of the baseline model, which accounts for approximately 94.79% of the model's parameters. This layer, with an input dimension of 16,384 and an output dimension of 64, was converted into a TT layer to significantly reduce the memory footprint.

Various TT configurations were tested to find the optimal TT rank for compression. Table 11 presents the different combinations of TT-ranks, input dimensions, and output dimensions explored in the experiments. The objective was to achieve the smallest possible TT-ranks while maintaining satisfactory model performance.

Table 11: Tensor Train Layer Configurations.

Configuration Type	TT-layer Parameters		
	TT-ranks	Input dims	Output dims
TT-I	4	(16,16,8,8)	(4,4,2,2)
TT-II	6	(16,16,8,8)	(4,4,2,2)
TT-III	8	(16,16,8,8)	(4,4,2,2)
TT-IV	10	(16,16,8,8)	(4,4,2,2)
TT-V	4	(16,8,16,8)	(4,2,4,2)
TT-VI	6	(16,8,16,8)	(4,2,4,2)
TT-VII	8	(16,16,8,8)	(4,4,2,2)
TT-VIII	10	(16,16,8,8)	(4,4,2,2)
TT-IX	4	(64,2,64,2)	(8,1,8,1)
TT-X	6	(64,2,64,2)	(8,1,8,1)
TT-XI	8	(64,2,64,2)	(8,1,8,1)
TT-XII	10	(64,2,64,1)	(8,1,8,1)

The original weight matrix of the dense layer, which has a shape of (16384, 64), was reshaped into a higher order tensor. The input dimension of 16,384 and the output dimension of 64 were factored into smaller dimensions. For example:

- (16, 16, 8, 8) refers to the factorized shape of the input tensor (in this case a 4-dimensional tensor), where each element represents the size of the factorized dimensions. These dimensions were chosen on the basis of the TT-ranks and the shape of the original tensor. For the input dimension of 16,384, it was decomposed into (16, 16, 8, 8), and similarly, the output dimension of 64 is decomposed into (4, 4, 2, 2).
- (4, 4, 2, 2) refers to the factorized dimensions of the output tensor after decomposition. These dimensions correspond to the TT-ranks and how the original output dimension (64) was divided into smaller tensor cores.

By decomposing the weight matrix into these smaller tensor cores, the overall number of parameters stored in memory was reduced due to the lower-rank approximation,

Table 12: Performance comparison of tensor trained compressed models with different configurations

Config. Type	Key Performance Indicators				
	MS (KB) ^a	PMF (KB) ^b	Acc. (%) ^c	Speed (ms) ^d	CC (MFLOPs) ^e
TT-I	243	3617.19	72.91	18.53	64.34
TT-II	250	3621.09	72.62	19.40	64.34
TT-III	259	3628.91	74.06	20.48	64.34
TT-IV	271	3644.53	70.32	21.80	64.34
TT-V	244	3613.28	73.48	18.48	64.34
TT-VI	251	3621.09	74.35	19.38	64.34
TT-VII	260	3632.81	72.62	20.38	64.34
TT-VIII	272	3640.62	70.89	21.78	64.34
TT-IX	278	3652.34	69.74	19.59	64.34
TT-X	321	3695.31	74.35	21.24	64.34
TT-XI	381	3746.09	74.92	23.21	64.34
TT-XII	458	3816.41	70.31	25.73	64.34

^aMS: Model Size, ^bPMF: Peak Memory Footprint, ^cAcc.: Accuracy, ^dSpeed: Model Speed, ^eCC: Computational Cost

allowing efficient compression without sacrificing too much accuracy. The TT-ranks (e.g. 4, 6, 8, 10) control the level of compression, as shown in Table 12, with smaller TT-ranks leading to higher compression, but potentially greater loss in accuracy.

Each configuration type (e.g., TT-I, TT-II, TT-III, TT-IV, TT-V, etc.) results in a tensor-trained model, as shown in Table 12. These models were evaluated and ranked using the weighted score-based ranking method discussed in Section 3.1 to obtain the configuration of the TT layer that produces the best performing tensor-trained model. The tensor trained model generated by the **TT-V** configuration type was the highest ranked and was selected for the final TT-layer configuration used in compression.

4.3.5 Tensor Trained Model Quantization

The parameters of the optimized tensor-trained model were quantized to 8-bit unsigned integers, further reducing memory, storage, and computational footprints. Quantization was applied to all layers except input and output layers to prevent significant accuracy loss, resulting in the quantized tensor-trained model (*QuantTT*).

QuantTT preserves the compressed structure introduced by TTD while enhancing memory efficiency and inference speed through 8-bit precision for parameter storage. By selectively quantizing only certain layers, accuracy loss typically associated with quantization was minimized, maximizing the compression benefits from tensor train decomposition.

4.3.6 Performance Comparison and Evaluation of Key Metrics

To evaluate the performance of the ensemble method, the base model was compared with three compressed variants: **tensor trained only**, **quantized only** and the **quantized tensor trained model (QuantTT)** produced by the ensemble method. Table 13 summarizes

Table 13: This table compares the baseline model with TTD, 8-bit quantization, and the ensemble method (QuanTT)

Model	MS (KB) ^a	PMF (KB) ^b	Acc (%) ^c	IS (ms) ^d	MFLOPs ^e
Baseline	4429.61	8907.81	77.23	22.88	66.44
Quantized Only	1119.40	3705.47	76.95	13.65	7.29
Tensor Trained Only	244.00	3613.28	73.48	18.48	64.34
QuanTT	76.70	1632.81	69.45	12.89	7.06

^aMS: Model Size

^bPM: Peak Memory Footprint

^cAcc: Accuracy

^dIS: Inference Speed

^eMFLOPs: Million Floating Point Operations

the results, highlighting key performance indicators such as model size, accuracy, inference speed, computational cost, and peak memory footprint.

Evaluation of key metrics highlights substantial improvements across various aspects of compressed models, particularly in the QuanTT model.

The QuanTT model achieves an exceptional reduction in size, compressing the baseline model from 4429.61 KB to just 76.7 KB, resulting in a remarkable **57x** reduction. This level of compression far exceeds the individual effects of tensor train decomposition or quantization alone, rendering the model highly compact and ideal for deployment in memory-constrained environments.

In terms of accuracy, the QuanTT model maintains a performance level of **69.45%**, despite experiencing a 10% reduction compared to the baseline. This balance between accuracy and efficiency ensures its suitability for scenarios that prioritize memory and computational efficiency over minor accuracy losses.

Inference speed is significantly improved, and the QuanTT model produces **2x** faster inference times than baseline. This improvement positions the model as an excellent choice for real-time applications on resource-limited devices, reducing latency, and improving response times.

The computational cost is another area of major improvement, and the QuanTT model achieving a **9x** reduction in mega floating-point operations (MFLOPs) compared to baseline. This efficiency makes it well-suited for devices with limited processing power, such as edge and embedded systems.

Peak memory usage sees a substantial decrease as well, with the QuanTT model consuming 1632.81 KB, which is **5x** lower than the baseline's 8907.81 KB. This significant reduction enables efficient deployment on devices with strict memory constraints, ensuring smooth operation in resource-limited environments.

Overall, the QuanTT model, produced by the TTD and 8-bit quantization ensemble, outperformed the other compressed models in every key metric. It provides the most effective balance between model size, speed, and computational efficiency while maintaining acceptable accuracy for practical applications. These results highlight the potential of this compression pipeline for real-world use-case in memory and computation-constrained environments.

4.4 Summary and Conclusion

4.4.1 Summary

This chapter introduced a novel ensemble method for deep compression of CNN models by integrating Tensor Train Decomposition (TTD) with 8-bit integer quantization.

The primary objective was to address the research question (RQ2): **"Can a deep model compression ratio of 32x be achieved while maintaining an acceptable level of accuracy for practical deployment?"**. The objective was to enable the deployment of deep learning models on devices with highly resource-constrained devices, where both memory and computational resources are severely limited.

Through a comprehensive review of traditional compression techniques such as pruning, quantization, knowledge distillation (KD), and tensor decomposition (TD), I highlighted their limitations when used in isolation. Although each technique is effective on its own, they often struggle to strike a balance between high compression ratios and preserving model accuracy. To overcome these limitations, an ensemble approach that combines TTD and 8-bit quantization was proposed to achieve substantial compression while maintaining performance within acceptable bounds.

The experimental results validated the effectiveness of this approach. By applying TTD to the dense layer and quantizing the parameters to 8-bit integers, the resulting compressed model, *QuantTT*, achieved a remarkable 57x reduction in model size, a 5x reduction in memory usage, and a 2x increase in inference speed, with only a 10% reduction in accuracy. These results demonstrate the potential of the proposed method for real-world applications in environments with stringent resource constraints.

Not only was the initial goal of a 32x compression ratio achieved, but it was significantly surpassed. The *QuantTT* model achieved a 57x compression ratio, far exceeding expectations. Despite this substantial reduction in model size, the compressed model retained a satisfactory level of accuracy, with only a minor 10% decrease compared to the baseline. This confirms that it is feasible to achieve deep compression without sacrificing the performance required for practical deployment.

The synergy between TTD and 8-bit quantization proved to be an optimal solution for this challenge. TTD effectively reduced the number of parameters stored in memory, while quantization improved computational efficiency by lowering parameter precision. The combination of these techniques allowed the model to balance compression and performance, making it a robust solution for deployment in resource-limited environments.

4.4.2 Conclusion

In conclusion, this chapter presented an end-to-end trainable pipeline for DL model compression that combines TTD and 8-bit quantization. By addressing RQ2, I demonstrated that it is possible to achieve a compression ratio far beyond the original 32x goal, reaching 57x, while maintaining a performance level suitable for different application contexts.

This offers a practical and scalable solution for deploying DL models on devices with very limited resources. The ability to compress models so drastically, without compromising critical performance metrics, is a significant step forward for the deployment of the DL model on the edge.

5 PRACTICAL USE CASE: OPTIMIZATION LIMITATION OF SCENE TEXT DETECTION AND RECOGNITION MODELS

This chapter addresses RQ3: "Can efficient model optimization be achieved without significantly compromising performance, especially for network architectures that are highly sensitive to compression and quantization?" focusing on scene text detection and recognition models. The aim is to overcome their sensitivity to compression by developing a method for efficient optimization that preserves performance, even on integer-only hardware.

Publication III: (Section 5.1)

O. A. Ademola, E. Petlenkov, and M. Leier, "Resource-Aware Scene Text Recognition Using Learned Features, Quantization, and Contour-Based Character Extraction," in IEEE Access, vol. 11, pp. 56865-56874, 2023, doi: 10.1109/ACCESS.2023.3283931.

5.1 Resource-Aware Scene Text Recognition Using Learned Features, Quantization, and Contour-Based Character Extraction

5.2 Scene Text Recognition In Embedded Systems

Scene text recognition refers to the detection and interpretation of textual information embedded within natural images or video frames. This task is crucial in a wide range of real-world applications, particularly those where decision making is based on visual context, such as intelligent transportation systems, autonomous navigation, and parcel sorting. In these scenarios, the recognition of text, such as road signs, traffic warnings, markers, or container numbers, allows systems to make intelligent and informed decisions.

While text in natural scenes is often easily recognizable to humans, for machines, it remains a challenging task. Variations in font size, orientation, background clutter, lighting conditions, and text distortions all contribute to the complexity of this problem. Detecting and recognizing scene text typically requires sophisticated algorithms that can adapt to these variations and extract accurate information in real-time.

Embedded systems, including autonomous robots, handheld & IoT devices, increasingly rely on scene text recognition for applications like navigation, labeling, and smart tracking. However, unlike general-purpose computing systems, embedded devices operate under stringent hardware constraints. These devices often lack the processing power, memory, and energy resources required to run large-scale DL models, which are typically used for tasks such as scene text recognition.

5.3 Resource Constraints in Embedded Hardware

Embedded systems, especially those used in real-time applications, are typically built on hardware platforms optimized for power efficiency rather than computational throughput. Microcontrollers and edge devices, for instance, often rely on integer-only hardware, restricting their ability to perform floating-point operations common in deep learning systems. These devices also have limited memory and processing power, making it impractical to deploy large, resource-intensive models directly.

Although modern deep learning methods excel at detecting and recognizing text in complex scenes, they require substantial resources in terms of storage, memory, and computation. For example, models based on convolutional neural networks (CNN) or recurrent neural networks (RNNs) require extensive floating-point computations, numerous

filter layers, and significant memory to achieve high accuracy. This poses a significant challenge when deploying these models on devices with constrained computational capacity and memory.

Given these constraints, there is a critical need for techniques that reduce the resource requirements of scene text recognition models while preserving high performance. This is particularly essential for systems operating on integer-only hardware, such as edge TPUs or microcontrollers, where all computations must be performed using integer arithmetic.

5.4 Challenges and Research Objectives

5.4.1 Challenges in Optimizing Scene Text Detection and Recognition Models

Optimizing scene text detection and recognition models, particularly those based on deep learning architectures, poses significant challenges. These models must handle the inherent variability of natural scenes, including various text orientations, lighting conditions, and background clutter, necessitating sophisticated algorithms for high accuracy. However, they are highly sensitive to compression, which can cause severe performance degradation if not managed carefully.

A primary challenge is that scene text recognition models often depend on floating point arithmetic for tasks like feature extraction, classification, and recognition. Compression techniques such as quantization and pruning, essential for deploying models on resource-constrained hardware, can introduce quantization errors and reduce model precision, severely affecting performance. In scene text recognition, even minor errors in text detection can lead to complete recognition failures, making these models especially vulnerable to optimization-induced errors.

Additional difficulties arise when deploying these models on embedded devices with integer-only hardware, such as edge TPUs and microcontrollers, which lack support for floating-point operations. Converting floating-point models to integer-based ones without careful management of quantization errors can lead to unacceptable accuracy losses. The challenge lies in achieving efficient compression while preserving the accuracy required for effective scene text recognition.

Moreover, while techniques such as quantization and pruning reduce model size and complexity, they also introduce new challenges. Quantization, for example, can cause a loss of precision that is particularly detrimental to text recognition models, which are sensitive to even small errors. The key challenge is to strike a balance between resource efficiency and maintaining the high level of accuracy needed for applications that rely on robust scene text recognition, especially when these models are deployed on integer-only hardware.

5.4.2 Research Objectives

This chapter seeks to address these optimization limitations inherent in scene text detection and recognition models, specifically addressing their sensitivity to model compression. The primary objective was to develop a method for efficient model optimization without significantly compromising the performance of the model, particularly when implemented on integer-only hardware.

The key research objectives focus on addressing the challenges associated with optimizing scene text recognition models for resource-constrained environments:

Handling Sensitivity to Compression: The research aims to develop strategies to mitigate the negative impact of quantization on scene text recognition models. This includes introducing mechanisms to counteract quantization-induced errors, thereby minimizing performance degradation.

Efficient Quantization Strategy: A tailored 8-bit quantization technique is proposed, designed specifically for scene text recognition models. This strategy accounts for the models' sensitivity to precision loss, ensuring accuracy retention while maintaining compatibility with integer-only hardware.

Resource-Efficient Solution: The research seeks to ensure that the optimized model operates efficiently on resource-constrained devices, such as microcontrollers and Edge TPUs, by reducing memory and computational requirements without compromising functionality.

Minimizing Performance Trade-Offs: Efforts are made to balance model size with performance, ensuring that compression techniques do not significantly impair the model's ability to detect and recognize text, even in challenging and noisy scene images.

Deploying on Integer-Only Hardware: The study emphasizes optimizing both text detection and recognition pipelines to function seamlessly on integer-only hardware. This allows for real-time text recognition in embedded systems that lack floating-point support.

The solution proposed in this chapter addresses the key issue of optimization sensitivity in scene text recognition models, offering a balanced approach that enables efficient deployment on embedded systems while maintaining an acceptable accuracy in challenging environments.

5.5 Scene Text Detection and Recognition

Scene text detection and recognition have evolved significantly over the past few years with advancements in both computer vision and DL methods. These methods have been developed to solve the complex task of extracting and identifying text in natural scenes, which poses several challenges due to varying lighting conditions, text orientations, fonts, and background clutter.

5.5.1 Text Detection

Text detection is the process of identifying regions in an image that contain text. Early approaches to text detection relied on traditional computer vision techniques, such as sliding windows, connected component analysis, and machine learning classifiers such as Support Vector Machines (SVM), Random Forests, and AdaBoost. These methods involved manually designed filters to detect candidate text regions based on features such as edges, contrast, and color. Although computationally efficient, these methods struggled with accuracy in complex scenes and text that varied in orientation or size.

With the advent of DL, modern text detection methods have significantly improved in accuracy and robustness. State-of-the-art techniques, such as Faster R-CNN, introduced by Ren et al. [65], and Connectionist Text Proposal Network (CTPN), developed by Tian et al. [66], rely on deep CNNs to propose regions that are likely to contain text. The Faster R-CNN method incorporates a region proposal network (RPN) to detect multi-orientated text, while CTPN combines CNN and RNN to enhance the detection of text in complex orientations.

One prominent text detection model is EAST (Efficient and Accurate Scene Text Detector), proposed by Zhou et al. [67]. EAST utilizes a fully CNN for efficient text detection by directly predicting text regions. EAST is known for its balance between accuracy and speed, making it well suited for real-time applications. However, despite these advances, the deployment of these methods on resource-constrained hardware poses significant challenges due to their computational complexity.

5.5.2 Text Recognition

Text recognition refers to the process of recognizing and converting detected text regions into readable format. Early recognition methods involved the use of hand-made features and machine learning algorithms to classify text. However, these methods suffered from limited accuracy, especially when faced with low resolution or distorted text.

Deep learning-based text recognition methods have since become the state-of-the-art, using convolutional and recurrent neural networks to recognize text in natural scenes. Jaderberg et al. [68] pioneered the use of deep CNN for word-level recognition, achieving significant improvements in accuracy. These models excel at learning complex text patterns directly from data, making them robust to variations in font, orientation, and background noise.

More recently, attention-based methods have been introduced to improve the accuracy of text recognition in challenging conditions. These methods, such as those proposed by Shi et al. [69], use sequence-to-sequence models with attention mechanisms to focus on relevant parts of the image during the recognition process.

Recent efforts have focused on addressing the resource constraints of embedded systems by developing lightweight models and optimization techniques. For example, lightweight text spotters such as the method introduced by Bagi et al. [70] have been designed to operate efficiently in resource-limited environments while still providing accurate text detection and recognition.

Quantization-aware training (QAT) is another approach that has been used to mitigate the negative effects of quantization on model accuracy. Adaptive bezier curve networks, such as ABCNet [71], further optimize the text recognition pipeline by reducing the number of parameters and operations required, making them more suitable for real-time applications on devices with limited resources.

Despite these advancements, there remains a need for more efficient methods that can handle the sensitivity of scene text recognition models to optimization techniques like quantization and pruning, particularly, on hardware supports integer-only operations.

5.6 Proposed Methodology

The focus of this methodology is to overcome the optimization limitations of scene text detection and recognition models when deployed on resource-constrained hardware, particularly integer-only devices. My approach achieves efficient model compression while preserving performance, adapted for embedded systems with limited resources. This section elaborates on the architecture of the system, the enhancements made to the base models, the quantization techniques, and the unique text recognition strategies developed.

5.6.1 System Architecture Overview

The overall system follows a two-stage architecture that begins with text detection, followed by text recognition. As shown in Figure 5, the detection model localizes text regions within the image, while the recognition model decodes the text contained within these regions.

I designed the system using a modified EAST detector for the detection stage and a novel contour-based extraction method for the recognition stage. Both models are heavily optimized to run efficiently on integer-only hardware, such as microcontrollers or edge TPUs, without compromising accuracy.

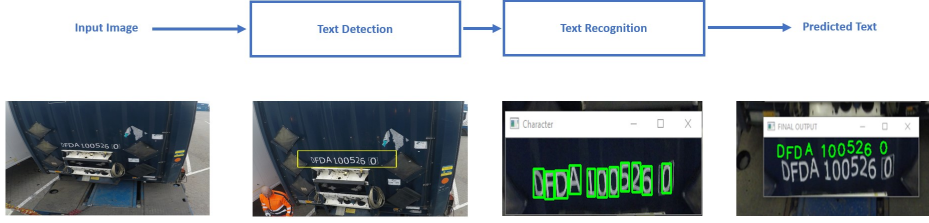


Figure 5: System Architecture for Scene Text Detection and Recognition. Appendix 3, Figure 1

5.6.2 Text Detection: Modified EAST Architecture

The text detection stage is built on the original EAST architecture, which performs direct prediction of text regions. Figure 6 illustrates the original EAST architecture, highlighting its three main stages: feature extraction, feature merging, and output generation.

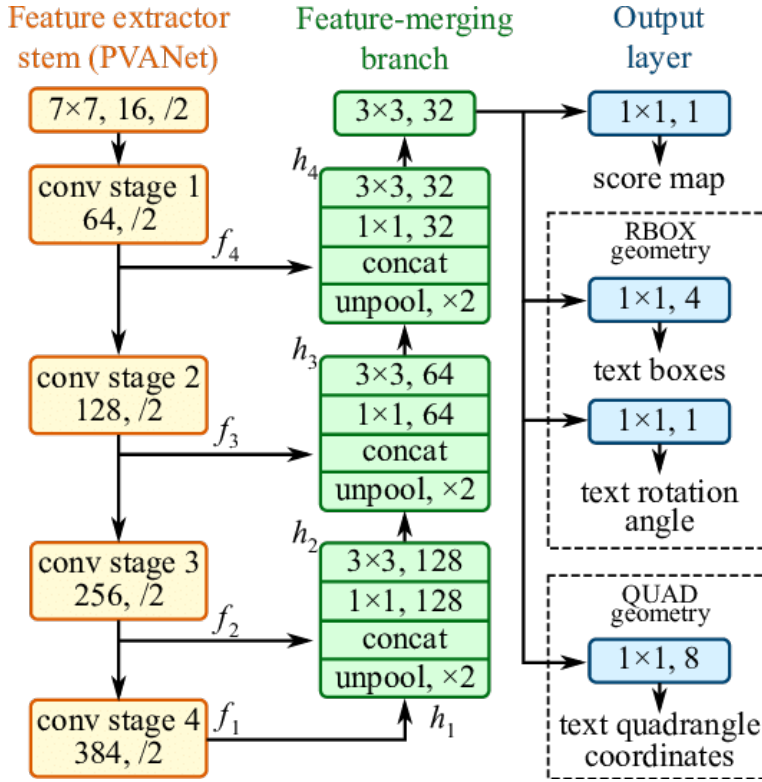


Figure 6: Original EAST Architecture for Scene Text Detection, [67]

To better suit the needs of resource-limited hardware, I introduced key modifications to the base EAST model. My approach replaces the original PVANet backbone with ResNet-50, which offers a more efficient trade-off between speed and accuracy. The ResNet-50 backbone, illustrated in Figure 7, uses a bottleneck design that reduces the number of operations, making it ideal for further compression for real-time embedded systems.

The modified EAST architecture, as shown in Figure 8, retains the three core stages of

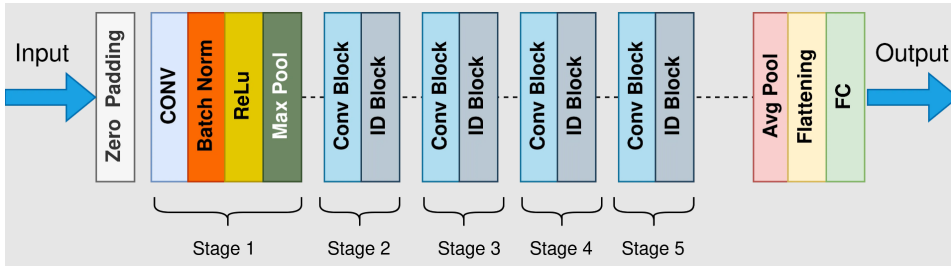


Figure 7: ResNet-50 Backbone Architecture. Appendix 3, Figure 4

the original design while incorporating ResNet-50 for feature extraction. This enhancement significantly reduces the model's size and computational footprint, making it more amenable to further optimization. This improvement is attributed to two key aspects of ResNet-50's design.

First, ResNet-50 employs a bottleneck architecture with 1x1 kernel filters, which reduces the number of matrix multiplications and network parameters, leading to faster propagation times. Second, it replaces fully connected layers with global average pooling, which effectively minimizes the overall size of the model. These features make ResNet-50 a more efficient backbone for the modified EAST architecture

The architecture is composed of three essential stages.

Feature extraction leverages ResNet-50, illustrated in Figure 7, as the backbone to extract high-level features from input images. It consists of five convolutional blocks, each designed to capture progressively more complex features from the input data.

Feature merging employs 1x1 and 3x3 convolutions to combine intermediate outputs from various stages of ResNet-50. This approach reduces computational complexity while retaining important feature information.

Output generation produces bounding boxes, shown in Figure 9, along with confidence scores for detected text regions. These outputs are then passed to the recognition model for further processing.

5.6.3 Quantization for Integer-Only Hardware

One of the critical challenges in deploying deep learning models on embedded devices is ensuring efficient computation on hardware that only supports integer arithmetic. To address this, I applied an 8-bit quantization technique to both the weights and activations of the models. This reduces the precision of parameters from 32-bit floating-point values to 8-bit integer values, drastically cutting memory usage and computational demands.

The quantization process followed a structured approach to minimize errors and maintain model performance.

A key innovation was the introduction of a quantization offset, or bias, to address small errors introduced by reduced precision in the integer-only representation. These errors, particularly significant in sensitive regions like bounding box coordinates in the text detection model, were mitigated by introducing an offset during ground-truth label generation. This adjustment compensated for quantization-induced errors during training, ensuring accuracy in regions most susceptible to small deviations.

Quantization-aware training (QAT) was employed to further enhance performance. By simulating quantization-induced errors during the training phase, the model adapted to lower precision values. This approach preserved accuracy even when static quantization was applied post-training, countering the typical performance degradation associated

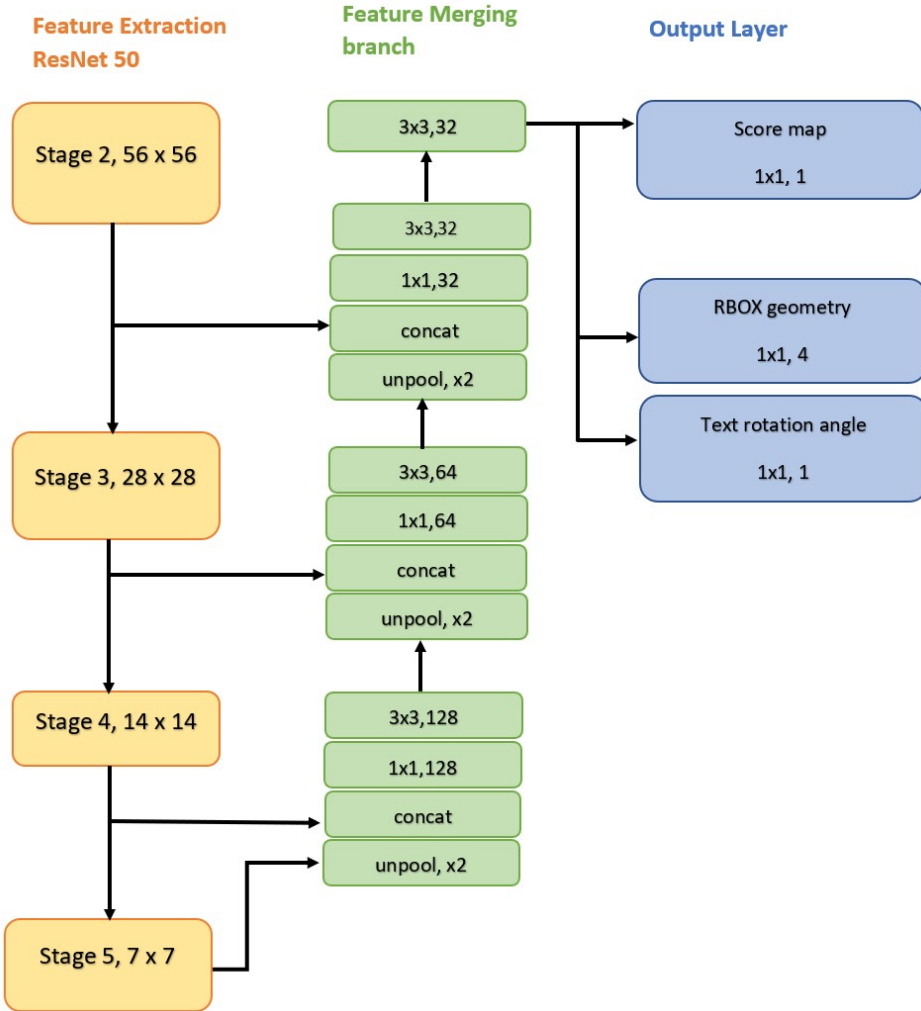


Figure 8: Modified EAST Architecture with ResNet-50 Backbone. Appendix 3, Figure 3

with static post-training quantization.

Finally, scaling and quantization were applied to convert 32-bit floating-point values into 8-bit integers using equations (1) and (2). This process, supported by the quantization offset, maintained robust bounding box accuracy, ensuring minimal deviation despite reduced precision. This step was critical in upholding the overall detection and recognition pipeline's reliability.

By combining QAT with the quantization offset, the final model achieves the necessary precision for accurate text detection.

5.6.4 Text Recognition: Contour-Based Character Extraction

After detecting the text regions, the next step was to extract and recognize individual characters within those regions, as illustrated in Figure 11. I developed a contour-based character extraction method, which improves recognition accuracy even in challenging scenarios like distorted or low image resolution text.



Figure 9: Bounding boxes generated by the text detection model. Appendix 3, Figure 5

The recognition pipeline is structured into three key stages to ensure accurate and efficient processing of detected text regions.

The first stage, pre-processing, involves cropping and normalizing the detected text regions based on the bounding-box coordinates provided by the detection model. This step ensures that only regions containing text are passed to the recognition pipeline, reducing computational overhead and eliminating noise from surrounding areas.

Next, contour-based character extraction is employed to isolate individual characters within the detected text regions. Contours are computed for each character, effectively separating them even in cases of close clustering or varying orientations. Irrelevant contours, such as those belonging to background objects, are discarded to prevent interference. This method significantly improves recognition accuracy by ensuring that only valid character candidates are processed.

Finally, the character recognition stage identifies specific characters using a CNN-based recognition model. The architecture, as detailed in Table 3, balances efficiency and accuracy. The model begins with convolutional layers that extract high-level features, capturing unique characteristics like edges and strokes essential for classification. Pooling layers follow, reducing the spatial dimensions of feature maps to enhance efficiency without losing critical information. Fully connected layers conclude the pipeline, mapping extracted features to character labels such as letters and numbers. This structured design ensures robust and accurate text recognition suitable for a variety of applications.

The text recognition model was quantized to support integer-only computations, ensuring both real-time performance and accuracy. The output of the character recognition model was aggregated string, representing the text recognized within the region of interest as seen in Figure 12.

By combining contour-based extraction with a lightweight CNN and quantization, the recognition model was able to deliver high accuracy in detecting and recognizing text. This method ensures that the system remains suitable for real-time embedded applications without the performance degradation typically associated with compressed or quantized scene text detection and recognition models.

The contour-based method has shown exceptional performance in addressing the challenges posed by the variability of real-world text. It effectively manages diverse text orientations, including horizontal and vertical layouts commonly seen in scene images.

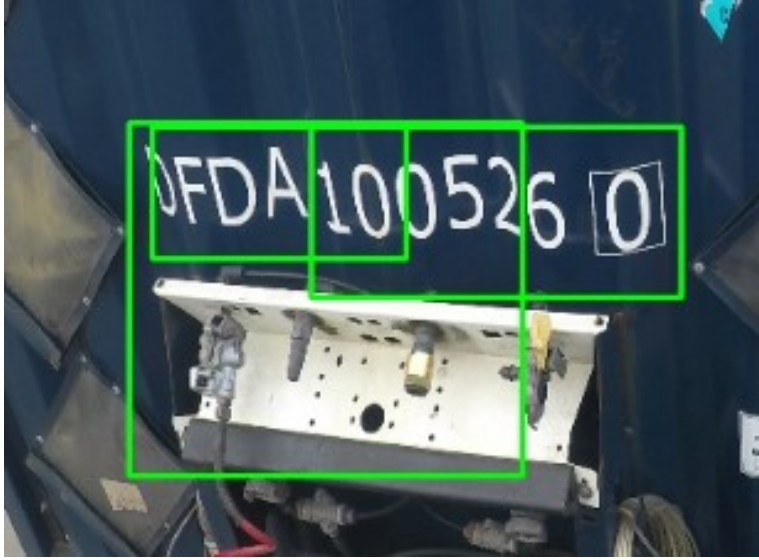


Figure 10: The bounding boxes generated without the introduction of quantization offset. Appendix 3, Figure 6

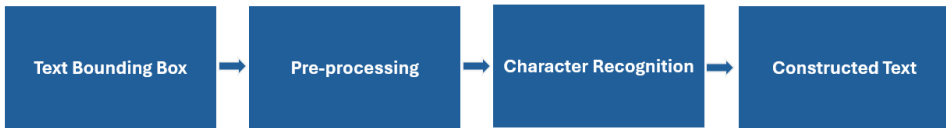


Figure 11: The text recognition pipeline. Appendix 3, Figure 8

Using deep feature extraction from convolutional layers, the method generalizes well across a wide range of fonts and styles, ensuring robust recognition irrespective of visual variations. In addition, it is adept at handling low-resolution or blurred text, making it particularly useful for applications where image quality fluctuates. This versatility underscores its suitability for complex real-world scenarios.

This approach was specifically designed to overcome the limitations of previous recognition models that struggled with resource constraints and real-world variability, ensuring that the model remains robust in different application scenarios.

5.7 Experiments

In this section, I present the experiments conducted to evaluate the performance of the proposed resource-aware scene text detection and recognition method, optimized for integer-only embedded hardware. The experiments include training and evaluation of both text detection and recognition models, followed by a comprehensive analysis of their performance in terms of model size, inference time, accuracy, computational cost, and peak memory footprint.

5.7.1 Experimental Setup

The experiments were carried out using a proprietary dataset. It comprises of 2000 images, with 1500 allocated for training the text detection model and 500 reserved for test-



Figure 12: Scene text detection and recognition results. Appendix 3, Figure 9

ing. Each image contains a shipping container with a unique cargo identification number, consisting of both letters and numbers. These container images were standardized to a size of 320×320 pixels to facilitate processing by the detection model.

For the text recognition model, 8750 character images were extracted. These images, resized to 64×64 pixels, included 35 characters, consisting of the digits 0-9 and the letters A-Z, excluding 'O' due to its visual similarity to digit 0. Each character class contained 250 images, ensuring a balanced dataset. From this set, 7000 images were used for training, while the remaining 1750 were reserved for testing.

To evaluate the models on embedded hardware that supports only integer operations, I selected the Google Coral Development Board as the target hardware platform. The Coral board is equipped with Quad Cortex A53 and Cortex M4F processors, along with a dedicated Edge TPU coprocessor. It includes 1 GB of RAM and 8 GB of flash memory, making it an ideal candidate for testing the performance of compressed models.

5.7.2 Evaluation Metrics

The performance of the quantized models was evaluated using these key metrics:

- **Model Size:** The size of the model in memory after quantization.
- **Inference Time:** The time required to process an input image and produce an output.
- **Computational Flop:** The total number of floating point operations during model inference.
- **Peak Memory footprint:** The peak memory usage during model inference.
- **Mean Loss:** A combination of dice loss and intersection-over-union (IoU) loss for the text detection model.
- **Accuracy:** It measures the predictive capacity of the text recognition model.

These metrics assess the trade-offs between model performance, memory footprint, and inference speed, which are critical in resource-constrained environments.

5.8 Results and Discussion

The results are presented in this section. The evaluation focuses on several key performance metrics that assess the applicability and efficiency of quantized models for both text detection and recognition tasks. The performance of both the original and quantized models was evaluated based on the size of the model, the inference time, the computational cost, and the maximum RAM usage.

Table 14: Evaluation metrics of the text detection and recognition models

	Text Detection		Text Recognition	
Metrics	Original	Quantized	Original	Quantized
Model Size (MB)	96.21	24.83	0.88	0.23
Inference Time (ms)	2356.00	1450.77	3.59	2.18
Computational Cost (MFLOP)	15072.50	0	20.20	0
Peak RAM Usage (MB)	286.23	40.63	5.04	3.29
	Mean Loss		Accuracy	
Model Performance (%)	25.51	26.23	99.73	99.62

As seen in Table 14, the quantized models significantly outperformed the original models in terms of resource efficiency. Quantized models achieved a reduction in model size. The size of the text detection model was reduced from 96.21 MB to 24.83 MB, while the size of the text recognition model decreased from 0.88 MB to 0.23 MB. This reduction highlights the efficiency of the quantization technique, making the models more suitable for deployment on resource-constrained devices like microcontrollers or TPUs.

A reduction in inference time was observed in both models after quantization. The text detection model's inference time improved from 2356.00 ms to 1450.77 ms, while the recognition model saw a reduction from 3.59 ms to 2.18 ms. This improvement is crucial for real-time applications as it ensures faster processing of text regions and overall system responsiveness.

The computational cost for both the quantized detection and recognition models was reduced to zero floating-point operations (FLOPs), demonstrating that the models were fully optimized to operate exclusively on integer-only hardware, thereby eliminating the need for any floating-point computations.

Quantized models demonstrated a reduction in maximum RAM usage. The RAM usage of the text detection model decreased from 286.23 MB to 40.63 MB, while the usage of the recognition model decreased from 5.04 MB to 3.29 MB. This reduction ensures that the models can run smoothly on devices with limited memory resources, improving overall system stability and efficiency.

In terms of performance, the mean loss and accuracy of the original and quantized models were compared. The results show that the quantized models maintained accuracy and performance, with only minimal deviations from the original models.

After quantization, the mean loss of the text detection model increased slightly from 25.51% to 26.23%, but this minor change did not significantly affect overall performance. The use of quantization bias and quantization-aware training (QAT) during training effectively mitigated quantization-induced errors, addressing a key factor contributing to the models' sensitivity to compression.

In terms of text recognition accuracy, the quantized model performed remarkably well, maintaining an accuracy of 99.62%, compared to 99.73% in the original model. The min-

imal difference of 0.11% shows that the quantized model is highly reliable, making it a suitable option for real-time applications without significant performance degradation.

5.9 Summary and Conclusion

5.9.1 Summary

This chapter addresses the research question (RQ3): *"Can efficient model optimization be achieved without significantly compromising model performance, particularly for network architectures that are very sensitive to compression?"*. The focus of this case study was on scene text detection and recognition models, which are known for their sensitivity to quantization. The primary objective was to develop a method for efficient optimization while maintaining acceptable model performance, especially when the models are deployed on integer-only hardware.

The results demonstrate that the quantization techniques introduced was effective. The quantized text detection model saw a reduction in size from 96.21 MB to 24.83 MB, and the inference time improved from 2356 ms to 1450.77 ms. Similarly, the size of the quantized text recognition model decreased from 0.88 MB to 0.23 MB, with the inference time dropping from 3.59 to 2.18 ms. Despite these optimizations, the accuracy of both models remained largely unaffected, with only minimal reductions observed.

By reducing the computational cost to zero for quantized models due to integer-based operations, the study confirms that efficient model optimization can indeed be achieved without a significant compromise in performance. These findings highlight the viability of model compression and quantization techniques for use in scene text processing.

5.9.2 Conclusion

The results clearly demonstrate that the applied quantization technique provides a substantial improvement in resource efficiency without significantly sacrificing performance. The quantized models exhibit reduced memory and computational requirements, making them ideal for deployment in embedded systems with constrained resources, such as microcontrollers and edge TPUs.

Despite the minimal increase in mean loss, the overall performance of the quantized models remains highly competitive with the original models. The slight decrease in accuracy in the recognition model is negligible in practical applications and is outweighed by the significant improvements in inference time, model size, and RAM usage.

The combination of quantization-aware training (QAT) and the introduction of a quantization offset successfully mitigated most of the adverse effects typically associated with quantization of scene text models. This approach proves effective for optimizing scene text detection and recognition models.

6 CONCLUSION

This thesis has addressed the research gaps described through three key research objectives (RO1, RO2, and RO3), each corresponding to the research questions (RQ1, RQ2, and RQ3). The contributions made towards efficient model compression for resource-constrained hardware have been validated through peer-reviewed publications and conference presentations—detailed in **Appendix I, 2, and 3**. The cumulative findings and contributions are extensively discussed throughout this thesis.

Research Question 1 (RQ1): *Can a numerically based mathematical method be developed to quantitatively evaluate state-of-the-art deep learning (DL) model compression techniques for diverse application requirements?*

Research Objective 1 (RO1) focused on developing a robust numerically based mathematical method to objectively evaluate and rank state-of-the-art model compression techniques. This method, detailed in **Publication I (Appendix I)**, utilized a weighted score-based ranking system that assessed the efficiency, effectiveness, and suitability of each compression technique across various applications. The proposed method provides a standardized and adaptable approach for objectively selecting optimal compression strategies tailored to specific use cases, ensuring that the most effective methods are employed.

Research Question 2 (RQ2): *“Can a deep model compression ratio of 32x be achieved while maintaining an acceptable level of accuracy for practical deployment?”*

Research Objective 2 (RO2) aimed to achieve a deep model compression ratio of 32x while maintaining acceptable levels of accuracy. This objective explored the practical limits of extreme compression for deep learning models, particularly in critical edge applications. The ensemble method developed in **Publication II (Appendix 2)**, which combines Tensor Train Decomposition (TTD) and 8-bit quantization, successfully achieved a remarkable 57x compression ratio, far exceeding the initial target. This demonstrates the viability of high-ratio compression techniques for practical deployment without significant performance loss.

Research Question 3 (RQ3): *“Can efficient model optimization be achieved without significantly compromising performance, especially for network architectures that are highly sensitive to compression and quantization?”*

Research Objective 3 (RO3) addressed the challenge of optimizing deep learning models that are highly sensitive to compression and quantization. The goal was to significantly reduce computational and memory requirements while preserving model performance. **Publication III (Appendix 3)** introduced a novel quantization offset technique, known as quantization bias, which effectively mitigated the sensitivity of state-of-the-art scene text detection and recognition models to quantization. This innovation enabled efficient deployment on hardware that is only integer with minimal accuracy degradation. In addition, a comprehensive text orientation detection module was integrated, improving the ability of the model to process text in various orientations, thus broadening its applicability.

In conclusion, the research presented in this thesis has made significant advancements in the field of efficient deep learning (DL) model compression and optimization. By addressing the research questions described here, this thesis has developed innovative methods to evaluate, compress, and optimize models, providing valuable insights and practical solutions to implement high performance DL models on resources-constrained devices. The proposed methods not only push the boundaries of model compression, but also ensure that even sensitive architectures can be efficiently deployed without compromising performance, making them well suited for a wide range of applications.

List of Figures

1	A typical CNN architecture with a normalized (5×5) input image convolved with a normalized filter (2×2 kernel) showing convolution, pooling, and weights matrix multiplication operations in a deep learning network. This low-level abstraction shows the internal computation performed on the network parameters (i.e., the input tensor (5×5 image), weights, and activations. Appendix 1, Figure 4.	23
2	The teacher-student model based on a temperature-based softmax function. [42].	27
3	An illustration of the trainable pipeline for model compression using the ensemble of Tensor Train Decomposition (TTD) and 8-bit quantization. The pipeline begins with a trained base model, followed by TTD applied to dense layers, and concludes with 8-bit quantization of the resulting tensor trained model. Appendix 2, Figure 3	52
4	A block representation of the end-to-end trainable pipeline for model compression using an ensemble of Tensor Train Decomposition (TTD) and 8-bit quantization. The pipeline first applies TTD to compress the dense layers, followed by 8-bit quantization to further reduce the model size and computational cost. Appendix 2, Figure 4.	54
5	System Architecture for Scene Text Detection and Recognition. Appendix 3, Figure 1	63
6	Original EAST Architecture for Scene Text Detection, [67]	63
7	ResNet-50 Backbone Architecture. Appendix 3, Figure 4	64
8	Modified EAST Architecture with ResNet-50 Backbone. Appendix 3, Figure 3.	65
9	Bounding boxes generated by the text detection model. Appendix 3, Figure 5	66
10	The bounding boxes generated without the introduction of quantization offset. Appendix 3, Figure 6	67
11	The text recognition pipeline. Appendix 3, Figure 8	67
12	Scene text detection and recognition results. Appendix 3, Figure 9	68

List of Tables

1 Summary of key literature on various model compression methods, describing their contributions and challenges. 19

2 A table showing a summary of all the layers of the baseline model architecture. 43

3 A table showing a summary of all the layers of the compact model architecture. 44

4 Summary of Weight Assignments for Different Optimization Goals 46

5 Benchmark Model Metrics 46

6 Raw Metric Values. 46

7 Scaled Metric Values. 47

8 Scored Metric Values. 47

9 Weighted Score of the Performance-Based Weight Set Profile. 47

10 Rank of the Performance-Based Weight Set Profile. 48

11 Tensor Train Layer Configurations. 55

12 Performance comparison of tensor trained compressed models with different configurations 56

13 This table compares the baseline model with TTD, 8-bit quantization, and the ensemble method (QuanTT) 57

14 Evaluation metrics of the text detection and recognition models 69

References

- [1] V. S. Marco, B. Taylor, Z. Wang, and Y. Elkhatib, "Optimizing deep learning inference on embedded systems through adaptive model selection," *ACM Trans. Embed. Comput. Syst.*, vol. 19, feb 2020.
- [2] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, (New York, NY, USA), p. 123–136, Association for Computing Machinery, 2016.
- [3] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '18, (New York, NY, USA), pp. 389—400, Association for Computing Machinery, 2018.
- [4] L. Hedegaard, "Efficient online processing with deep neural networks," in *Efficient Processing of Deep Neural Networks*, pp. 145–282, Cham: Springer Nature Switzerland, 2020.
- [5] X. Zhang, Y. Chen, C. Hao, S. Huang, Y. Li, and D. Chen, *Compilation and Optimizations for Efficient Machine Learning on Embedded Systems*, pp. 37–74. Cham: Springer Nature Switzerland, 2024.
- [6] H. Cai, J. Lin, Y. Lin, Z. Liu, H. Tang, H. Wang, L. Zhu, and S. Han, "Enable deep learning on mobile devices: Methods, systems, and applications," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 27, mar 2022.
- [7] T. P. Nagarhalli, V. Vaze, and N. K. Rana, "Impact of machine learning in natural language processing: A review," in *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, pp. 1529–1534, 2021.
- [8] S. Sengupta, S. Basak, P. Saikia, S. Paul, V. Tsalavoutis, F. Atiah, V. Ravi, and A. Peters, "A review of deep learning with special emphasis on architectures, applications and recent trends," *Knowledge-Based Systems*, vol. 194, p. 105596, 2020.
- [9] X. Wang, Y. Zhao, and F. Pourpanah, "Recent advances in deep learning," *International Journal of Machine Learning and Cybernetics*, vol. 11, no. 4, pp. 747–750, 2020.
- [10] X. Zhai, X. Chu, C. S. Chai, M. S. Y. Jong, A. Istenic, M. Spector, J.-B. Liu, J. Yuan, Y. Li, and N. Cai, "A review of artificial intelligence (ai) in education from 2010 to 2020," *Complex.*, vol. 2021, jan 2021.
- [11] T. Panch, P. Szolovits, and R. Atun, "Artificial intelligence, machine learning and health systems," *Journal of Global Health*, vol. 8, p. 020303, Dec 2018.
- [12] O. A. Ademola, E. Petlenkov, and M. Leier, "Resource-aware scene text recognition using learned features, quantization, and contour-based character extraction," *IEEE Access*, vol. 11, pp. 56865–56874, 2023.

- [13] O. A. Ademola, E. Petlenkov, and M. Leier, "Ensemble of tensor train decomposition and quantization methods for deep learning model compression," in *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6, 2022.
- [14] Q. Wu, Y. Liu, Q. Li, S. Jin, and F. Li, "The application of deep learning in computer vision," in *2017 Chinese Automation Congress (CAC)*, pp. 6522–6527, IEEE, 2017.
- [15] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Comput.*, vol. 29, pp. 2352–2449, Sep 2017.
- [16] S. Alshahrani and E. Kapetanios, "Are deep learning approaches suitable for natural language processing?," in *Natural Language Processing and Information Systems* (E. Métais, F. Meziane, M. Saraee, V. Sugumaran, and S. Vadera, eds.), vol. 9612 of *Lecture Notes in Computer Science*, (Cham), Springer, 2016.
- [17] S. Pattanayak, "Natural language processing using recurrent neural networks," in *Pro Deep Learning with TensorFlow*, Berkeley, CA: Apress, 2017.
- [18] J. Hirschberg and C. D. Manning, "Advances in natural language processing," *Science*, vol. 349, pp. 261–266, 2015.
- [19] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.
- [20] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in Bioinformatics*, vol. 19, pp. 1236–1246, 05 2017.
- [21] C. Cao, F. Liu, H. Tan, D. Song, W. Shu, W. Li, Y. Zhou, X. Bo, and Z. Xie, "Deep learning and its applications in biomedicine," *Genomics, Proteomics & Bioinformatics*, vol. 16, no. 1, pp. 17–32, 2018.
- [22] K. Shameer, K. W. Johnson, B. S. Glicksberg, J. T. Dudley, and P. P. Sengupta, "Machine learning in cardiovascular medicine: are we there yet?," *Heart*, vol. 104, no. 14, pp. 1156–1164, 2018.
- [23] X. M. Liu, L. M. Faes, A. U. M. Kale, S. K. B. Wagner, D. J. P. Fu, A. M. Bruynseels, and et al., "Comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: a systematic review and meta-analysis," *Published in Open Access*, September 25 2019.
- [24] M. Sendak, M. Gao, M. Nichols, A. Lin, and S. Balu, "Machine learning in health care: A critical appraisal of challenges and opportunities," *eGEMs (Generating Evidence & Methods to improve patient outcomes)*, vol. 7, no. 1, p. 1, 2019.
- [25] E. Chong, C. Han, and F. C. Park, "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies," *Expert Systems with Applications*, vol. 83, pp. 187–205, 2017.
- [26] S. Sohanger, D. Wang, A. Pomeranets, and et al., "Big data: Deep learning for financial sentiment analysis," *J Big Data*, vol. 5, no. 3, 2018.
- [27] A. M. Ozbayoglu, M. U. Gudelek, and O. B. Sezer, "Deep learning for financial applications : A survey," *Applied Soft Computing*, vol. 93, p. 106384, 2020.

- [28] K. Polat, D. Lv, S. Yuan, M. Li, and Y. Xiang, "An empirical study of machine learning algorithms for stock daily trading strategy," *Mathematical Problems in Engineering*, vol. 2019, p. 7816154, 04 2019.
- [29] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Comput. Surv.*, vol. 52, feb 2019.
- [30] B. Dellal-Hedjazi and Z. Alimazighi, "Deep learning for recommendation systems," in *2020 6th IEEE Congress on Information Science and Technology (CiSt)*, pp. 90–97, 2020.
- [31] J. Liu and C. Wu, *Deep Learning Based Recommendation: A Survey*, vol. 424 of *Lecture Notes in Electrical Engineering*. Singapore: Springer, 2017.
- [32] N. Anantrasirichai and D. Bull, "Artificial intelligence in the creative industries: a review," *Artif Intell Rev*, vol. 55, pp. 589–656, 2022.
- [33] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '18*, (New York, NY, USA), p. 974–983, Association for Computing Machinery, 2018.
- [34] A. Mujtaba, W.-K. Lee, and S. O. Hwang, "Low latency implementations of cnn for resource-constrained iot devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 12, pp. 5124–5128, 2022.
- [35] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.
- [36] 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 *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, IEEE, 2018.
- [37] R. Banner, Y. Nahshan, and D. Soudry, *Post training 4-bit quantization of convolutional networks for rapid-deployment*. Red Hook, NY, USA: Curran Associates Inc., 2019.
- [38] B. Liu, F. Li, X. Wang, B. Zhang, and J. Yan, "Ternary weight networks," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5, 2023.
- [39] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: training deep neural networks with binary weights during propagations," in *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2, NIPS'15*, (Cambridge, MA, USA), p. 3123–3131, MIT Press, 2015.
- [40] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), (Cham), pp. 525–542, Springer International Publishing, 2016.
- [41] Y. Zhu, B. Li, Y. Xin, and L. Xu, "Addressing representation collapse in vector quantized models with one linear layer," *ArXiv*, vol. abs/2411.02038, 2024.

- [42] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *ArXiv*, vol. abs/1503.02531, 2015.
- [43] L. Sun, J. Gou, B. Yu, L. Du, and D. Tao, "Collaborative teacher-student learning via multiple knowledge transfer," *Pattern Recognition*, vol. 118, p. 108004, 2021.
- [44] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, "Be your own teacher: Improve the performance of convolutional neural networks via self distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3713–3722, IEEE, 2020.
- [45] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. S. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *CoRR*, vol. abs/1412.6553, 2014.
- [46] 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," *CoRR*, vol. abs/1511.06530, 2015.
- [47] A. Novikov, P. Izmailov, V. Khurlov, M. Figurnov, and I. Oseledets, "Tensor train decomposition on tensorflow (t3f)," *Journal of Machine Learning Research*, vol. 20, no. 84, pp. 1–7, 2019.
- [48] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems* (D. Touretzky, ed.), vol. 2, Morgan-Kaufmann, 1989.
- [49] B. Hassibi, D. G. Stork, T. Watanabe, and G. Wolff, "Optimal brain surgeon: Extensions and performance comparisons," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 6, pp. 263–270, MIT Press, 1994.
- [50] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations (ICLR 2016), San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.
- [51] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *International Conference on Learning Representations*, 2017.
- [52] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019.
- [53] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Sparsity-aware deep learning accelerator design supporting cnn and lstm models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 799–812, 2020.
- [54] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l0 regularization," in *International Conference on Learning Representations (ICLR)*, 2017.
- [55] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *ArXiv*, vol. abs/1602.07360, 2016.

- [56] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.
- [57] 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*, vol. abs/1704.04861, 2017.
- [58] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 6848–6856, IEEE Computer Society, June 2018.
- [59] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *ArXiv*, vol. abs/1905.11946, 2019.
- [60] D. Barry, M. Shah, M. Keijsers, H. Khan, and B. Hopman, "xyolo: A model for real-time object detection in humanoid soccer on low-end hardware," *2019 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pp. 1–6, 2019.
- [61] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," in *International Conference on Learning Representations (ICLR)*, 2017.
- [62] H. Li, P. Li, Y. Liu, L. Zhang, and X. Hou, "Dynamic knowledge distillation for efficient network compression," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.
- [63] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Hr-nas: Searching efficient high-resolution neural architectures with lightweight transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10767–10776, IEEE, 2020.
- [64] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019.
- [65] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, Jun. 2015.
- [66] Z. Tian, W. Huang, T. He, P. He, and Y. Qiao, "Detecting text in natural image with connectionist text proposal network," in *Proceedings of the European Conference on Computer Vision (ECCV)*, (Amsterdam, The Netherlands), pp. 56–72, 2016.
- [67] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: An efficient and accurate scene text detector," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2642–2651, Jul. 2017.
- [68] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Reading text in the wild with convolutional neural networks," *International Journal of Computer Vision*, vol. 116, no. 1, pp. 1–20, 2016.

- [69] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 2298–2304, Nov. 2017.
- [70] R. Bagi, T. Dutta, and H. P. Gupta, "Cluttered textspotter: An end-to-end trainable light-weight scene text spotter for cluttered environments," *IEEE Access*, vol. 8, pp. 111433–111447, 2020.
- [71] Y. Liu, C. Shen, L. Jin, T. He, P. Chen, C. Liu, and H. Chen, "Abcnet v2: Adaptive bezier-curve network for real-time end-to-end text spotting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 8048–8064, Nov. 2022.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisors, Professor Eduard Petlenkov and Dr. Mairo Leier. Professor Petlenkov, your unwavering guidance, insightful feedback, and immense support have been instrumental in shaping my research journey. Your dedication to academic excellence and your belief in my abilities have continually inspired me to push beyond my limits. Dr. Leier, your support and constant encouragement have greatly influenced the quality of this work. You have both been incredible mentors, and I am deeply grateful for the opportunity to learn under your supervision.

To my family, your love and unwavering support have been the foundation upon which this thesis rests. My mother, Mrs. Olutosin Adenike, your prayers, encouragement, and boundless faith in me have given me the strength to persevere through the most challenging moments. To my elder brother, Tolulope, and my younger siblings, Michael and Tobiloba, thank you for your constant love, understanding, and patience. Your belief in me has been a beacon of hope and motivation throughout this journey.

To my fiancée, Fatima, your love, patience, and understanding have been my rock during this demanding period. Your presence has brought balance and joy into my life, and I am deeply grateful for your unwavering support and encouragement. Thank you for standing by me through the highs and lows, for your constant cheerleading, and for believing in my dreams as much as I do.

To my friends, Dr. Stephen, Dr. Nafisat, Tomisin, and Richard, thank you for your friendship, laughter, and for being there whenever I needed a listening ear or a word of encouragement. Your support has made this journey more bearable, and I am thankful for the memories we have shared.

Finally, I would like to extend my sincere gratitude and appreciation to the organizations that provided financial support during my PhD studies. This was my dream, but these organizations made it a reality.

- Estonian Research Council through the Institutional Research Project PRG620 and PUT1435
- TTU Development Program 2016-2022, Project Code 2014-2020.4.01.16-0032
- Study IT in Estonia Programme
- ICT Doctoral School at Tallinn University of Technology
- DoRa Programme
- TALTECH ERASMUS CHARTER FOR HIGHER EDUCATION 2021-2027

This thesis is not just a reflection of my efforts but also a testament to the love, support, and guidance of all the amazing people around me. I am truly blessed to have each of you in my life. Thank you all, from the bottom of my heart.




Appendix 1

I

O. A. Ademola, M. Leier, and E. Petlenkov, "Evaluation of deep neural network compression methods for edge devices using weighted score-based ranking scheme," in *MDPI, Sensors*, 21(22):7529, 2021

Article

Evaluation of Deep Neural Network Compression Methods for Edge Devices Using Weighted Score-Based Ranking Scheme

Olutosin Ajibola Ademola ^{1,*} , Mairo Leier ^{1,†}  and Eduard Petlenkov ^{2,†} 

¹ Embedded AI Research Laboratory, Department of Computer Systems, Tallinn University of Technology, Ehitajate tee 5, 19086 Tallinn, Estonia; mairo.leier@taltech.ee

² Centre for Intelligent Systems, Department of Computer Systems, Tallinn University of Technology, Ehitajate tee 5, 19086 Tallinn, Estonia; eduard.petlenkov@taltech.ee

* Correspondence: olutosin.ademola@taltech.ee

† These authors contributed equally to this work.

Abstract: The demand for object detection capability in edge computing systems has surged. As such, the need for lightweight Convolutional Neural Network (CNN)-based object detection models has become a focal point. Current models are large in memory and deployment in edge devices is demanding. This shows that the models need to be optimized for the hardware without performance degradation. There exist several model compression methods; however, determining the most efficient method is of major concern. Our goal was to rank the performance of these methods using our application as a case study. We aimed to develop a real-time vehicle tracking system for cargo ships. To address this, we developed a weighted score-based ranking scheme that utilizes the model performance metrics. We demonstrated the effectiveness of this method by applying it on the baseline, compressed, and micro-CNN models trained on our dataset. The result showed that quantization is the most efficient compression method for the application, having the highest rank, with an average weighted score of 9.00, followed by binarization, having an average weighted score of 8.07. Our proposed method is extendable and can be used as a framework for the selection of suitable model compression methods for edge devices in different applications.

Keywords: deep neural network compression; compression method evaluation; weighted score-based ranking; embedded deep learning; edge computing



Citation: Ademola, O.A.; Leier, M.; Petlenkov, E. Evaluation of Deep Neural Network Compression Methods for Edge Devices Using Weighted Score-Based Ranking Scheme. *Sensors* **2021**, *21*, 7529. <https://doi.org/10.3390/s21227529>

Academic Editor: Alex Alexandridis

Received: 13 September 2021

Accepted: 5 November 2021

Published: 12 November 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In deep learning, object classification tasks are solved using Convolutional Neural Networks (CNNs). CNNs are variants of Deep Neural Network (DNN) architectures that accept batches of images as input and return the probability vectors of all the possible outcomes [1]. These architectures are used as the backbone of state-of-the-art DNN-based object detection methods. R-CNN [2] was one of the most successful methods proposed to solve object classification, localization, and segmentation problems. R-CNN used AlexNet (a variant of the CNN architecture developed in [1], having over 62M trainable parameters and requiring a storage size of 250MB) as the backbone of the network. Other CNN architectures used as the backbone of object detection models are ResNet-50 [3], which requires over 95MB of storage space, and VGG16 [4].

Recent works have shown that microarchitectures (e.g., SqueezeNet [5], ShuffleNet [6], EfficientNet [7], MobileNet [8]) with fewer parameters and small model sizes can achieve the same level of accuracy as the macroarchitectures (e.g., Inception [9], AlexNet [1], ResNet-50 [3], VGG16 [4]).

Modern object detection methods have shown excellent results in terms of accuracy and generalization. This is due to the complexity of the networks used as the models' backbone. This complexity hinders their applications on edge computing devices that are usually liable to computational power and memory constraints. To deploy such models on

these pieces of hardware while maintaining the performance (i.e., accuracy, robustness), it is necessary to optimize the models efficiently.

Different studies have proposed different compression methods, such as bit reduction, low-rank matrix decomposition, network pruning, sparsity, domain residual, knowledge distillation, etc. These methods have shown excellent results in terms of model size reduction, fast inference, and computational efficiency, without a significant decrease in accuracy when compared with the original model [5,10]. However, identifying the most effective and efficient methods based on the application requirements is challenging.

In this work, we propose a weighted score-based ranking scheme to address this problem. Our proposed scheme utilizes the performance metrics of the compressed models to evaluate and rank the compression methods. We show the effectiveness of this scheme by applying it on the baseline, compressed, and micro-CNN models trained on our dataset.

2. Project Description

2.1. Project Background

The research projects ROROGREEN and Smart Car Deck Solution (SCDS) are initiatives developed by DFDS, Denmark (Europe's largest Ferry operator), and Tallink AS, Estonia (an Estonian based shipping company), respectively. These projects set out to develop an automatic vehicle detection (classification and localization), positioning, and tracking system for the cargo ships operated by both companies. The projects aim to digitize, automate, and optimize the end-to-end process of vessel stowage, loading/discharge of cargo units, and terminal operations. The actualization of these projects is crucial because it is envisioned that other shipping companies in Europe will also benefit from the projects' results in the future.

2.2. Project Requirements

The ROROGREEN and SCDS projects require a real-time automated solution that can track and monitor different objects on the decks of a cargo ship. Our goal is to achieve this using a cost-effective camera-based edge device that is capable of processing the images from the camera locally and in real-time. Our proposed solution is based on modern CNN-based object detection methods, which will be optimized for the proposed hardware. For the evaluation of the optimization methods implemented in this work, we used a portion of the entire dataset that consisted of 1400 images of different categories gathered locally. The sample of the dataset is described in Figure 1.



Figure 1. Sample of objects, including a person class, to be detected in the ROROGREEN and SCDS projects. There are eleven images and each corresponds to the respective target class.

To fulfil the goal of the projects, we proposed a solution that leverages modern object detection methods. These methods serve as the backbone for the other functionalities of the system (e.g., tracking, positioning, monitoring). We extracted the requirements of the projects from the projects' goals, use cases, and areas of use. We found that the requirements of both projects are quite similar. The ROROGREEN project aims to classify, localize, and track two classes of vehicles according to DFDS vehicle grouping. The SCDS project aims to detect and track ten classes of vehicles with people inclusive, which makes it a total of eleven classes. It is worth noting that DFDS's vehicle classes are also embedded in Tallink's vehicle categories.

The base networks of modern CNN-based object detection methods are usually deep. These usually require hardware having high processing power, high memory (flash size), and high RAM size. These requirements are usually lacking in resource-constrained devices; as such, deploying the models without optimization is impossible. Therefore, model compression is mandatory for efficient computation and storage, which, in turn, reduces the inference time without a significant drop in model accuracy. This will also minimize the overall power consumption of the system.

As discussed above, it is evident that our goal is to develop a real-time, low-power, cost-effective, and efficient solution using camera-based embedded hardware. This implies that the trained custom model must be optimized to meet the system/hardware requirements as highlighted below.

- High accuracy;
- Small model size;
- Small peak memory footprint;
- Fast inference;
- Computational efficiency.

3. Problem Statement

The ROROGREEN and SCDS projects aim to automate the loading and offloading operations of the cargo units by using the real-time data generated (e.g., vehicle type, object bounding coordinates, lane position, and other application parameters) by the camera-based embedded hardware installed on each deck. The generated data are processed by the external server, which automatically generates the loading/offloading plan, which is validated with the standard deck plan.

Predicting the vehicle type, bounding box coordinates, and position of each object requires modern object detection and position estimation algorithms. These methods must be energy-efficient, less computationally intensive, and accurate. This is quite onerous because of the choice of hardware required. The contrived visual representation of a ship deck containing different vehicle types on respective lanes is described in Figure 2.

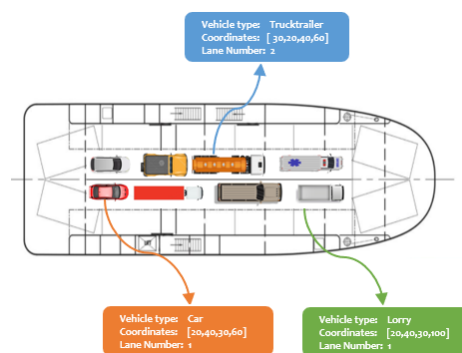


Figure 2. A contrived deck plan showing the vehicle loading operation. Each vehicle type occupies different regions and the respective lane position.

We proposed a CNN-based object detection method. This was adopted due to the remarkable results it has shown. Modern object detection models are quite large in depth and sometimes width. The depth is usually characterized by the number of hidden layers in the backbone of the CNN architectures used. The depth contributes to the total number of parameters (i.e., the weights in each layer), which usually result in large models, as described in Figure 3. These models are power-inefficient, computationally expensive, and require large memory for storage. This cannot be processed by our low-power camera-based edge device. As such, the models need to be optimized/compressed for the hardware.

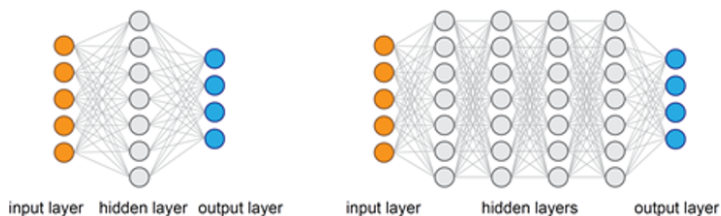


Figure 3. A visual representation of a deep neural network (right) and a shallow neural network (left).

There are existing compression methods for large CNN models' compression. Each method has its drawbacks and determining the most suitable and effective method is of major concern. To properly evaluate the effectiveness of each method, it is important to understand how the method affects the original models. Therefore, we can rank all the methods based on the following key parameters: model size, accuracy, memory footprint, computational cost, and inference time.

4. Related Work

CNN models have shown unprecedented results in solving problems related to computer vision (e.g., image classification, object detection, tracking). Due to the high computational power and memory that are required, this has impeded their adoption in embedded applications. This problem led to a new area of research (i.e., deep neural network compression) to tackle such challenges.

Several methods for model compression have been proposed in different studies. These methods have shown remarkable results, albeit with certain drawbacks. In this section, we categorize the methods into five groups: bit reduction, knowledge distillation, tensor decomposition, network pruning, and microarchitecture.

4.1. Bit Reduction

Bit reduction techniques have been around for quite a while [11,12]. These techniques aim to reduce the size of the model without a significant loss in the model performance. In practice, this is somewhat difficult to achieve due to the loss of information when approximating the 32-bit full precision weights and activations to a fixed point integer representation [13,14]. Quantization can be implemented using (16, 8, or 4 bits); however, there can be extreme cases where 2 bits or 1 bit are used to represent the weights and/or activations. These are referred to as binarization and ternarization. Binary networks encode the weights and activations with 1 bit (−1, 1), in contrast to ternary, which uses 2 bits (−1, 0, 1) [15].

The works of [11,12,16,17] showed the possibility of training deep networks with low bit-width weights. The weights and activations of the networks were discretized and a drastic reduction in model size with an insignificant drop in accuracy was achieved in [14]. An adaptive quantization framework that achieved a 20–40% higher compression rate, in contrast to non-adaptive quantization methods that relied on a uniform quantizer, was proposed in [18]. A vector-based quantization method that reduces the reconstruction in the network output was introduced in [19]. The work of [18] also showed that the network

layers contribute differently to the model prediction result; therefore, it is logical not to use uniform bit-width quantization.

Quantization techniques have shown promising results for large model compression. This breakthrough has caused different industries developing on-device/edge-based artificial intelligence solutions to adopt the methods. It is worth noting that the lower the bit-width used in quantization, the higher the compression rate and the model sensitivity to accuracy degradation.

4.2. Knowledge Distillation

The idea of transferring the knowledge learnt by a large model to a small model is primordial. This has shown quite a reasonable result (i.e., accuracy). Ensembles (i.e., combining the predictions of several models) of large models were compressed using this method [20]. The authors used the parameters learned by the large, slow, and complex models to train an ensemble of small and fast models. This method gained traction from the results shown by the authors. The authors of [21] trained a single model by transferring the attributes of the ensemble of models to the single model. This achieved higher accuracy than the prediction of the individual model of the ensemble.

The transfer of knowledge from a large and accurate model does not guarantee that the small model will be accurate. It was demonstrated that not all the students (small models) can learn effectively from the teachers (large models) in [22]. The authors also pointed out that past work has not addressed this area but rather focused on the trend of the subject.

Knowledge distillation has proven to be very relevant in many applications. This is due to its simplicity and the ability to use synthetic data (data generated artificially) where real data are not readily available; however, the statistical attributes of the synthetic data must conform with the mimic real data [22].

4.3. Low-Rank Tensor Decomposition

Tensor decomposition is the generalization of low-rank matrix decomposition. Its use case has been extended to CNNs. CNN models are composed of different layers, which are defined by the types of mathematical operations performed in the layer. These layers include the convolutional layer (CL), activation layer (AL), fully connected layer (FCL), etc. A layer in the network is an array of nodes or neurons that can be expressed as a matrix or tensor (i.e., a generalized form of a matrix). Each node is a regression function that performs some computations (e.g., matrix multiplication) on higher-dimensional input data and a weights matrix. Matrix-based optimization techniques (e.g., singular value decomposition, eigendecomposition) can be applied to the convolutional and/or the fully connected layers [10,23] to reduce the number of parameters in these layers.

When a tensor (e.g., weights matrix) is factorized into its sub-components (i.e., sub-tensors or factors), all the sub-tensors do not contribute equally to the main tensor. This implies that the sub-tensors can be ranked based on the order of importance. An approximate tensor can be derived, which results in having a low rank in contrast to the original high-order tensor. Singular value decomposition (SVD) (the most commonly used matrix decomposition method) was used to compress the weights matrix in the fully connected layer to reduce the model size. A two-times increase was achieved in computation time by decomposing the weights in the convolutional layers in [24]. Kholiavchenko [25] proposed an iterative-based tensor decomposition technique and showed that a layer can be decomposed several times.

It is also worth pointing out that the compression ratio of the model when adopting this method is greatly dependent on the rank value (i.e., the most significant to the least significant). In the case of an extreme rank value, the model size will be drastically small and vice versa.

4.4. Network Pruning

It has been shown in several studies that deep neural networks are usually overparameterized. This is quite common in large and complex networks [26]. It affects the model convergence time and contributes to the computation and storage overheads. The goal of pruning is to reduce a large network to a smaller and faster network. This is possible because the parameters of the network do not contribute equally to the model output. As such, the level of contribution can be ranked by the order of importance and the less significant parameters can be pruned (i.e., set to zero). A penalty factor was introduced to the loss function to penalize the weights that do not contribute significantly to the network output, resulting in a smaller network, in [27]. A stochastic gradient descent (SGD) momentum-based pruning method for setting redundant weights to zero was proposed in [28].

The network parameters that can be pruned include the weights (connections), neurons (nodes), or convolutional filters (kernels). The choice of the parameters to be explored for pruning is dependent upon the application requirements (e.g., memory footprint, computational cost, bandwidth). A weights- or connections-only-based pruning technique is also referred to as unstructured pruning, in contrast to structured pruning, which involves the removal of low-rank neurons or filters [28]. Redundant weights were eliminated to compress the network size and compensate for the accuracy drop by retraining the network in [29].

Pruning is a very old compression method [30] and has shown its strength in the reduction of model size. The effectiveness varies with the techniques adopted (e.g., brute-force [30], penalty-factor [31], sensitivity error [32,33]). Network pruning usually results in an accuracy drop. When a network is over-pruned, it can render the network useless. This is why it is important to estimate the pruning threshold, evaluate the network after pruning, and retrain to compensate for the decrease in accuracy.

4.5. Microarchitecture

Microarchitecture is a concept for designing small and compact models. This method is based on the background information (i.e., residual knowledge) of the critical and most important blocks needed in the design of the CNN architecture. This is quite different from the other methods (i.e., pruning, quantization, binarization, tensor decomposition) because it does not rely on any external compression techniques that are usually applied to the CNN model either after or during training. It focuses on using domain knowledge to carefully design the network architecture. The same level of accuracy obtained by AlexNet [1] with 50-times fewer parameters was achieved in [5]. Before the term microarchitecture was standardized by these authors [5], smaller kernel sizes (i.e., smaller-sized convolutional filters) were used in practice, and this has shown significant improvements in terms of model performance (i.e., speed, accuracy) [34–36].

The research on the design of efficient and lightweight CNN models has increased as a result of the exponential growth in the demand for real-time, efficient, and power-consumption-aware embedded computer vision applications in diverse areas. A category of models called MobileNet was proposed in [8]; these CNN microarchitectures have become some of the state-of-the-art methods for image classification and object detection for low-power and resource-constrained devices. Other models in this category are SqueezeNet [5], ShuffleNet [6], EfficientNet [7], and TinyYOLO [37].

5. Compression Methods, Evaluation, and Ranking

Our goal is to evaluate and develop a novel method for ranking the performance of state-of-the-art techniques for compressing deep learning models. The performance of each technique will be ranked using the following five key metrics: model size, accuracy, peak runtime memory usage, computational cost, and inference time. In this section, we describe each compression method and its implementation.

5.1. Baseline Model Description

This section describes the architecture of our baseline model. This simple and lightweight network was developed to serve as the reference model for evaluating the different compression methods for this work. The input layer of the network takes an RGB image of shape (64, 64, 3) and (3, 3) kernel filters were used throughout the entire network, resulting in a total of 1,106,209 parameters. The choice of the convolutional filter size was inspired by the work of [34]. The network includes a stack of two sets of CONV2D, RELU, BN, and POOL layers, followed by a set of CONV, RELU, and BN. The final block is the dense block, which consists of FC, BN, FC, and SOFTMAX layers, as shown in Table 1.

Table 1. A table showing a summary of all the layers of the baseline model architecture.

Layer Type	Output Size	Parameters
CONV2D	(None, 64, 64, 32)	864
BN	(None, 64, 64, 32)	96
MAXPOOL2D	(None, 32, 32, 32)	0
CONV2D	(None, 32, 32, 64)	18,432
BN	(None, 32, 32, 64)	192
MAXPOOL2D	(None, 16, 16, 64)	0
CONV2D	(None, 16, 16, 64)	36,928
BN	(None, 16, 16, 64)	192
FLATTEN	(None, 16384)	0
DENSE	(None, 64)	1,048,576
BN	(None, 64)	192
DENSE1	(None, 11)	704
ACTIVATION	(None, 11)	0
TOTAL PARAMETERS		1,106,209

5.2. Compression Methods

CNNs play a vital role in deep learning methods for object detection. A CNN is composed of different layers. Each layer is made up of computational nodes (i.e., neurons) that process the input signals. The contribution made by each layer to the computation and memory requirements of the whole network is usually unevenly distributed. This is because of the different operations and parameters that are associated with each layer. The majority of the weights are in the dense layers (i.e., fully connected layer), but these account for a lesser percentage of the total floating-point operations. This implies that optimizing the FC layers alone will result in only a dramatic reduction in the model size, without a significant improvement in the overall speed, in contrast with optimizing the convolutional layers.

Several methods have been proposed for CNN model compression for resource-constrained devices. Each of these methods has its advantages and drawbacks. This makes it very challenging to identify the most appropriate, effective, and efficient compression method to adopt. The choice of the method is strictly dependent on the requirements of the application. In this section, we describe the compression methods that we considered in this work.

5.2.1. Quantization

Bit precision reduction is an important concept in mathematics that has been widely adopted in different applications, including deep neural network compression. Quantization limits the width of the bit that is used to represent a digit or number. The bit width

of the operands controls the precision level of the output when mathematical operations are performed.

The most common types of operations that are performed in CNNs are convolution operation, bias addition, and the dot product of the weights matrix and float input tensor, as described in Figure 4. These operations are computed in a 32-bit full precision floating point. Quantization aims to replace the 32-bit floating-point operations with low-precision number formats such as 1 bit, 4 bit, 8 bit, or 16 bit. Binarization transforms full precision models into a single-bit model (the weights and activations are encoded using 1 bit). Binarization can be described as an extreme case of quantization where the weights and activations are encoded using 4 bit, 8 bit, or 16 bit.

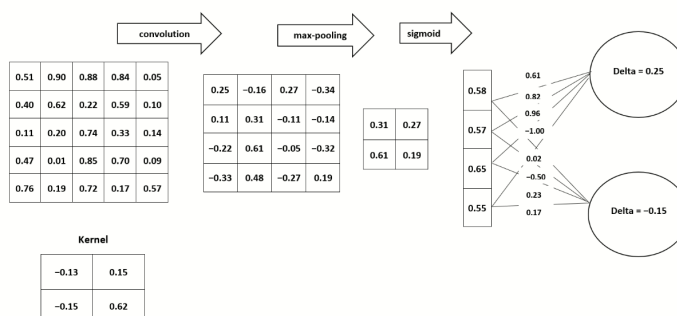


Figure 4. A CNN architecture with a normalized (5 × 5) input image convolved with a normalized filter (2 × 2 kernel) showing the three most common operations (convolution, pooling, and weights matrix multiplication operations) in a CNN. This low-level abstraction shows the internal computation performed on the network parameters (i.e., the input tensor (5 × 5 image), weights, and activations).

A model can be quantized either during training (Bit-Reduction-Aware Training, also called Quantization-Aware Training) or after training (Post-Quantization). The latter often results in a significant decrease in accuracy. However, this can be mitigated by retraining the network to compensate for the decrease in accuracy as a result of the error induced during the quantization operation.

We quantized the weights and activations of the baseline model using a symmetric mode 8-bit signed full integer quantizer implemented in Keras [38], using TensorFlow [39] as the computing engine. The mapping of the 32-bit input float tensors (i.e., weight matrix, activations) to the 8-bit quantized range is described in Figure 5. The mapping function (i.e., 8-bit quantizer) maps the input float tensors of the baseline model to the 8-bit quantized output. This function is defined in Equation (1):

$$q_{8bit} = \text{round}(m_f i_f) \quad (1)$$

where q_{8bit} is the 8-bit quantizer, m_f is the multiplier, and i_f is the input float tensor. The multiplier is the quantization constant that is multiplied with the float input tensor, as expressed in Equation (2):

$$m_f = \frac{2^7 - 1}{2 \max(|i_f|)} \quad (2)$$

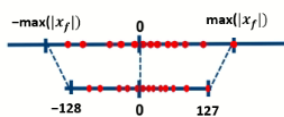


Figure 5. This figure shows the mapping of the input tensor float range to the 8-bit signed integer quantized range in symmetric mode.

5.2.2. Binarization

Binarization is a bit reduction technique that is considered an extreme case of quantization in which the weights and/or activations are encoded using a single bit (i.e., 1 bit). A single bit can be considered the atomic bit level of a number system; therefore, a significant decrease in the model accuracy is imminent due to loss of information during the binarization process. We did not binarize the first and last layer of the baseline model to minimize this loss.

In the binarization process, updating the weights during backward pass using the standard gradient descent approach is impossible because computing the loss gradient would result in zero in almost all conditions. We adopted a Straight-Through-Estimator (STE) pseudo function that has been proven to solve this issue.

We binarized the baseline model using Larq [40], an open-source Binary Neural Network library built on Keras [38]. The binarization function (a non-zero sign function) b_{1bit} takes the float tensor as input and returns a binary output (-1 and $+1$), as shown in Equation (3):

$$o_{1bit} = b_{1bit}(i_f), o_{1bit} \in \{-1, 1\}, i_f \in \{R\} \quad (3)$$

where b_{1bit} is the binarization function, o_{1bit} is the binary output generated, and i_f is the input float tensor. During the backward pass, the loss gradient is calculated using the STE function, which takes the output tensors as input and returns a binary output, which is constrained to the threshold value, as expressed in Equation (4):

$$loss_{gradient} = \begin{cases} 1 & \text{abs}(i_f) \leq threshold_{value} \\ 0 & \text{abs}(i_f) > threshold_{value} \end{cases} \quad (4)$$

where the $threshold_{value}$ is the float value that controls the $loss_{gradient}$ and i_f is the float tensor processed by the STE pseudo gradient function.

5.2.3. Network Pruning

Pruning is one of the oldest methods for compressing large CNN models for low-power and resource-constrained devices. Pruning explores and exploits redundant parameters that do not contribute significantly to the model performance. The effectiveness of the pruning method is dependent on how efficiently we can evaluate the parameters that are redundant in the network.

Over-pruning the baseline model will decrease the accuracy and damage the network completely. This can be mitigated by evaluating the model based on certain criteria after each pruning operation in an iterative manner. There are different pruning methods (e.g., weight-only pruning, node-only pruning, or layer pruning).

In this work, we pruned the baseline model using the magnitude-based weight pruning approach as opposed to the neuron-based method. We implemented this pruning method because it does not affect (i.e., decrease) the model accuracy significantly, especially when the base model is not complex (i.e., having few hidden layers, as with the baseline model). The weights of the baseline model pruned were selected using rank-based criteria, calculated using the absolute value of the individual weight in Equation (5):

$$rank_{weight} = |w_i|, w_i \in W \quad (5)$$

where $rank_{weight}$ is the rank of the individual weight, w_i is the weight, and W is the weight matrix associated with each neuron within the respective layer of the network.

5.2.4. Knowledge Distillation

Distilling knowledge (i.e., useful information) from a large model (the teacher) to a small model (the student) has shown excellent results in model compression. The concept of knowledge transfer is based on the idea that large models are robust and can learn complex patterns from data such that useful information can be transferred to the small model to mimic the behaviour of the large model.

We implemented the teacher-student model using a temperature-based softmax function at the output layer in Keras [38]. This technique was inspired by [21]. The teacher model (i.e., VGG16) was trained on our dataset, and the class probabilities vector for each data point (i.e., observation) was calculated and extracted. These probabilities vectors, also called soft labels, were distilled to the small model as the target label during training. We also trained the small model using the hard labels, and the overall losses generated by the small model were combined and weighted, as shown in Equation (6):

$$Loss_{total} = \alpha \times H(y, \sigma(z_s), T = 1) + \beta \times H(\sigma(z_t; T = t), \sigma(z_s, T = t)) \quad (6)$$

$Loss_{total}$ is the total loss, which is the combination of the student and distillation losses. The student loss is computed using the standard loss function by making the temperature parameter ($T = 1$). The temperature parameter controls the amount of information that can be distilled to the student. However, we need to keep in mind that the student has a threshold that limits the amount of information that it can retain from the teacher. The α and β are constants associated with the individual loss function taking the respective unnormalized log probabilities (z_s, z_t) for each class label.

5.2.5. Tensor Train Decomposition

The decomposition of a matrix into its low-rank embedding is a very important concept in linear algebra. Matrix decomposition is used extensively in applied data processing for computation acceleration and data compression. Tensor decomposition is a means of generalizing the concept of low-rank matrix decomposition by treating the matrix as a tensor (i.e., a higher-order array). There are standard matrix decomposition methods (e.g., QR, LU, Eigen, singular value decomposition (SVD), etc). SVD is the most widely used method but cannot operate directly on higher-order data structures such as tensors. Working directly on tensors offers the benefit of keeping the correlation between data points intact.

Tensor decomposition is still relatively new. Few methods have been developed (e.g., Canonical Polyadic (CP), Tucker, Tensor-Ring, Tensor-Train, etc.). We adopted the TT decomposition method due to the computational time (i.e., reconstruction and decomposition time) and storage space advantages that it has over CP, Tucker, and other tensor decomposition methods.

TT decomposition factorizes a tensor into sub-tensors called cores/factors. The number of cores is dependent on the dimensions of the input tensor. TT is based on SVD and the factorized outputs are expressed as a train of tensors (i.e., a product of smaller core tensors). The dense layer in the baseline model has a set of weights in its nodes transformed into TT matrices (i.e., 4D tensor shape). The TT matrices are factorized into four TT cores and each element of the tensor can be reconstructed as defined in Equation (7):

$$T(i_1, i_2, i_3, i_4) = \sum_{r_1, r_2, r_3}^{R_1 R_2 R_3} G^1(i_1 r_1) \cdot G^2(r_1 i_2 r_2) \cdot G^3(r_2 i_3 i_3) G^4(r_3 i_4) \quad (7)$$

where T is the original tensor, i_n represents the tensor indices, r_n corresponds to the ranks of the tensor, R_i are the compressed/contracted hidden indices, and G^i are the TT cores.

We retrained the baseline model using TT, which transforms the input and output parameters of the dense layer, excluding the softmax layer, into TT matrices, and the outputs were decomposed into tensor cores. We implemented this method using the t3f framework [41], a tensor train library built on TensorFlow [39].

5.2.6. Microarchitecture

The concept of designing sub-blocks/modules as a micro-unit in CNNs has shown promising results in terms of model size reduction and improved inference time without a significant decrease in model accuracy. The CNN microarchitecture relies on the residual knowledge that is adopted to carefully design each of the CNN sub-blocks that make up the entire network. The microarchitecture defines the dimensions and structure of each sub-module and how they are integrated to form the entire network.

There are several CNN microarchitectures that have been proposed for resource-constrained devices. In this work, we trained our dataset on MobileNet V1, MobileNet V2, MobileNet V3, and ShuffleNet. The results of these models were compared with those of the compressed models.

5.3. Evaluation and Ranking

An object detection model performs a detection task by fusing object classification and localization methods. Some methods involve using two separate algorithms (i.e., MobileNet and SSD). Other methods (e.g., YOLO) use a single model to perform both the classification and localization tasks. To optimize such a model for memory compression, speed, and accuracy, the parts of the model contributing to these metrics need to be optimized.

Much of the memory and computational power is expended by the base network of the object detection model. This formed the basis of our evaluation of the compression methods on the classification model instead of both the classification and localization models.

We based our evaluation and ranking on five key metrics: model size, accuracy, peak runtime memory footprint, computational cost, and inference time. The results corresponding to the key metrics obtained from the compressed models, micro-CNN models, and the base model are described in Table 2. These results were mean values calculated over a small number of experiments in order to reduce the error margin due to the stochastic nature of training CNN models.

Table 2. Evaluation metrics results of the compressed models, base model, and micro-CNN models.

Metric	Uncompressed		Compressed				Micro-CNN			
	Baseline	Quantized	Binarized	Pruned	Distilled	Tensor-Trained	V1	V2	V3	ShuffleNet
Model Size (KB)	4429.61	1119.54	105.50	1308.82	4429.61	258.02	12,856.38	8935.84	12,171.81	7761.17
Accuracy (%)	77.23	76.95	67.10	74.64	72.04	71.47	68.88	64.43	71.18	65.99
Inference Time (ms)	22.88	13.65	5.40	22.64	22.87	17.31	39.09	20.79	16.79	40.09
Computational Cost (MFLOPs)	66.44	7.29	6.96	66.44	66.44	64.05	94.01	52.88	15.24	158.63
Peak Memory Footprint (KB)	8907.81	3705.47	1775.78	8900.78	8907.59	4971.88	19,857.04	12,582.8	10,550.02	12,796.72

We evaluated the baseline, compressed, and micro-CNN models on a Google Coral Development Board with the following technical specifications: Quad Cortex-A53 and Cortex-M4F processors, Edge TPU co-processor (supports only int8 operations), 1 GB LPDDR4 RAM, and 8 GB eMMC flash memory [42]. The performance (i.e., key metrics) of the compressed, micro-CNN, and base models was evaluated using the weighted score-based ranking scheme that we developed.

Our ranking scheme relies on two major components: the weights (i.e., relevance scores) assigned to the criteria (i.e., key metrics) and the computed scores of the results generated by the compressed models, base model, and micro-CNN models. It is critical that the weights and scores should have the same scale (i.e., value range). The raw results generated in Table 2 were not scaled and this does not meet the requirement of the weighted score-based ranking method.

We applied a scaling and scoring function to the unscaled results generated in Table 2. This function maps and scores each unscaled result to a value in the range (1–10). The scaling function is defined as:

$$n_{scaled} = \frac{n - m_{min}}{m_{max} - m_{min}} \times (r_{max} - r_{min}) + r_{min} \quad (8)$$

where n_{scaled} is the scaled output value, n is the metric value to be scaled into $[r_{min}-r_{max}]$, m_{min} is the minimum of the metric value range, and m_{max} is the maximum of the metric value range. The r_{min} and r_{max} represent the minimum and maximum value of the target scale range (i.e., 1–10 as used in our experiment). The scale corresponds to the range of the weighted relevance score assigned to each evaluation metric.

Each metric was assigned a weighted relevance score in the range (1–10); a high weighted relevance score (e.g., 10) indicates that the corresponding metric has the highest priority (i.e., most significant) and vice versa, as shown in Table 3. The scores (i.e., the scaled metric values) and weights assigned to the evaluation metrics are described in Table 3.

The weighted relevance score controls the significance score of a metric during the evaluation of the performance of the compressed models, base model, and micro-CNN models. The weight of each metric is determined based on the application requirements. We assigned a weight value to the individual metric, as shown in Table 3. These values were generated based on our application requirements. The values can be adjusted to suit other applications.

The scaled values were scored on a scale of (1–10). A score of 10 assigned to the model per metric value means that the model with respect to the metric is the most significant, while a score of 1 means that the model is less significant. Table 4 shows the score of each model per metric value computed from the results (scaled metric values) obtained in Table 3.

The weighted score was calculated by computing the product of the corresponding weight of the relative importance score assigned to the metric and the score of the model corresponding to the metric. The mean weighted scores of the compressed models, micro-CNN models, and base model were ranked using an inverse ranking method similar to Spearman's rank approach (i.e., the largest mean weighted score was assigned a rank of 1, the second-largest mean weighted score was assigned a rank of 2, and as the mean weighted score decreases, the rank number increases by 1 until the maximum rank number n is reached, where n is the count of the models evaluated), as shown in Table 5.

Table 3. Weighted relevance score assigned to evaluation criteria and model results scaled to (1–10), which corresponds to evaluation metrics' scale range.

Metric	Weight	Uncompressed			Compressed			Micro-CNN			
		Baseline	Quantized	Binarized	Pruned	Distilled	Tensor-Trained	V1	V2	V3	ShuffleNet
Model Size (KB)	8	4.05	1.72	1.0	1.85	4.05	1.11	10.00	7.23	9.52	6.40
Accuracy (%)	10	10.00	9.80	2.88	8.18	6.35	5.95	4.13	1.00	5.75	2.10
Inference Time (ms)	6	4.87	2.82	1.00	4.81	4.86	3.63	8.45	4.40	3.52	10.00
Computational Cost (MFLOPs)	6	4.53	1.02	1.00	4.53	4.53	4.39	6.17	3.72	1.49	10.00
Peak Memory Footprint (KB)	7	4.55	1.96	1.00	4.55	4.55	2.59	10.00	6.38	5.37	6.49

Table 4. Metric scores corresponding to the scaled metric value.

Metric	Weight	Uncompressed		Compressed				Micro-CNN			
		Baseline	Quantized	Binarized	Pruned	Distilled	Tensor-Trained	V1	V2	V3	ShuffleNet
Model Size (KB)	8	6.00	8.00	10.00	7.00	6.00	9.00	2.00	4.00	3.00	5.00
Accuracy (%)	10	10.00	9.80	2.88	8.18	6.35	5.95	4.13	1.00	5.75	2.10
Inference Time (ms)	6	3.00	9.00	10.00	5.00	4.00	7.00	2.00	6.00	8.00	1.00
Computational Cost (MFLOPs)	6	5.00	9.00	10.00	5.00	5.00	6.00	4.00	7.00	8.00	3.00
Peak Memory Footprint (KB)	7	7.00	9.00	10.00	7.00	7.00	8.00	3.00	5.00	6.00	4.00

Table 5. Compressed models, base model, and micro-CNN models ranked by the weighted mean score.

Metric	Weight	Uncompressed		Compressed				Micro-CNN			
		Baseline	Quantized	Binarized	Pruned	Distilled	Tensor-Trained	V1	V2	V3	ShuffleNet
Model Size (KB)	8	48.00	64.00	80.00	56.00	48.00	72.00	16.00	32.00	24.00	40.00
Accuracy (%)	10	100.00	98.00	28.80	81.80	63.50	59.50	41.30	10.00	57.50	21.00
Inference Time (ms)	6	18.00	54.00	60.00	30.00	24.00	42.00	12.00	36.00	48.00	6.00
Computational Cost (MFLOPs)	6	30.00	54.00	60.00	30.00	30.00	36.00	24.00	42.00	48.00	18.00
Peak Memory Footprint (KB)	7	49.00	63.00	70.00	49.00	49.00	56.00	21.00	35.00	42.00	28.00
Weighted Mean Score		6.62	9.00	8.07	6.67	5.80	7.18	3.09	4.19	5.93	3.05
Rank		5	1	2	4	7	3	9	8	6	10

We calculated the mean weighted score per model for all metrics by computing the ratio of the arithmetic mean of the weighted scores and the sum of the weights assigned to all metrics. The equation is defined as:

$$W = \frac{\sum_{i=1}^n w_i m_i}{\sum_{i=1}^n w_i} \quad (9)$$

where W is the mean weighted score, i is the metric index, n is the total number of evaluation metrics, w_i is the weighted relevance score assigned to each metric, and m_i is the calculated score of the model corresponding to the metric.

The ranking result showed that the application's most effective, efficient, and suitable compression method is quantization with the rank of 1 (having the largest mean weighted score of 9.0), followed by binarization (having a mean weighted score of 8.07), as shown in Figure 6. As the rank number increases, the methods' effectiveness, efficiency, and suitability to meet the application requirements decrease.

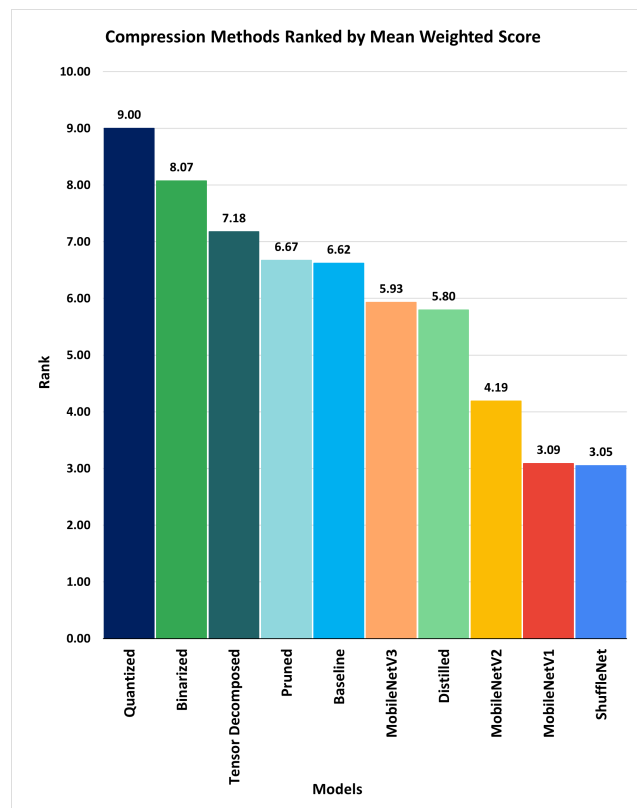


Figure 6. Compression methods ranked by the mean weighted score.

6. Discussion

The state-of-the-art methods for compressing deep learning models have shown excellent results in terms of peak runtime memory reduction, low latency, model size reduction, and computational efficiency, without a significant decrease in accuracy. These results vary from one compression method to another. The application requirements also differ, thus making it challenging to choose the optimal compression method for different applications (e.g., our ROROGREEN and SCDS projects). To address this, we propose a weighted score-based ranking scheme that enables us to evaluate and rank the compression methods based on their computed weighted mean scores. The weighted mean score is dependent on the metric scores and the weights assigned to the evaluation criteria, as shown in Table 4. The values in Table 4 correspond to the metric scores computed for each scaled evaluation criterion with respect to all the models evaluated.

The weight assigned to each evaluation metric determines the relevance score of the individual criterion. A high relevance score assigned to the metric with respect to the score range gives the metric a high effect when calculating the weighted metric score and the weighted mean score, as shown in Table 5. The metric scores correspond to the scaled metric values in Table 3. The scaled metric values correspond to the scaled model results in Table 2. The scaling ensures that all the values (i.e., the results obtained for all the metrics during model evaluation) use a common range. This eliminates the dominance effect of larger values regardless of the units when calculating the weighted mean scores. The metric scores ensure that the least and the most significant values of the evaluation metrics have the same interpretation (e.g., the lower the model size, the higher the compression rate with respect to the original model, whereas the higher the accuracy, the better).

We compare and rank the result of the computed weighted mean score of the baseline, compressed (i.e., quantized, binarized, pruned, distilled, and tensor-trained), and micro-CNN (i.e. MobileNetV1, MobileNetV2, MobileNetV3, and ShuffleNet) models, as shown in Figure 6. The weighted mean score for each model indicates the suitability score of the compression method that produces the model. A higher suitability score is desirable. The higher the weighted mean score (i.e., suitability score), the higher the rank (the highest rank is equivalent to 1 and it decreases as the rank value increases). The model with the highest weighted mean score (i.e., a rank of 1) is considered the most efficient and effective model for the hardware. As such, the compression technique that produces the model is considered the most suitable compression method for optimizing the base model for the application.

The limitation of our proposed scheme is the selection of weights for the performance metrics. These weights must reflect the priorities (i.e., level of significance) of the application criteria. This is considered a limiting factor because the weights are dependent on the application requirements, such as the available hardware resources (e.g., processor capability, flash size, RAM, etc.), model requirements, and the other requirements that are specific to the application (e.g., real-time capability, on-device model performance, etc.), and thus must be chosen appropriately. We do not consider this a major limiting factor because we have demonstrated how we selected the weights that are appropriate for the performance metrics in our case studies (ROROGREEN and SCDC projects).

Choosing the optimal model compression method for on-device AI applications is challenging due to the lack of an application-specific framework for evaluating methods for deep learning model compression for resource-constrained edge devices. We addressed this issue using our proposed weighted score-based ranking scheme. The scheme helps us to identify the quantization technique as the optimal method for compressing our object detection and tracking models for the application based on our requirements.

7. Conclusions and Future Work

In this work, we evaluated and ranked the state-of-the-art methods for CNN model compression for resource-constrained edge devices using a weighted score-based ranking scheme that we developed. Our ranking method uses five key metrics (i.e., model size, accuracy, inference time, computational cost, and peak memory footprint) computed for each model generated by the compression methods. We introduced an individual weight to these key metrics. The individual weight reflects how relevant/important the metric is in our application.

This work was motivated by the lack of a clear framework/method for selecting the most efficient methods for compressing CNN models for edge computing devices (e.g., micro-controllers, small computer boards, portable devices, mobile devices, DNN hardware accelerators, etc.). As such, we developed a weighted score-based ranking scheme to address this issue.

We applied our method to the baseline model developed, compressed versions of the baseline model produced by the state-of-the-art compression methods that we implemented, and the micro CNN models trained on a portion of the SCDS and ROROGREEN dataset. According to the ranking of the mean weighted scores computed, the quantized model obtained the highest rank, with a mean weighted score of 9.00, followed by the binarized, model having a mean weighted score of 8.07. ShuffleNet has the lowest rank in Table 5, with a mean weighted score of 3.05. This clearly shows that the quantization technique is the most suitable model compression method for both the SCDS and ROROGREEN projects.

Determining the most effective, efficient, and optimal method/methods for optimizing deep learning models for edge devices can be very challenging. We addressed this issue using our weighted score-based evaluation and ranking method.

In the future work, we will focus on how we can further improve the metrics of the best-ranked method (i.e., the quantized model).

Author Contributions: O.A.A.: conceptualization, methodology, data curation, software, resources, validation, visualization, writing—original draft. M.L.: conceptualization, investigation, resources, data curation, writing—review and editing, supervision, project administration, funding acquisition. E.P.: conceptualization, supervision, funding acquisition, writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been conducted as part of the project “ICT programme”, which was supported by the European Union through the European Social Fund. This work was also supported by the Innovation Fund Denmark “Green RORO shipping through digital innovation (ROROGREEN)”. We also want to express our gratitude to the Estonian Research Council for partially supporting this work through grant PRG658.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The ROROGREEN and SCDC datasets used are not publicly available.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AL	Activation Layer
BN	Batch Normalization
CNN	Convolutional Neural Network
CP	Canonical Polyadic
Conv2D	Convolution 2D
DNN	Deep Neural Network
F-RCC	Faster Region-Based Convolutional Neural Network
FC	Fully Connected
FCL	Fully Connected Layer
ReLU	Rectified Linear Unit
SVD	Singular Value Decomposition
SGD	Stochastic Gradient Descent
TT	Tensor Train

References

1. Zhao, Z.Q.; Zheng, P.; Xu, S.-T.; Wu, X. Object detection with deep learning: A review. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *1*, 1–21. [[CrossRef](#)] [[PubMed](#)]
2. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014.
3. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
4. Liu, S.; Deng, W. Very deep convolutional neural network-based image classification using small training sample size. In Proceedings of the 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), Kuala Lumpur, Malaysia, 3–6 November 2015; pp. 730–734.
5. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and less than 0.5 mb model size. *arXiv* **2016**, arXiv:1602.07360.
6. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv* **2017**, arXiv:1707.01083.
7. Tan, M.; Le, Q.V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv* **2020**, arXiv:1905.11946.
8. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
9. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2014.
10. Cheng, Y.; Wang, D.; Zhou, P.; Zhang, T. A survey of model compression and acceleration for deep neural networks. *arXiv* **2017**, arXiv:1710.09282.
11. Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv* **2016**, arXiv:1602.02830.

12. Courbariaux, M.; Bengio, Y.; David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In Proceedings of the 28th International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2016.
13. Nahshan, Y.; Chmiel, B.; Baskin, C.; Zheltonozhskii, E.; Banner, R.; Bronstein, A.M.; Mendelson, A. Loss aware post-training quantization. *arXiv* **2020**, arXiv:1911.07190.
14. Long, X.; Zeng, X.; Ben, Z.; Zhou, D.; Zhang, M. A novel low-bit quantization strategy for compressing deep neural networks. *Comput. Intell. Neurosci.* **2020**, *2*, 1–7. [[CrossRef](#)] [[PubMed](#)]
15. Deng, X.; Zhang, Z. An embarrassingly simple approach to training ternary weight networks. *arXiv* **2020**, arXiv:2011.00580.
16. Shayer, O.; Levi, D.; Fetaya, E. Learning discrete weights using the local reparameterization trick. *arXiv* **2018**, arXiv:1710.07739.
17. Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. *Xnor-Net: Imagenet Classification Using Binary Convolutional Neural Networks*; Springer: Cham, Switzerland, 2016.
18. Zhou, Y.; Moosavi-Dezfooli, S.-M.; Cheung, N.-M.; Frossard, P. Adaptive quantization for deep neural network. *arXiv* **2017**, arXiv:1712.01048.
19. Stock, P.; Joulin, A.; Gribonval, R.; Graham, B.; Jegou, H. And the bit goes down: Revisiting the quantization of neural networks. *arXiv* **2020**, arXiv:1907.05686.
20. Bucilun, C.; Caruana, R.; Niculescu-Mizil, A. Model compression. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Ser. KDD'06, New York, NY, USA, 20–23 August 2006; pp. 535–541.
21. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531.
22. Cho, J.H.; Hariharan, B. On the efficacy of knowledge distillation. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea, 27 October–2 November 2019.
23. Lu, Y.; Kumar, A.; Zhai, S.; Cheng, Y.; Javidi, T.; Feris, R. Fullyadaptive feature sharing in multi-task networks with applications in person attribute classification. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2016.
24. Denton, E.; Zaremba, W.; Bruna, J.; LeCun, Y.; Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In Proceedings of the 27th International Conference on Neural Information Processing Systems—Volume 1 (NIPS'14), Montreal, QC, Canada, 8–13 December 2014; pp. 1269–1277.
25. Kholavchenko, M. Iterative low-rank approximation for CNN compression. *arXiv* **2020**, arXiv:1803.08995.
26. Sankararaman, K.A.; De, S.; Xu, Z.; Huang, W.R.; Goldstein, T. The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. *arXiv* **2020**, arXiv:1904.06963.
27. Huynh, T.Q.; Setiono, R. Effective neural network pruning using cross-validation. In Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, Montreal, QC, Canada, 31 July–4 August 2005; Volume 2, pp. 972–977.
28. Ding, X.; Ding, G.; Zhou, X.; Guo, Y.; Han, J.; Liu, J. Global sparse momentum SGD for pruning very deep neural networks. *arXiv* **2019**, arXiv:1909.12778.
29. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning both weights and connections for efficient neural networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015.
30. Reed, R. Pruning algorithms—A survey. *IEEE Trans. Neural Netw.* **1993**, *4*, 740–747. [[CrossRef](#)]
31. Chang, J.; Sha, J. Prune deep neural networks with the modified l1/2 penalty. *IEEE Access* **2019**, *7*, 2273–2280. [[CrossRef](#)]
32. Mozer, M.C.; Smolensky, P. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Proceedings of the 1st International Conference on Neural Information Processing Systems, Ser. NIPS'88*; MIT Press: Cambridge, MA, USA, 1988; pp. 107–115.
33. Augasta, M.; Kathirvalavakumar, T. A novel pruning algorithm for optimizing feedforward neural network of classification problems. *Neural Process. Lett.* **2011**, *34*, 241–258. [[CrossRef](#)]
34. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2016**, arXiv:1409.1556.
35. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the inception architecture for computer vision. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2015.
36. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv* **2016**, arXiv:1602.07261.
37. Barry, D.; Shah, M.; Keijsers, M.; Khan, H.; Hopman, B. xyolo: A model for real-time object detection in humanoid soccer on low-end hardware. *arXiv* **2019**, arXiv:1910.03159.
38. Chollet, F. Keras. 2015. Available online: <https://github.com/fchollet/keras> (accessed on 14 July 2021).
39. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: tensorflow.org (accessed on 14 July 2021).
40. Geiger, L.; Team, P. Larq: An open-source library for training binarized neural networks. *J. Open Source Softw.* **2020**, *5*, 1746. [[CrossRef](#)]
41. Novikov, A.; Izmailov, P.; Khrulkov, V.; Figurnov, M.; Oseledets, I. Tensor train decomposition on tensorflow (t3f). *J. Mach. Learn. Res.* **2020**, *21*, 1–7. Available online: <http://jmlr.org/papers/v21/18-008.html> (accessed on 16 July 2021).
42. Google Coral Products Page. Available online: <https://coral.ai/docs/dev-board/datasheet/> (accessed on 30 September 2021).

Appendix 2

II

O. A. Ademola, P. Eduard, and L. Mairo, “ Ensemble of tensor train decomposition and quantization methods for deep learning model compression,” in *2022 International Joint Conference on Neural Networks (IJCNN)*,, pp. 1–6, IEEE, 2022

Ensemble of Tensor Train Decomposition and Quantization Methods for Deep Learning Model Compression

Olutosin Ajibola Ademola

Department of Computer Systems

Tallinn University of Technology

Embedded AI Research Laboratory

Ehitajate tee 5, 19086 Tallinn, Estonia

olutosin.ademola@taltech.ee

Petlenkov Eduard

Department of Computer Systems

Tallinn University of Technology

Centre for Intelligent Systems

Ehitajate tee 5, 19086 Tallinn, Estonia

eduard.petlenkov@taltech.ee

Leier Mairo

Department of Computer Systems

Tallinn University of Technology

Embedded AI Research Laboratory

Ehitajate tee 5, 19086 Tallinn, Estonia

mairo.leier@taltech.ee

Abstract—We have seen tremendous growth in the adoption of convolutional neural networks (CNNs) over the years in solving real-world problems such as image analysis, object detection, auto-translation, etc. This exponential growth has been due to the unprecedented outcomes achieved over traditional methods. However, these remarkable achievements come at a cost. Convolutional neural networks are memory intensive, computationally expensive, and usually cause the underlying pieces of hardware running the models to consume an excessive amount of power during inference. These factors impede its deployment in mobile and embedded applications due to the availability of limited hardware resources. To address this problem, different methods for compressing deep learning models have been presented in different works. In this paper, we propose an approach to further improve the compression results. Our approach leverages an ensemble of two model compression methods — tensor train decomposition and 8-bit integer quantization. Our goal is to demonstrate the efficiency and effectiveness of the ensemble technique by applying it to the baseline CNN model trained on our dataset. We compared the performance of the baseline and compressed model produced by the ensemble. We achieved a 57x reduction in model size compared with 4x and 17x compression factors that would have been achieved with either only quantization or tensor train decomposition methods respectively. We further demonstrated an end-to-end trainable pipeline for training any CNN model based on our proposed method.

Index Terms—model compression, an ensemble of compression methods, 8-bit quantization, tensor train representation, and tensor decomposition.

I. INTRODUCTION

Convolutional neural networks have gained traction over the years in solving problems related to vision processing (e.g., image recognition, pose estimation, object detection, etc.) as a result of the remarkable results shown over non-deep learning-based methods. This landmark was achieved due to the availability of powerful processing units, a large number of datasets, the capability of the networks to learn

automatically, and most importantly the complexity of the network architectures used in generating the models.

Owing to this breakthrough, there has been a shift from classical computer vision/machine learning-based models running in edge applications to deep learning-based models (e.g., CNN). However, there are impeding factors hindering this transition such as the computation & storage cost, runtime memory usage, and the power requirements of the CNN models [1]. As such, few compression techniques (e.g., binarization, pruning, quantization, tensor decomposition, clustering, distillation, etc.) have been proposed to address these factors [2].

The performance of the compressed models varies with the compression techniques. In this paper, we present an ensemble of tensor train decomposition and 8-bit quantization methods for compressing large CNN models. This method offers a deep compression of more than a factor of 20x when compared with the individual compression method such as binarization, quantization, tensor decomposition, distillation, pruning, etc. This allows low-power and resource-constrained devices such as micro-controllers and other portable/small devices to run deep learning models seamlessly and efficiently.

The choice of the compression methods to coalesce was inspired by the results produced based on the weighted score-based ranking scheme proposed by [3].

II. RELATED WORK

Deep learning models (e.g., CNN) have been proven to be very efficient in solving vision-based tasks such as image recognition [4]. This efficiency comes at some cost (e.g., computation, memory, energy, time, etc.). This has hindered its application in portable devices [5]. Different methods have been proposed for optimizing these large models for the devices and have shown remarkable results. However, the performance of the optimized models usually varies with the compression methods.

The variation in performance of the compressed model is dependent on the layers (i.e., convolutional or fully con-

This work was fully funded by the IT Academy Programme, Estonia. We also want to express our gratitude to the Estonian Research Council for partially supporting this work through the grant PRG658.

nected layers) being optimized by the compression techniques. The objectives of optimization also vary (e.g., model size, bandwidth, energy consumption, runtime memory, accuracy, inference speed, etc.) [3]. In the works of [6] [7] [8], the network weights that are inconsequential to the network output are pruned (i.e., set to zero) resulting in a small network. [9] implemented a structured-based pruning by eliminating the low-rank kernel filters of the convolutional layers.

By default, the parameters (e.g., weights, biases, activations, etc.) of the networks are encoded in 32-bit floating point, [10] [11] [12] showed the possibility of compressing the networks by using a low precision fixed-point bit-width to represent these parameters. [13] [14] trained deep networks with weights constrained to 1 bit (i.e., -1 and +1) respectively. However, the extreme case of quantization (i.e., binarization) usually hurts the model accuracy, especially when the first & last layers are binarized. This is due to the level of sensitivity of both layers to binarization.

Another method proposed for deep neural network compression is the matrix decomposition/factorization technique. This method treats each layer of the network as matrices/tensors (e.g., dense and convolutional layers). It involves computing an approximation of the matrices/tensors using numerical methods. The approximated matrices/tensors are usually compact and require lesser parameters, hence, producing a smaller and faster network. [15] proposed a low-rank approximation method for factorizing the convolutional filters. [16] [17] [18] applied tensor decomposition methods to the dense layer of the network resulting in a smaller network size with a 1-2% drop in accuracy.

In this paper, we propose an ensemble of tensor train decomposition and 8-bit quantization for deep compression and acceleration of deep learning models for resource-constrained hardware.

III. TENSOR TRAIN DECOMPOSITION

A tensor is a higher-order array, as such, the combination of the factors of this tensor is referred to as a tensor train. Tensor Train decomposition (TT-decomposition) factorises a tensor into sub-tensors (i.e., low-rank cores or factors) with each core being the low-rank representation of the decomposed tensor. There are different layers in a convolutional neural network, however, the computational and memory resources requirements of each layer vary.

The dense and convolutional layers of the network are the most computational and memory hungry. However, the dense layer hogs more memory than the convolutional layer because it stores about 90% of the overall network weights. This layer also accounts for a less percentage of the total floating-point operations during propagation when compared with the convolutional layer. This gives us the opportunity to explore and exploit the layer for further optimization.

The weights vectors of all the neurons of the dense layer can be treated as a matrix, as such, we can transform this weights matrix into its TT-representation. This compact format which is controlled by the TT-ranks allows us to represent the

weights tensor with fewer amount of parameters (i.e., weights and biases), hence, reducing the size of the memory needed to store the weights and accelerating computation.

A. TENSOR TRAIN REPRESENTATION

Given a weight matrix W of shape $(I$ and J corresponding to rows and columns of the matrix), we can transform W into a tensor \mathcal{W} . The tensor \mathcal{W} is converted into its TT-representation. \mathcal{W} is formed by combining the indexes $(a_i(m) \& b_j(n))$ where m & n are the row and column indices and i & j are the rows and columns respectively).

A tensor is represented in its TT format using cores or factors. Each core has a size of $(r_{n-1} \times r_n)$ where r_0 (i.e., the TT-rank of the first core) and r_n (i.e., the TT-rank of the last core) are equal to 1 (i.e., $r_0 = r_n = 1$). The number of cores scales linearly with the dimension of the input tensor (i.e., $G_i \in [G_1, \dots, G_n]$ where n is the dimension of the tensor).

Given a 4D tensor \mathcal{T} decomposed into its four cores (G_1, \dots, G_4) , we can compute the elements of tensor \mathcal{T} from the four cores using Equation (1):

$$\mathcal{T}(i_1, i_2, i_3, i_4) = \sum_{r_1}^{R_2} \sum_{r_2}^{R_3} \sum_{r_3}^{R_4} G_1(i_1 r_1) \cdot G_2(r_1 i_2 r_2) \cdot G_3(r_2 i_3 i_3) \cdot G_4(r_3 i_4) \quad (1)$$

where \mathcal{T} is the 4D tensor, i_n represents the tensor element index, r_n corresponds to the TT-ranks of the cores, R_i are the compressed/contracted hidden indices, and G_i are the TT-cores or factors. The amount of computational and storage resources needed by the cores scales with the r (i.e., the maximal TT-rank), as such, must be kept as small as possible.

IV. QUANTIZATION

There are different layer operations that are performed in a deep neural network. A typical convolutional neural network involves operations such as pooling, convolution, activation, linear transformation etc. These operations are performed on the parameters of the network in each layer as shown in Figure 1. These parameters (e.g., weights, biases, activations, input, and output) are of float data types encoded in 32-bit.

Quantization reduces a full precision 32-bit float parameter to a small bit width (e.g., 1, 2, 3, 4, 8, etc.).

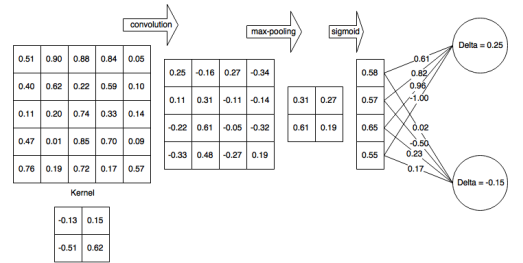


Fig. 1. The figure shows the different types of layer operations that are performed in a typical convolutional neural network.

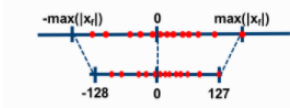


Fig. 2. A figure showing the conversion of 32-bit floating-point values to 8-bit unsigned integer quantized range.

The lower the number of bits that represents each network parameter, the lower the storage cost and the faster the computation in the layer.

This reduction often leads to a little degradation in model performance (i.e., accuracy) which is usually within an acceptable limit. However, this acceptance limit is application-specific. To mitigate the decrease in model accuracy, we quantized all the network parameters except the input and output tensors. This allows the model to behave closely to the uncompressed 32-bit float model.

Given a 32-bit input float tensor X_f (e.g., the weight matrix), each weight can be quantized to an 8-bit unsigned integer using both Equation (2) and Equation (3):

$$m_f = \frac{2^7 - 1}{\max(|X_f|)} \quad (2)$$

$$q_{8bit} = \text{round}(m_f X_f) \quad (3)$$

where m_f is the scaling factor or multiplier and q_{8bit} is the quantized output whose boundary is restricted by the absolute value of X_f as shown in Figure 2.

V. ENSEMBLE OF TENSOR TRAIN AND 8-BIT QUANTIZATION

The Ensemble technique has been proven to be a very effective approach to improving machine learning model accuracy. The idea of the ensemble in relation to machine learning involves combining different models trained on the same dataset such that the final prediction is the average of the predictions made by the different models. It has also been shown that instead of combining multiple models during inference due to the computational and memory requirements, the information (i.e., model expressiveness) of the ensemble can be distilled to a single model.

The Ensemble technique is usually adopted when there are different possibilities (i.e., methods) to solve a problem or improve an existing solution. It is also mandatory for these methods to be capable of being integrated either in a sequential or parallel manner. This is valid for large model compression because different methods can be combined sequentially (i.e., one method compresses the original model and the resulting model is further compressed by another compression method). However, the resulting model format must be supported by the other compression method. In this paper, our approach leverages an ensemble of tensor train decomposition and 8-bit quantization methods as shown in Figure 3.

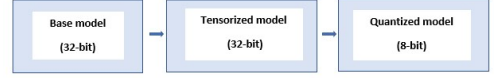


Fig. 3. A block representation of our trainable end-to-end compression pipeline using an ensemble of tensor train decomposition and 8-bit quantization methods.

TABLE I
BASE CNN MODEL ARCHITECTURE.

Layer Type	Output Size	Parameters
CONV2D	(None, 64, 64, 32)	864
BN	(None, 64, 64, 32)	96
MAXPOOL2D	(None, 32, 32, 32)	0
CONV2D	(None, 32, 32, 64)	18432
BN	(None, 32, 32, 64)	192
MAXPOOL2D	(None, 16, 16, 64)	0
CONV2D	(None, 16, 16, 64)	36928
BN	(None, 16, 16, 64)	192
FLATTEN	(None, 16384)	0
DENSE	(None, 64)	1048576
BN	(None, 64)	192
DENSE1	(None, 11)	704
ACTIVATION	(None, 11)	0
TOTAL PARAMETERS		1106209

The end-to-end model compression pipeline using an ensemble of tensor train decomposition and 8-bit quantization is described in Figure 4. This pipeline is trainable (i.e., the compression operations are implemented during model training). The first block of the pipeline is the base model. This has a set of convolution and dense blocks as shown in Table I. The architecture of the baseline model was inspired by the work of [3]. The tensor train block compresses the weights matrix of the dense layer of the base network. This results in a compact model (i.e., a tensortrained model with a reduced number of parameters). The 8-bit quantization block transforms the tensor trained model parameters (i.e., weights, biases, and activations) from their 32-bit representations to 8-bit unsigned integer values. This results in a deeply compressed model without a significant decrease in accuracy when compared with the baseline model.

VI. EXPERIMENTS

Tensor train decomposition transforms a network layer into a tensor train layer. The resulting network is referred to as a TT-network. The parameters of the tensor train layer (TT-layer) need to be configured just like any layer of the network. These parameters include the maximal tensor train ranks of the weight tensors and the dimensions of both the input & output tensors of the layer. The baseline network as described in Table I has a single dense layer in its hidden layer. This layer accounts for 94.79% (1048576/1106209 parameters) of the total parameters of the network. The dense layer transforms an input vector of size (16384) to an output vector of size (64).

We applied tensor train decomposition to the dense layer of the baseline model.

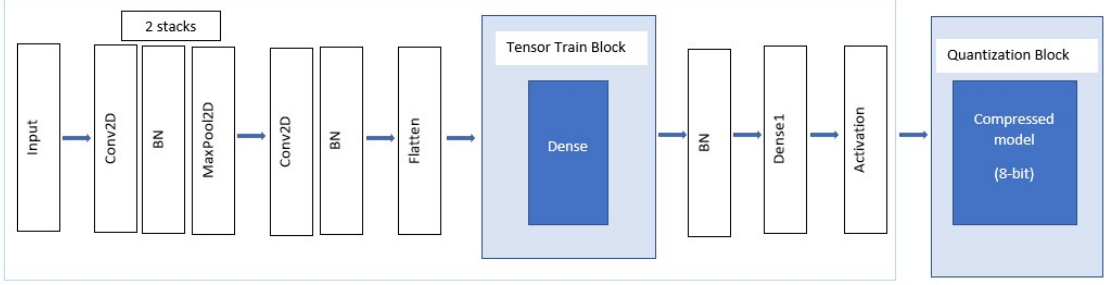


Fig. 4. An end-to-end trainable pipeline of compressing a model using an ensemble of Tensor Train decomposition and 8-bit quantization.

TABLE II
THE DIFFERENT PARAMETERS CONFIGURATION OF THE TT-LAYER

Configuration Type	TT-layer Parameters		
	TT-ranks	Input dims	Output dims
TT-I	4	(16,16,8,8)	(4,4,2,2)
TT-II	6	(16,16,8,8)	(4,4,2,2)
TT-III	8	(16,16,8,8)	(4,4,2,2)
TT-IV	10	(16,16,8,8)	(4,4,2,2)
TT-V	4	(16,8,16,8)	(4,2,4,2)
TT-VI	6	(16,8,16,8)	(4,2,4,2)
TT-VII	8	(16,16,8,8)	(4,4,2,2)
TT-VIII	10	(16,16,8,8)	(4,4,2,2)
TT-IX	4	(64,2,64,2)	(8,1,8,1)
TT-X	6	(64,2,64,2)	(8,1,8,1)
TT-XI	8	(64,2,64,2)	(8,1,8,1)
TT-XII	10	(64,2,64,1)	(8,1,8,1)

This decomposition transforms the dense layer into a TT-layer with fewer parameters while maintaining the expressiveness of the layer.

We experimented with the different values of the TT-layer parameters as shown in Table II. We trained the model so that we can use small TT-ranks because it controls the compression efficiency, and hence, a small TT-rank is desirable.

The low-rank assumption inherited by the model during training is minimised by the optimizer. We trained several models using the different configurations of the TT-layer to generate different tensor trained models as shown in Table III. We quantized the parameters of the best tensor trained model to their 8-bit unsigned integer representations to achieve a more compact model without a significant decrease in the model performance.

Our end-to-end trainable pipeline of the ensemble of tensor train decomposition and 8-bit unsigned integer quantization was implemented in the T3F framework [19]. T3F is an open-source tensor train decomposition library built on Keras [20] and TensorFlow [21].

VII. RESULT AND DISCUSSION

We evaluated the performance of the models — baseline, quantized only, tensor trained only, and the quantized tensor trained generated by our ensemble approach. We based our evaluation on five key performance indicators — model size, accuracy, model speed, computational cost (total number of floating-point operations during a single forward pass), and the peak runtime memory footprint during inference. All the models were evaluated on the Coral development board having quad Cortex-A53 & Cortex-M4F processors with an integrated edge tensor processor unit (TPU) [22].

To compress the baseline model efficiently, the appropriate TT-layer parameters setting is required. This is the same as fine-tuning any deep learning model using a domain-residual approach during training. We experimented with different parameter values as shown in Table II. Each configuration type (e.g., TT-I, TT-II, TT-III, etc.) results in a tensor trained model. These models were evaluated and ranked using a weighted score-based ranking method to obtain the TT-layer configuration that yields the most optimal tensor trained model. The tensor trained model generated by the TT-V configuration type ranked highest, hence, was selected as the values for the parameters of TT-layer for compression.

The baseline model was compressed using the TT-V configuration type obtained via fine-tuning. The resulting tensor trained model was quantized into its 8-bit unsigned integer representation. This yields a deep compressed model (i.e., a quantized tensor trained model produced by the ensemble of tensor train decomposition and 8-bit quantization methods).

We compared the performance of the QuanTT model (i.e., the quantized tensor trained model) trained on our dataset with the baseline model as shown in Table IV. Our results showed that QuanTT model achieves an accuracy of 69.45% on the test data compared with a 77.23% accuracy achieved by the baseline model. This is approximately a 10% decrease in the model accuracy. However, QuanTT model utilizes a peak memory footprint of 1632.81KB, whereas the baseline model uses 8907.81KB.

TABLE III
THE PERFORMANCE COMPARISON OF THE TENSOR TRAINED COMPRESSED MODELS WITH DIFFERENT CONFIGURATION

Configuration Type	Key Performance Indicators				
	Model Size (KB)	Peak Memory Footprint (KB)	Accuracy%	Model Speed (ms)	Computational Cost (MFLOPs)
TT-I	243	3617.19	72.91	18.53	64.34
TT-II	250	3621.09	72.62	19.40	64.34
TT-III	259	3628.91	74.06	20.48	64.34
TT-IV	271	3644.53	70.32	21.80	64.34
TT-V	244	3613.28	73.48	18.48	64.34
TT-VI	251	3621.09	74.35	19.38	64.34
TT-VII	260	3632.81	72.62	20.38	64.34
TT-VIII	272	3640.62	70.89	21.78	64.34
TT-IX	278	3652.34	69.74	19.59	64.34
TT-X	321	3695.31	74.35	21.24	64.34
TT-XI	381	3746.09	74.92	23.21	64.34
TT-XII	458	3816.41	70.31	25.73	64.34

TABLE IV
THE PERFORMANCE COMPARISON OF THE UNCOMPRESSED AND COMPRESSED MODELS BASED ON KEY INDICATORS

Model Type	Key Performance Indicators				
	Model Size (KB)	Peak Memory Footprint (KB)	Accuracy (%)	Model Speed (ms)	Computational Cost (MFLOPs)
Baseline	4429.61	8907.81	77.23	22.88	66.44
Quantized	1119.4	3705.47	76.95	13.65	7.29
Tensor trained	244	3613.28	73.48	18.48	64.34
QuanTT*	76.7	1632.81	69.45	12.89	7.06

* Quantized tensor trained model.

This is a 5x reduction in the model overall runtime memory requirement. The QuanTT model is about 57x (i.e., 76.7 KB) smaller in model size and 2x (12.89 ms) faster than the baseline model whose model size and speed are 4429.61KB and 22.88ms respectively. The total number of floating-point operations of the baseline is about 9x greater than that of the QuanTT model.

In this paper, we have shown the effectiveness of an ensemble of tensor train decomposition and 8-bit quantization methods for deep compression of CNN model that cannot fit into the memory of mobile and embedded devices.

It is important to point out the limitation of one of the proposed methods. There are limiting factors associated with the application tensor train decomposition to the dense layers of the networks. These limitations are choosing the appropriate values of the input & output tensors dimensions and the TT-ranks of the TT-layer. However, we demonstrated how we addressed these using a fine-tuning approach as shown in Table III.

VIII. CONCLUSION

Due to the success of CNN, we have seen rapid growth in its adoption in different vision-based applications such as object detection, image classification, text processing, etc. However, there are impeding factors such as the memory and computational constraints that are limiting its adoption in mobile and embedded applications. Different methods such as pruning, quantization, binarization, tensor decomposition, knowledge distillation etc. have been proposed in different works to address these impeding factors. In this paper, we leverage an ensemble of tensor train decomposition and 8-bit unsigned

integer quantization to further improve the compression results of the existing methods.

The choice of compression methods is dependent on the application requirements. These requirements determine the optimization goals (e.g., accuracy, model speed, storage size, runtime memory usage, FLOPs, etc.). Existing methods such as pruning and clustering are excellent methods for model size optimization but do not offer any benefit relating to speed, FLOPs, and runtime memory usage improvement. Knowledge distillation relies on the architecture of the student model and the benefits offered are dependent on the architecture and the capacity of the teacher model. Our approach leverages an ensemble of tensor train decomposition and 8-bit quantization methods that individually offers all the optimization goals i. As such, combining these methods significantly improves the overall performance of the compressed model.

We demonstrated the effectiveness and efficiency of our proposed method by applying it to the baseline model in Table I. However, any CNN model of choice (e.g., Mobilenet, Resnet, Shufflenet, efficient, etc.) can be treated as the baseline model. The dense layer of our baseline model is transformed into a TT-layer. We fine-tuned the parameters of the TT-layer to obtain the parameters that yielded the best compression result as shown in Table II. We applied the best TT-layer configuration to the baseline model and the parameters (excluding the input and output tensors) of the tensor trained model were quantized to their 8-bit representations. The quantized tensor trained model achieved a 57x reduction in model size compared to a 4x or 18x compression factor that would have been achieved with either only 8-bit quantization or tensor

train decomposition methods respectively.

REFERENCES

- [1] N.O. Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. Velasco-Hernández, L. Krpalkova, D. Riordan, & J. Walsh (2019). Deep Learning vs. Traditional Computer Vision. CVC.
- [2] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 10 2017.
- [3] O. A. Ademola, M. Leier, and E. Petlenkov, "Evaluation of Deep Neural Network Compression Methods for Edge Devices Using Weighted Score-Based Ranking Scheme," *Sensors*, vol. 21, no. 22, p. 7529, Nov. 2021, doi: 10.3390/s21227529.
- [4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12)*. Curran Associates Inc., Red Hook, NY, USA, 1097–1105.
- [5] Mahony, Niall O' et al. "Deep Learning vs. Traditional Computer Vision." CVC (2019).
- [6] T. Q. Huynh and R. Setiono, "Effective neural network pruning using cross-validation," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, 2005, pp. 972–977 vol. 2.
- [7] X. Ding, G. Ding, X. Zhou, Y. Guo, J. Han, and J. Liu, "Global sparse momentum sgd for pruning very deep neural networks," 2019.
- [8] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015.
- [9] J. Chang and J. Sha, "Prune deep neural networks with the modified 11/2 penalty," *IEEE Access*, vol. 7, pp. 2273–2280, 2019.
- [10] Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, and A. Mendelson, "Loss aware post-training quantization," 2020.
- [11] X. Long, X. Zeng, Z. Ben, D. Zhou, and M. Zhang, "A novel low-bit quantization strategy for compressing deep neural networks," *Computational Intelligence and Neuroscience*, vol. 2020, pp. 1–7, 02 2020.
- [12] O. Shayer, D. Levi, and E. Fetaya, "Learning discrete weights using the local reparameterization trick," 2018.
- [13] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," 2016.
- [14] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," 2016.
- [15] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'14)*. MIT Press, Cambridge, MA, USA, 1269–1277.
- [16] Dechun Song, Peiyong Zhang, and Feiteng Li. 2020. Speeding Up Deep Convolutional Neural Networks Based on Tucker-CP Decomposition. In *Proceedings of the 2020 5th International Conference on Machine Learning Technologies (ICMLT 2020)*. Association for Computing Machinery, New York, NY, USA, 56–61.
- [17] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, vol. 2. Neural information processing systems foundation, 2014, pp. 1269–1277, 28th Annual Conference on Neural Information Processing Systems 2014, NIPS 2014 ; Conference date: 08-12-2014 Through 13-12-2014.
- [18] M. Kholiavchenko, "Iterative low-rank approximation for CNN compression," 2019.
- [19] A. Novikov, P. Izmailov, V. Khurlov, M. Figurnov, I. Oseledets, Tensor train decomposition on tensorflow (t3f). *J. Mach. Learn. Res.* **2020**, *21*, 1–7, Available online: <http://jmlr.org/papers/v21/18-008.html> (accessed on 16 July 2021).
- [20] F. Chollet, Keras. 2015. Available online: <https://github.com/fchollet/keras> (accessed on 14 July 2021).
- [21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems". 2015. Available online: tensorflow.org (accessed on 14 July 2021).
- [22] Google Coral Products Page. Available online: <https://coral.ai/docs/dev-board/datasheet/> (accessed on 30 September 2021).

Appendix 3

III

O. A. Ademola, E. Petlenkov, and M. Leier, "Resource-aware scene text recognition using learned features, quantization, and contour-based character extraction," *IEEE Access*, 11:56865–56874, 2023

Received 8 May 2023, accepted 28 May 2023, date of publication 7 June 2023, date of current version 13 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3283931

 RESEARCH ARTICLE

Resource-Aware Scene Text Recognition Using Learned Features, Quantization, and Contour-Based Character Extraction

OLUTOSIN AJIBOLA ADEMOLA¹, EDUARD PETLENKOV², (Member, IEEE),
AND MAIRO LEIER¹

¹Embedded AI Research Laboratory, Department of Computer Systems, Tallinn University of Technology, 19086 Tallinn, Estonia

²Centre for Intelligent Systems, Department of Computer Systems, Tallinn University of Technology, 19086 Tallinn, Estonia

Corresponding author: Olutosin Ajibola Ademola (olutosin.ademola@taltech.ee)

This work was supported in part by the “Information and Communications Technology (ICT) Program” through the European Union via the European Social Fund, in part by the Innovation Fund Denmark under the Project “Green Roll-on/Roll-off (RORO) Shipping through Digital Innovation (ROROGREEN)” under Grant 0177-00022B, and in part by the Estonian Research Council under Grant PRG658.

ABSTRACT Scene texts serve as valuable information for humans and autonomous systems to make informed decisions. Processing scene texts poses significant difficulties for computer systems due to several factors, primarily due to variations in image characteristics. These factors make it very challenging for computer systems to accurately detect and interpret scene texts, despite being easily understandable to humans. To address this problem, scene text detection and recognition methods leverage computer vision and/or deep learning methods. Deep learning methods require substantial resources, including computing power, memory, and energy. As such, their use in real-time embedded applications, particularly those that run on integer-only hardware, is very challenging due to the resource-intensive nature of these methods. In this paper, we developed an approach to address this challenge and to showcase its effectiveness, we trained end-to-end models for shipping container number detection and recognition. By doing so, we were able to demonstrate the accuracy and reliability of our proposed method for processing scene texts on integer-only hardware. Our efforts to optimize the models yielded impressive results. We reduced the model size by a factor of 3.8x without significantly affecting the models’ performance. Moreover, the optimized models were 1.6x faster, and the maximum RAM usage was 6.6x lower than the base models. These results demonstrate the efficiency and practicality of our approach for scene text processing on integer-only embedded hardware.

INDEX TERMS Deep learning model quantization, integer-only hardware, resource-constrained devices, scene text detection, scene text recognition.

I. INTRODUCTION

We carried out a thorough review of journal search and indexing databases to examine current state-of-the-art methods for scene text detection and recognition. Based on our analysis, we found that no prior work has been done to address the challenges of implementing these methods on integer-only embedded hardware. This highlights the significance and novelty of this research work.

The associate editor coordinating the review of this manuscript and approving it for publication was Yongming Li¹.

The emergence of resource-efficient hardware for deep learning applications, which only supports integer-based operations and operates under stringent storage, memory, and computational power constraints, has been a significant development.

The possibility of accurately detecting and recognizing text in natural scene images has created endless use cases in different embedded applications. One predominant area is autonomous systems. Autonomous systems have a wide range of applications and one of the most prominent areas is their use in various tasks that require intelligent decision-making capabilities. These tasks may involve

intelligent navigation, traffic management, parcel sorting, ticketing, natural language translation, and guiding systems, among others.

Scene text detection and recognition techniques are based on computer vision and/or deep learning methods, and deep learning methods are resource intensive in terms of computing power, memory, and energy usage. Consequently, implementing these methods in real-time embedded applications, particularly those that operate on integer-only hardware, can be highly challenging due to these resource requirements. Different methods have been proposed for text detection in natural scenes [1], [2], [3].

Classic methods (i.e., computer vision-based techniques) utilize sliding windows or connected component analysis to detect the region of text [4], [5], [6], [7]. The sliding window uses a window of multiple scales that moves through the receptive field of the image. The receptive regions (i.e., the text region candidates) are cropped and a machine learning classifier such as Support Vector [8], Random Forest [9], or AdaBoost [10], etc., is trained to predict the text candidates.

Connected component analysis utilizes manual filters to extract salient features such as edges, text texture, boundary points, and text color, among others, from images. These features are used to train a machine learning model [11], [12], [13], [14].

Due to the rise in the adoption of deep learning technology influenced by improved computing resources, availability of big data, etc., unparalleled results have been achieved in almost all computer vision-related tasks that require artificial intelligence such as scene text detection, text recognition, image classification, multi-object detection, etc [15].

Deep learning methods outperform computer vision-based methods because distinctive features are automatically learned using kernel filters instead of relying on manually designed filters to extract fundamental features. As the tasks become more complex, such as in the case of scene text where there are variations in light intensity, surface roughness, low-quality images, etc., the effectiveness of hand-crafted filters tends to decrease. This is because these filters are not able to handle the intricacies of such complex tasks, thus, leading to reduced efficiency.

Several deep learning-based algorithms have been proposed for detecting scene text [16], [17], [18], [19], [20]. These methods rely on state-of-the-art region-based convolutional neural network frameworks for object detection. The region proposal network is responsible for computing the objectness score of the region containing the text region using sets of predetermined anchors. Proposed regions, also known as anchors, are cropped and then fed into the fully connected layer to predict the location of the text region.

Other deep learning methods proposed involve the use of state-of-the-art image segmentation algorithms that classify the text using pixels such that the pixels of the regions containing text are classified as the text class and vice-versa [21], [22], [23], [24], [25].

The high computational and memory requirements of these methods make them expensive, which limits their use in embedded applications running on integer-only hardware. Our proposed method for scene text detection and recognition involves using learned features, a quantization technique with offset, and contour-based character extraction. Our method is designed to be resource-aware, making it suitable for use in integer-only hardware where resources such as memory and compute power are limited.

In summary, our main contributions are as follows:

- We introduced an 8-bit quantization technique for text detection and recognition models. This makes it possible to deploy the models on embedded hardware that only supports integer operations, without a notable drop in performance.
- We introduced a quantization bias to the ground-truth labels to offset the quantization-induced error and improve the accuracy of the models.
- We introduced a module specifically for text orientation detection to improve our recognition pipeline's ability to process text that is oriented both vertically and horizontally.

This paper is divided into several sections, each focusing on different aspects of scene text detection and recognition. The first section provides an introduction, which includes a discussion of existing methods and their limitations, as well as the potential use cases for autonomous systems. Additionally, this section highlights our novel contributions to addressing the challenges of deploying these methods on integer-only hardware.

In section two, we describe the problems associated with implementing text detection and recognition models on integer-only embedded hardware. We also explain the novelty of our work and the need for resource-efficient solutions.

Section three provides a comprehensive review of the state-of-the-art methods for scene-text detection and recognition, highlighting the limitations of each approach. In section four, we present our proposed method in detail, which addresses the challenges of deploying scene text detection and recognition models on integer-only embedded hardware.

Section five discusses the dataset used in our experiments, its source, and the development hardware we used. In section six, we present the results of our experiments in detail. Finally, section seven provides a concluding discussion on the need for resource-aware text detection and recognition, the effectiveness of our proposed method, and a summary of the results achieved.

II. PROBLEM STATEMENT

There are numerous potential applications for scene text detection and recognition in real-time embedded systems. In this section, we will showcase a case study to illustrate this point. In Fig. 1, there are different trucks carrying shipping containers. The containers have unique identification numbers, known as cargo identification numbers, which consist of both numbers and letters.

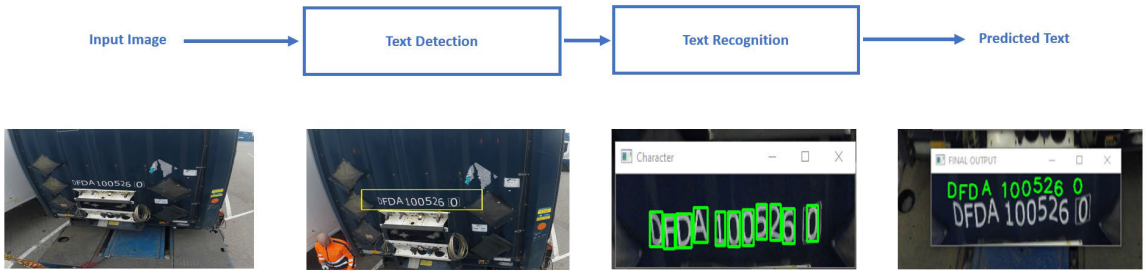


FIGURE 1. Text detection and recognition stages involved in textual information extraction in natural scene images.

Our goal is to efficiently and accurately track every container being transported from the port terminal to the decks of the ship. This ensures that each container, regardless of its size or type, is placed on the designated deck. This objective arises from the need for proper stowage management, which is critical for ensuring the safety of the crew and the successful delivery of the transported containers containing valuable goods.

Identifying containers by their unique cargo identification number presents a challenge in text detection and recognition. While we have reviewed existing methods, none of them meet the specific needs of our use case. Our requirements are particularly strict, as we need a solution that is compatible with integer-only hardware and efficient in terms of storage, computational power, and memory usage.

III. STATE OF THE ART

Scene text recognition methods rely on text detection algorithms. As such, the accuracy depends on how accurately the region of interest is estimated. In this section, we will discuss the state-of-the-art methods for scene text detection and recognition.

As discussed in the introduction section, scene text detection and recognition methods are based on two techniques—computer vision [4], [5], [6] and deep learning [17], [19], [25]. Deep learning methods have been proven to outperform computer vision-based approaches [26], [27], [28], therefore, our work focuses on deep learning-based techniques.

A. TEXT DETECTION

Jaderberg et al. [16] proposed a single pipeline for text detection and recognition. The detection module in their approach relies on a region proposal network. Another method, DeepText [17], utilizes a unified framework that combines a convolutional neural network for region proposal and detection. The region proposal component in DeepText employs an inception module. In [18], the authors used Faster R-CNN for detecting multi-orientation text. Faster R-CNN also incorporates a region proposal network.

Tian et al. [19] introduced CTPN (Connectionist Text Proposal Network), a text proposal network that combines convolutional neural network (CNN) and recurrent neural

network (RNN) with an anchor mechanism for fixed-width proposals. Zhang et al. [21] combined a Fully Connected Network (FCN) with text line hypotheses to detect multi-oriented text. In [22], scene text detection was approached as a segmentation problem, utilizing holistic and multi-channel prediction.

TextEdge [24] implemented a multi-oriented FCN scene text detector that employs region segmentation and edge classification. Zhou et al. [25] introduced EAST, an Efficient and Accurate Scene Text detector, which utilizes a fully convolutional network for scene text detection. Rong et al. [29] proposed a dense text localization network combined with context reasoning for scene text retrieval.

B. TEXT RECOGNITION

Jaderberg et al. [16] introduced deep convolutional neural networks for word-level recognition. Their approach differs from our work, which employs a character-based classifier for scene text recognition. In [26], an end-to-end text spotting method was proposed, utilizing a convolutional recurrent neural network. This unified pipeline requires both ground truth labels for the scene text and bounding box labels.

Bagi et al. [30] introduced a lightweight text spotter that utilizes a lightweight deep neural network for word-level recognition. Cao et al. [31] employed a fully convolutional neural network with an attention module for detecting small text. In [29], the authors utilized a recurrent neural network for the recognition module.

Liu et al. [32] introduced an adaptive bezier-curve network for end-to-end text spotting. The text spotter was further quantized with different bit widths to enhance the network's inference time. However, the emphasis was not placed on the model size and peak runtime memory of the model.

Previous studies have shown that an end-to-end scene text detection and recognition system can employ a single pipeline for both tasks [33], [34], [35]. However, to create a resource-efficient text detection and recognition model suitable for hardware limited to integer operations, certain requirements need to be fulfilled.

Firstly, the model should be lightweight, typically ranging from a few kilobytes to megabytes in size. Secondly, it should have a small memory footprint, typically a few kilobytes to megabytes, to ensure compatibility with the device's capacity.

Finally, the model must be optimized to exclusively support integer-based operations, aligning with the hardware's limitations.

The existing state-of-the-art methods are not well-suited for implementation on integer-only hardware, such as Edge TPUs or microcontrollers. In order to address this challenge, we propose a deep learning-based method that is specifically tailored for such hardware. Our approach takes advantage of learned features, utilizes a quantization technique with offset, and integrates contour-based character extraction.

By being resource-aware, our method is specifically designed to be suitable for integer-only hardware, where limitations in resources such as memory and compute power are prevalent. This resource awareness allows our method to optimize the utilization of available resources, making efficient use of the limited memory and computational capabilities of integer-only hardware. Thus, our method offers a viable solution for enabling effective text detection and recognition on integer-only embedded hardware.

IV. PROPOSED METHOD

A. OVERALL ARCHITECTURE

Scene text recognition methods typically follow a two-stage approach, consisting of text detection and recognition stages, as depicted in Fig. 1. During the text detection stage, the system localizes the region of the text in the image by determining the bounding box coordinates. This stage is of utmost importance as the subsequent recognition stage heavily relies on accurate text detection.

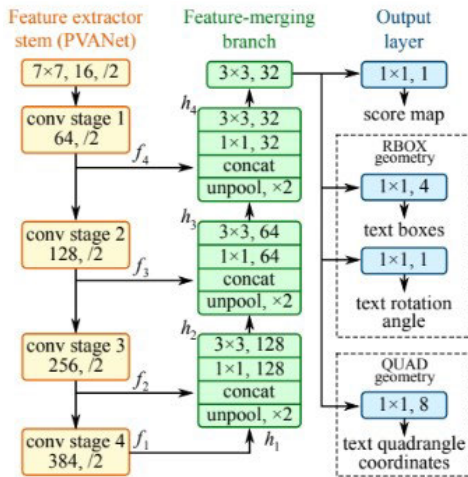


FIGURE 2. The original EAST architecture [25].

B. TEXT DETECTION

Our text detection method is architecturally inspired by the EAST (Efficient and Accurate Scene Text) model [25]. EAST, known as the Efficient and Accurate Scene Text Detector, utilizes a fully convolutional neural network to predict the region of interest where text is present. EAST lacks a

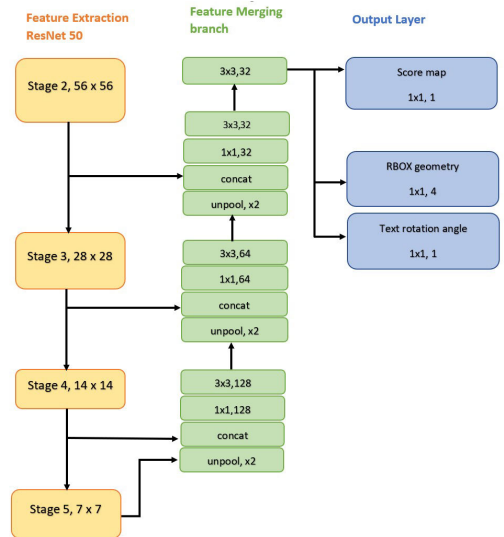


FIGURE 3. The modified EAST architecture using ResNet50 [36] as the base network for the extraction of features.

recognition module. We selected the EAST architecture due to its excellent suitability for our specific use case. Moreover, the EAST architecture seamlessly integrates into our pipeline, as illustrated in Fig. 1.

Several factors influence the suitability and effectiveness of scene text detectors in different applications, and the characteristics and type of the scene text are particularly influential. The architecture consists of three stages: feature extraction, feature merging, and output generation, as illustrated in Fig. 2.

In our modified architecture, we opted for ResNet-50 [36] as the base network for feature extraction, deviating from the original EAST architecture that employed PVANET [37], as depicted in Fig. 3. We made this selection for the following reasons:

- It is faster because it uses a 1×1 kernel filter in its bottleneck design. This design reduces the number of matrix multiplication and network parameters, therefore, reducing the time it takes during propagation.
- ResNet-50 uses a global average pooling rather than fully connected layers. Thus, reduces the size of the model.
- ResNet-50 generalized well on our dataset compared to VGG16 and VGG19.

ResNet-50 is composed of 50 layers, which are divided into five stages of convolution blocks. Fig. 4 illustrates this architecture. The first stage contains a convolution block with 64 filters of size 7×7 and a stride of 2, as well as a max pooling layer with a stride of 2. The input image size is “320 px \times 320 px.” The second stage comprises three sets of three convolution blocks. These blocks consist of 64 filters of size 1×1 , 64 filters of size 3×3 , and 512 filters of size

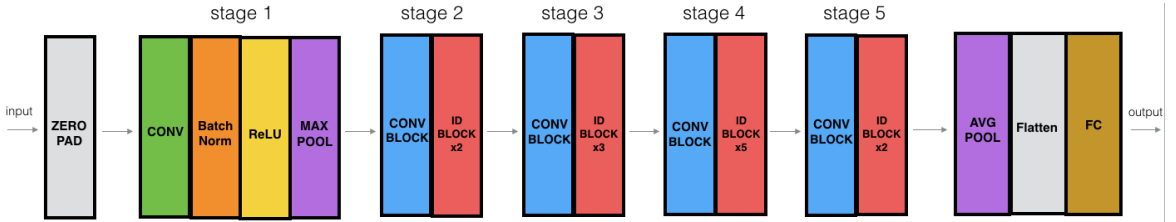


FIGURE 4. The architecture of ResNet50 used as the base network [36].

1×1 . The third stage contains four sets of three convolution blocks.

These blocks consist of 128 filters of size 1×1 , 128 filters of size 3×3 , and 512 filters of size 1×1 . The fourth stage consists of six stacks of three convolution blocks. These blocks consist of 256 filters of size 1×1 , 256 filters of size 3×3 , and 1024 filters of size 1×1 . The fifth stage consists of three stacks of three convolution blocks. These blocks consist of 512 filters of size 1×1 , 256 filters of size 3×3 , and 2048 filters of size 1×1 . The feature merging stage uses the intermediate output of each ResNet-50 stage to reduce the computational complexity of processing all merged features at once, as shown in Fig. 3.

The output of each stage is upsampled so that the output size (i.e., the feature map size) will be of the same size as the input of the stage for concatenation along the channel of the feature maps. 1×1 and 3×3 kernel filters are applied. This is repeated for the other stages. A 3×3 kernel filter is applied to the output of the last upsampled stage which serves as the input of the output stage. The output stage consists of a series of 1×1 kernel filters to produce the confidence score and the coordinates of the region of interest of the text, as shown in Fig. 3.

Our approach focuses primarily on obtaining two key features: the confidence score of text presence, represented by the score map, and the coordinates of the corresponding bounding boxes. These bounding boxes can correspond to horizontal or vertical text regions, as depicted in Fig. 5.

Accurately determining the bounding box type is a crucial aspect of our method. We achieve this by utilizing the bounding boxes generated at the output stage. The precise estimation of the bounding box type plays a pivotal role in the subsequent text recognition stage. It assists in correctly identifying the first and last characters of a word, which is essential for the reconstruction of the words.

C. QUANTIZATION

The parameters of the text detection model are typically represented using 32-bit full-precision floating point values. However, when it comes to quantization for integer-only hardware, text detectors can be highly sensitive to dynamic quantization, where only the model weights are integers, and even more sensitive to full integer quantization, where all parameters, including weights, biases, and activations, are



FIGURE 5. The horizontal and vertical text orientations.

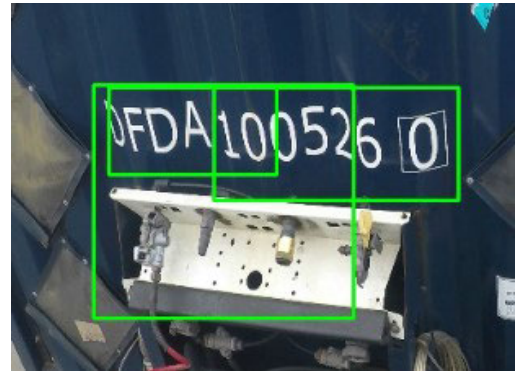


FIGURE 6. Multiple bounding boxes overlapping the scene text caused by quantization-induced error.

integers. To address this challenge, we introduced a quantization offset during the generation of ground-truth labels.

The purpose of the tolerance is to account for the error introduced by quantization, as illustrated in Fig. 6. To ensure compatibility with integer-only hardware, we applied quantization to the text detection model using an 8-bit symmetric signed integer quantizer.

The quantizer takes a 32-bit input float tensor X_f (e.g., the weight matrix of the model), and each parameter is quantized to an 8-bit signed integer using both “equation (1),” and “equation (2).”

$$m_f = \frac{2^7 - 1}{2 \max(|X_f|)}, \quad (1)$$

$$q_{8bit} = \text{round}(m_f X_f). \quad (2)$$

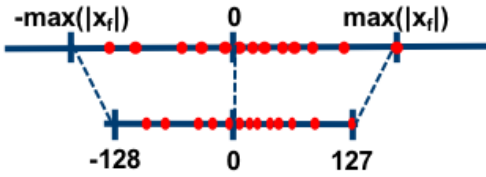


FIGURE 7. The mapping of floating point weight values to 8-bit quantized signed integer representations.

where m_f is the scaling factor and q_{8it} is the quantized output whose range is limited by the absolute value of X_f as shown in Fig. 7.

D. TEXT RECOGNITION

The text recognition stage relies on the output of the text detection model as described in Fig. 1. The text recognition pipeline has two phases (the preprocessing and recognition phases) as shown in Fig. 8.



FIGURE 8. The two stages involved in text recognition.

The preprocessing phase determines the type of bounding box (i.e., the text orientation) using the text coordinates produced by the detection model. This phase extracts the region of interest from the image and removes unwanted contours.

The extraction method is based on a contour-based extraction algorithm that we developed. This algorithm computes the contour of each character, discards unwanted contours, and uses the computed contours to extract the characters.

The characters that have been extracted from the detected text region are then inputted into the text recognition model. The architecture of our lightweight text recognition model incorporates convolutional and dense blocks, which are outlined in detail in Table 1. After the individual characters are predicted, they are aggregated and combined to form the complete recognized text, as shown in Fig. 1.

To ensure compatibility with integer-only hardware, the text recognition model undergoes full quantization using the 8-bit symmetric signed integer quantizer. This quantization process is defined by “equation (1)” and “equation (2)”.

V. EXPERIMENTS

In our experiment, we utilized a dataset comprising 2000 images. Out of these, 1500 images were allocated for training the text detection model, while the remaining 500 images were reserved for testing purposes. It is important to note that the images used in this experiment are proprietary and specifically developed for this project.

The input images for the text detection model were standardized to a size of 320×320 pixels. The dataset consists of various images of containers, each displaying their unique cargo identification number, as depicted in Fig. 1.

For the purpose of training our text recognition model, we extracted a total of 8750 images. These images were

TABLE 1. Architecture of the text recognition model.

Layer Type	Output Size	Parameters
CONV2D	(None, 64, 64, 32)	864
BN	(None, 64, 64, 32)	96
MAXPOOL2D	(None, 32, 32, 32)	0
CONV2D	(None, 32, 32, 64)	18432
BN	(None, 32, 32, 64)	192
MAXPOOL2D	(None, 16, 16, 64)	0
CONV2D	(None, 16, 16, 64)	36928
BN	(None, 16, 16, 64)	192
FLATTEN	(None, 16384)	0
DENSE	(None, 64)	1048576
BN	(None, 64)	192
DENSE1	(None, 35)	2240
ACTIVATION	(None, 35)	0
TOTAL PARAMETERS		1107712

resized to a dimension of 64 pixels by 64 pixels. Each image contained one of the 35 uppercase characters, including numbers (0-9) and letters (A-Z) excluding ‘O’. There were precisely 250 images per character, resulting in a well-balanced dataset.

Out of the extracted images, we allocated 7000 for training the text recognition model, while the remaining 1750 images were set aside for testing purposes.

To ensure compatibility with our desired integer-only model, we selected the Google Coral Development Board as the target hardware. This board is equipped with Quad Cortex-A53 and Cortex-M4F processors, along with an Edge TPU coprocessor. Additionally, it provides 1 GB of RAM and 8 GB of flash memory [38].

Our text detection and recognition models were trained until no further improvements in performance were observed. Nevertheless, we are unable to deploy these models on the target hardware due to its support for only integer-based operations, as well as the strict requirements of our application, which include a small model size footprint, fast inference, high accuracy, efficient peak RAM usage, and computational efficiency. Therefore, further optimization is necessary to meet these requirements.

Quantization plays a significant role in the performance of text detection and recognition models. It refers to the process of reducing the precision of numerical values in a model, typically from floating-point to integer representations. However, quantization can introduce errors and affect the accuracy of the models.

To overcome this challenge, we introduced a quantization offset to the ground-truth labels. This offset is designed to compensate for the errors induced by quantization, ensuring that the model’s predictions align closely with the original floating-point values.

By incorporating the quantization offset, we aim to minimize the impact of quantization on the performance of our text detection and recognition models. This approach allows us to achieve a balance between model optimization for integer-only hardware and preserving the accuracy and reliability of the model’s predictions.

We applied quantization to both the text detection and recognition models, reducing the precision of the model’s parameters such as weights, biases, and activations. Specifically, the parameters were converted from their original 32-bit floating-point representation to 8-bit signed integer representations.

By quantizing the models, we aimed to make them compatible with integer-only hardware and improve their efficiency in terms of memory usage and computational cost. Quantization helps to reduce the model size and allows for faster inference, making it suitable for resource-constrained environments such as edge TPUs or microcontrollers.

The performance of the quantized models was evaluated by measuring their accuracy and overall effectiveness using five key evaluation metrics as described in Table 2 and Table 3. These metrics include the model performance, peak RAM footprint, model size, computational cost, and inference time.

The evaluation considered the performance metrics outlined in Table 2 and Table 3, allowing us to analyze and compare the impact of quantization on various aspects of the model’s performance.

By examining these metrics, we gained deep insights into the trade-offs and improvements achieved through the quantization process, enabling us to make informed decisions regarding the suitability of the models for resource-constrained hardware.

TABLE 2. Performance evaluation metrics for validating quantized model applicability.

Metrics	Text Detection		Text Recognition	
	Original	Quantised	Original	Quantised
Model Size (MB)	96.21	24.83	0.88	0.23
Inference Time (ms)	2356.00	1450.77	3.59	2.18
Computational Cost (MFLOP)	15072.50	0	20.20	0
Peak RAM Usage (MB)	286.23	40.63	5.04	3.29

TABLE 3. Performance evaluation metrics for validating quantized model applicability.

Metric	Text Detection		Text Recognition	
	Original	Quantised	Original	Quantised
Model Performance (%)	25.51	26.23	99.73	99.62

VI. RESULT AND DISCUSSION

Recognizing text in natural scenes is a challenging task due to several factors, including variations in image quality, diverse device types, varying lighting conditions, different text orientations, and the presence of clustered text in scene images. The accurate prediction of text heavily relies on the performance of text detection methods.

It is crucial to highlight that the effectiveness of text detection algorithms significantly impacts the accuracy and precision of text recognition methods. Therefore, ensuring high-quality text detection is essential for achieving reliable and robust text recognition results.

To assess the suitability of the quantized models for our intended purpose, we conducted a comprehensive evaluation that considered various key performance metrics. These

metrics are essential in determining the applicability of the models on the target hardware.

The suitability of the quantized models was evaluated by measuring their accuracy and overall effectiveness in text detection and recognition tasks. Additionally, we assessed the peak RAM usage, which indicates the maximum amount of memory consumed by the models during operation. Model size, another important metric, reflects the storage requirements of the models.

Furthermore, we analyzed the computational cost associated with running the quantized models, considering factors such as the number of operations performed and the processing power required. Lastly, we measured the inference time, which indicates the speed at which the models can process input data and provide output.

By evaluating these performance metrics, we gained valuable insights into the practicality and efficiency of the quantized models for deployment on resource-constrained devices, especially integer-only hardware. This information is crucial for designing effective and optimized solutions that meet the requirements of our target hardware.

To conduct a comprehensive comparison between the base models and their quantized counterparts, we utilized the key performance indicators presented in Table 2 and Table 3. These indicators were derived from a series of experiments conducted using diverse sample data, ensuring a representative evaluation.

The results presented in Table 2 and Table 3 are derived from a thorough evaluation conducted through multiple experiments using diverse sample data. This rigorous approach of averaging the performance metrics over various experiments enhances the reliability and validity of the reported findings.

By using different data samples, we obtain a comprehensive evaluation that provides a more accurate representation of the models’ performance. This ensures that the conclusions drawn from the comparison between the base models and their quantized counterparts are robust and applicable in real-time embedded applications.

The model size refers to the amount of flash memory required to store the model’s parameters, such as weights and biases. By default, the weights are stored using a 32-bit full-precision float. In our approach, we applied an 8-bit symmetric quantizer, as described in Figure 7 and Equations (1) and (2), to both the text detection and recognition models. As a result, we achieved a 3.87x reduction in the flash size required to store the quantized text detection model.

Similarly, the quantized text recognition model demonstrated a 3.82x reduction in model size compared to the uncompressed text recognition model, as indicated in Table 2. Notably, the quantized models maintained their performance, as evidenced by the results presented in Table 3.

We evaluated the text detection model’s performance using the mean loss metric, which is a combination of the dice and intersection over union (IoU) losses. The lower the mean loss value, the better the model’s performance.



FIGURE 9. End-to-end text detection and recognition results of our proposed method.

The quantized text detection model demonstrated a 2% increase in mean loss compared to the base model. On the other hand, the quantized text recognition model showed no significant decrease (only a 0.11% decrease) in performance despite having undergone significant model compression.

The speed of a model during inference is affected by multiple factors, including but not limited to the number of reads and write operations, memory bit width, and types of operations performed. We achieved an improvement of approximately 1.65x in model speed for both quantized models.

In real-time embedded applications, the availability of random access memory (RAM) is crucial for the application's smooth operation without interruptions or delays. RAM is used to store dynamic data that the application requires to function properly.

Deep learning models, such as our base text detection and recognition models, are computationally expensive, especially in terms of RAM resource usage. As indicated in Table 2, the text detection model requires at least 286.23 MB of RAM, while the text recognition model requires at least 5.04 MB. This results in a total RAM requirement of 291.27 MB for the end-to-end pipeline.

Our proposed method enabled us to achieve a significant reduction in RAM usage for the quantized models, resulting in a total of only 43.92 MB of RAM required. This represents a compression factor of 6.63x when compared to the RAM requirements of the base models.

We need to acknowledge a limitation of our proposed method, which is its applicability to less clustered text in scene images. This limitation arises from the need to introduce a quantization bias when preparing the ground-truth labels to compensate for the quantization-induced error.

It's important to note that scene text can vary greatly, and our method may not be suitable for all types of scene text.

VII. CONCLUSION

The increasing utilization of deep learning technology in computer vision tasks owes to a multitude of factors, including advancements in computing power, the availability of vast datasets, and the development of sophisticated algorithms.

Deep learning technology has brought about remarkable breakthroughs, especially in the domain of scene text detection and recognition. The process involves the precise localization of text regions within scene images and subsequent identification of the text contained within these regions.

Scene text detection and recognition have become pronounced due to the rise in the number of portable and embedded devices. These devices are capable of running different intelligent applications. Some of these applications require understanding textual information in scene images for decision-making. Such applications include an intelligent transportation system, text-to-speech, auto navigation, object detection, etc.

The emergence of resource-efficient hardware for deep learning applications, that only supports integer-based operations and operates under stringent constraints on storage, memory, and computational power, has been a significant development.

The current state-of-the-art methods for scene text detection and recognition rely heavily on deep learning approaches that demand significant resources, such as computing power, memory, and energy. As such, the implementation of these methods in real-time embedded applications, especially those

operating on integer-only hardware, poses a considerable challenge.

We developed a resource-efficient method to tackle this issue. To demonstrate its effectiveness and suitability for integer-only hardware, we trained end-to-end models specifically designed for detecting and recognizing shipping containers. Subsequently, these models were deployed on the target hardware.

We demonstrated the accuracy and reliability of our proposed method for processing scene texts on this piece of hardware. Our efforts to optimize the models yielded impressive results as shown in Table 2 and Table 3.

Our optimization efforts resulted in a significant reduction in model size, achieving a compression factor of 3.8x while maintaining comparable performance to the base models. Additionally, the optimized models exhibited a 1.6x increase in speed, accompanied by a substantial decrease in maximum RAM usage by a factor of 6.6x compared to the original models. These results highlight the efficiency and feasibility of our approach for processing scene text on integer-only embedded hardware.

REFERENCES

- [1] D. Cao, Y. Zhong, L. Wang, Y. He, and J. Dang, "Scene text detection in natural images: A review," *Symmetry*, vol. 12, no. 12, p. 1956, Nov. 2020, doi: [10.3390/sym12121956](#).
- [2] X. Li, J. Liu, and S. Zhang, "Text recognition in natural scenes: A review," in *Proc. Int. Conf. Culture-Oriented Sci. Technol. (ICCST)*, Oct. 2020, pp. 154–159, doi: [10.1109/ICCST50977.2020.00036](#).
- [3] B. Zhi-Cheng, L. Qing, C. Peng, and G. Li-Qing, "Text detection in natural scenes: A literature review," *Chin. J. Eng.*, vol. 42, no. 11, pp. 1433–1448, 2020, doi: [10.13374/j.issn2095-9389.2020.03.24.002](#).
- [4] C. Gopalan and D. Manjula, "Sliding window approach based text binarisation from complex textual images," 2010, *arXiv:1003.3654*.
- [5] K. Wang and S. J. Belongie, "Word spotting in the wild," in *Proc. Eur. Conf. Comput. Vis.* Glasgow, U.K.: Springer, Sep. 2010, pp. 591–604.
- [6] J. Fabrizio, B. Marcotegui, and M. Cord, "Text detection in street level images," *Pattern Anal. Appl.*, vol. 16, pp. 519–533, Nov. 2013.
- [7] T. He, W. Huang, Y. Qiao, and J. Yao, "Text-attentional convolutional neural network for scene text detection," *IEEE Trans. Image Process.*, vol. 25, no. 6, pp. 2529–2541, Jun. 2016.
- [8] Y. C. Wei and C. H. Lin, "A robust video text detection approach using SVM," *Exp. Syst. Appl.*, vol. 39, no. 12, pp. 10832–10840, Sep. 2012.
- [9] Y. Zhang, C. Wang, B. Xiao, and C. Shi, "A new method for text verification based on random forests," in *Proc. Int. Conf. Frontiers Handwriting Recognit.*, Sep. 2012, pp. 109–113.
- [10] S. M. Hanif and L. Prevost, "Text detection and localization in complex scene images using constrained AdaBoost algorithm," in *Proc. 10th Int. Conf. Document Anal. Recognit.*, Barcelona, Spain, 2009, pp. 1–5.
- [11] X. Zhao, K.-H. Lin, Y. Fu, Y. Hu, Y. Liu, and T. S. Huang, "Text from corners: A novel approach to detect text and caption in videos," *IEEE Trans. Image Process.*, vol. 20, no. 3, pp. 790–799, Mar. 2011.
- [12] Q. Ye, Q. Huang, W. Gao, and D. Zhao, "Fast and robust text detection in images and video frames," *Image Vis. Comput.*, vol. 23, no. 6, pp. 565–576, Jun. 2005.
- [13] W. Wu, X. Chen, and J. Yang, "Detection of text on road signs from video," *IEEE Trans. Intell. Transp. Syst.*, vol. 6, no. 4, pp. 378–390, Dec. 2005.
- [14] Y. Zhu, C. Yao, and X. Bai, "Scene text detection and recognition: Recent advances and future trends," *Frontiers Comput. Sci.*, vol. 10, no. 1, pp. 19–36, Feb. 2016.
- [15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, vol. 1. Red Hook, NY, USA: Curran Associates, 2012, pp. 1097–1105.
- [16] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Reading text in the wild with convolutional neural networks," *Int. J. Comput. Vis.*, vol. 116, pp. 1–20, Jan. 2015.
- [17] Z. Zhong, L. Jin, S. Zhang, and Z. Feng, "DeepText: A unified framework for text proposal generation and text detection in natural images," 2016, *arXiv:1605.07314*.
- [18] Y. Jiang, X. Zhu, X. Wang, S. Yang, W. Li, H. Wang, P. Fu, and Z. Luo, "R2CNN: Rotational region CNN for orientation robust scene text detection," 2017, *arXiv:1706.09579*.
- [19] Z. Tian, W. Huang, T. He, P. He, and Y. Qiao, "Detecting text in natural image with connectionist text proposal network," in *Proc. Eur. Conf. Comput. Vis.* Amsterdam, The Netherlands: Springer, Sep. 2016, pp. 56–72.
- [20] D. Xiang, Q. Guo, and Y. Xia, "Robust text detection with vertically-regressed proposal network," in *Proc. Eur. Conf. Comput. Vis.* Amsterdam, The Netherlands: Springer, 2016, pp. 351–363.
- [21] Z. Zhang, C. Zhang, W. Shen, C. Yao, W. Liu, and X. Bai, "Multi-oriented text detection with fully convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 4159–4167.
- [22] C. Yao, X. Bai, N. Sang, X. Zhou, S. Zhou, and Z. Cao, "Scene text detection via holistic, multi-channel prediction," 2016, *arXiv:1606.09002*.
- [23] X. Li, W. Wang, W. Hou, R.-Z. Liu, T. Lu, and J. Yang, "Shape robust text detection with progressive scale expansion network," 2018, *arXiv:1806.02559*.
- [24] C. Du, C. Wang, Y. Wang, Z. Feng, and J. Zhang, "TextEdge: Multi-oriented scene text detection via region segmentation and edge classification," in *Proc. Int. Conf. Document Anal. Recognit. (ICDAR)*, Sep. 2019, pp. 375–380.
- [25] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "EAST: An efficient and accurate scene text detector," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 2642–2651, doi: [10.1109/CVPR.2017.283](#).
- [26] K. Wang, B. Babenko, and S. Belongie, "End-to-end scene text recognition," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 1457–1464, doi: [10.1109/ICCV.2011.6126402](#).
- [27] G. Li, "CSNet-PGNet: Algorithm for scene text detection and recognition," in *Proc. 3rd Int. Conf. Comput. Vis., Image Deep Learn. Int. Conf. Comput. Eng. Appl. (CVIDL ICCEA)*, May 2022, pp. 1217–1224, doi: [10.1109/CVIDLICCEA56201.2022.9824815](#).
- [28] A. Khalil, M. Jarrah, M. Al-Ayyoub, and Y. Jararweh, "Text detection and script identification in natural scene images using deep learning," *Comput. Electr. Eng.*, vol. 91, May 2021, Art. no. 107043.
- [29] X. Rong, C. Yi, and Y. Tian, "Unambiguous text localization, retrieval, and recognition for cluttered scenes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 3, pp. 1638–1652, Mar. 2022, doi: [10.1109/TPAMI.2020.3018491](#).
- [30] R. Bagi, T. Dutta, and H. P. Gupta, "Cluttered TextSpotter: An end-to-end trainable light-weight scene text spotter for cluttered environment," *IEEE Access*, vol. 8, pp. 111433–111447, 2020, doi: [10.1109/ACCESS.2020.3002808](#).
- [31] Y. Cao, S. Ma, and H. Pan, "FDTA: Fully convolutional scene text detection with text attention," *IEEE Access*, vol. 8, pp. 155441–155449, 2020, doi: [10.1109/ACCESS.2020.3018784](#).
- [32] Y. Liu, C. Shen, L. Jin, T. He, P. Chen, C. Liu, and H. Chen, "ABCNet v2: Adaptive Bezier-curve network for real-time end-to-end text spotting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 11, pp. 8048–8064, Nov. 2022, doi: [10.1109/TPAMI.2021.3107437](#).
- [33] J. Guo, R. You, and L. Huang, "Mixed vertical-and-horizontal-text traffic sign detection and recognition for street-level scene," *IEEE Access*, vol. 8, pp. 69413–69425, 2020, doi: [10.1109/ACCESS.2020.2986500](#).
- [34] C. Zhang, Y. Tao, K. Du, W. Ding, B. Wang, J. Liu, and W. Wang, "Character-level street view text spotting based on deep multisegmentation network for smarter autonomous driving," *IEEE Trans. Artif. Intell.*, vol. 3, no. 2, pp. 297–308, Apr. 2022, doi: [10.1109/TAI.2021.3116216](#).
- [35] Y. Liu, L. Jin, and C. Fang, "Arbitrarily shaped scene text detection with a mask tightness text detector," *IEEE Trans. Image Process.*, vol. 29, pp. 2918–2930, 2020, doi: [10.1109/TIP.2019.2954218](#).

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[37] C. Zaharia, D. Dinu, and A. Caliman, "PVANet optimization for person detection," in *Proc. Int. Conf. Optim. Electr. Electron. Equip. (OPTIM)*, *Int. Aegean Conf. Electr. Mach. Power Electron. (ACEMP)*, May 2017, pp. 959–964, doi: [10.1109/OPTIM.2017.7975094](https://doi.org/10.1109/OPTIM.2017.7975094).

[38] *Google Coral Products Page*. Accessed: Jan. 5, 2023. [Online]. Available: <https://coral.ai/docs/dev-board/datasheet/>



OLUTOSIN AJIBOLA ADEMOLA received the M.Sc. (Eng.) degree from the School of Information Technology, Tallinn University of Technology, Tallinn, Estonia, in 2020. His research interests include deep learning, machine learning, edge AI, intelligent systems, and computational intelligence.



EDUARD PETLENKOV (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computer and systems engineering from the Tallinn University of Technology, in 2001, 2003, and 2007, respectively. He is currently a tenured Full Professor with the Department of Computer Systems, Tallinn University of Technology, and the Head of the Centre for Intelligent Systems. His main research interests include the domain of intelligent control, system analysis, and computational intelligence.



MAIRO LEIER received the Ph.D. degree in computer systems from the Tallinn University of Technology, in 2016. He was a Research Scientist with the Department of Computer Systems, Tallinn University of Technology, where he leads the Embedded AI Research Laboratory. His current research interests include machine learning on embedded systems, optimization techniques, and edge computing.

...

Curriculum Vitae

1. Personal data

Name	Olutosin Ajibola Ademola
Date and place of birth	31 December 1990, Ilorin, Nigeria
Nationality	Nigeria

2. Contact information

Address	Poorise 5, 4-21 Tallinn, Estonia
Phone	+37253564286
E-mail	olutosin.ademola@ttu.ee

3. Education

2020–Present	Tallinn University of Technology, PhD studies
2018–2020	Tallinn University of Technology, School of Information Technologies Communicative Electronics, MSc
2008–2013	University of Ilorin, Department of Electrical and Electornics Engineer- ing Electical/Electronics Engineering, BSc

4. Language competence

English	Fluent
Yoruba	Fluent
Estonian	A1

5. Professional employment

2018–2022	Tallinn University of Technology, Tallinn, Estonia, Software Engineer, Machine Learning
2017–2018	Citrans Global Limited, Lagos, Nigeria, Head of IT
2016–2017	Hausba Smarthomes, Lagos, Nigeria, System Integrator
2014–2015	T-One Technologies, Lagos, Nigeria, IT Support Engineer

Elulookirjeldus

1. Isikuandmed

Nimi	Olutosin Ajibola Ademola
Sünniaeg ja -koht	31.12.1990, Ilorin, Nigeria
Kodakondsus	Nigeria

2. Kontaktandmed

Aadress	Poorise 5, 4-21 Tallinn, Estonia
Telefon	+37253564386
E-post	olutosin.ademola@ttu.ee

3. Hariduskäik

2020– Tänapäev	Tallinna Tehnikaülikool, doktoriõpe
2018–2020	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond Kommunikatsioonielektroonika, MSc
2008–2013	Ilorini Ülikool, elektri- ja elektroonikatehnika osakond elektroonika/elektronikatehnika, BSc

4. Keelteoskus

inglise keel	kõrgtase
yoruba keel	kõrgtase
eesti keel	A1

5. Teenistuskäik

2018–2022	Tallinna Tehnikaülikool, Tallinn, Eesti , Tarkvarainsener, masinõpe
2017–2018	Citrans Global Limited, Lagos, Nigeeria, IT-valdkonna juht,
2016–2017	Hausba Smarthomes, Lagos, Nigeeria, süsteemiintegraator,
2014–2015	T-One Technologies, Lagos, Nigeeria, IT-tugiinsener,

ISSN 2585-6901 (PDF)
ISBN 978-9916-80-320-2 (PDF)