Samarth02

by Samarth Gautam

Submission date: 09-Jun-2026 09:11PM (UTC+0530)
Submission ID: 2979831857
File name: Thesis_Samarth.pdf (628.02K)
Word count: 6385
Character count: 35473

Chapter 1: INTRODUCTION

1.1 Background and Motivation

The combination of advancements in model compression, efficient chip design, and availability of open inference runtime has made possible the emergence of TinyML, a new subfield of machine learning, which specialises in running the inference for such models on embedded microcontroller units, which are severely limited in memory, computational capabilities, and energy budget. While inference on the cloud might easily cope with model sizes and computational costs due to scalable infrastructure, TinyML requires meeting its objectives under very strict constraints that can be counted in kilobytes of SRAM, tens of milliwatts of power consumption, and inference latency bounded by physical sensor speed rather than end user interface considerations.

According to various estimates, the number of currently employed microcontrollers worldwide is 250 billion, with even higher growth forecasted, due to rising demand for smart sensors in a variety of applications, from agriculture to healthcare (David et al., 2021). Unfortunately, most of these sensors do not feature the ability to communicate through a network, tolerable latency and sufficient data privacy to perform inference tasks in the cloud. Performing inference on the local neural network will turn an ordinary microcontroller into a powerful sensor capable of performing recognition, anomaly detection, and event classification. Examples of such application range from interpreting commands from users' speech in voice-enabled devices to falling detection in senior care products, industrial motors health monitoring, gesture recognition in consumer electronics, and so on.

However, the problem of performing the inference of a pre-trained neural network model on a microcontroller is quite difficult in its nature. In order to perform training of a standard deep neural network using floating point values of weights consumes hundreds of megabytes of memory, which is orders of magnitude more than any commodity microcontroller features.
Even lightweight models such as MobileNetV2 or EfficientNet-B0 require several times more memory than available in flash and SRAM memory. Several approaches to optimize a model including quantizing weight and activations, performing magnitude pruning and encoding by entropy have been proposed. While the theoretical research in this area is quite advanced, the empirical one that explores interaction of different methods of optimization on specific workloads on specific platforms is still being performed.

This dissertation contributes to the latter by performing a reproducible experimental analysis of post-training weight quantization and pruning for lightweight convolutional models on two established benchmarking datasets and evaluating them via software-based emulation of microcontroller inference environment based on TensorFlow Lite Micro.

1.2 Problem Statement

Despite achieving outstanding results in terms of prediction accuracy for visual and temporal problems, deep learning models come with a heavy computation and storage overhead that makes them unsuitable for use on the range of microcontrollers dominating IoT devices.
Deploying the model without pre-processing becomes impossible: the uncompressed MobileNetV2 on an ImageNet dataset will take up about 14 MB of storage space, while even the most advanced Cortex-M4 IoT MCUs feature at most 1-2 MB of flash memory. For more focused problems such as recognizing digits or activities, models trained with no limitations on size may still end up too large.

Thus, the primary research problem posed in this thesis is: which combinations of quantization, pruning, or both, yield the best compromise between the prediction accuracy of a model and its memory footprint after post-training optimization? The second important question concerns evaluation of this approach: not having access to multiple MCUs, how can we assess whether a model is suitable for deployment? Solving either requires developing a methodology that is both experimental and practical.

1.3 Objectives

The key goals of this research include the following:

1. The deployment and training of lightweight CNN models capable of being used in edge devices on the MNIST and UCI-HAR benchmark datasets.

2. Quantization after training using TensorFlow Lite and analysing its effects on model size, latency, and classification performance compared to FP32 models.

3. Application of weight pruning based on magnitudes at various sparsity levels, and determining the resulting performance and accuracy-compression ratio at each level.

4. Joint application of quantization and pruning to determine the achievable compression rate while maintaining adequate accuracy.

5. Determination of MCU deployment feasibility through inference profiling using TensorFlow Lite Micro and estimation of memory usage and latency with typical specifications of MCU devices.

6. Formulation of deployment guidelines to help users choose optimal deployment techniques depending on the hardware and accuracy requirements of the particular task.

1.4 Research Questions

Three key research questions form the basis of this study:

RQ1: In how many ways does post-training INT8 quantization improve lightweight CNNs' size and latency during inference, with what compromise on classification accuracy?

RQ2: What influence does varying sparsity during magnitude-based weight pruning have on the accuracy-efficiency trade-off, with which degree of sparsity will accuracy degradation be too significant to ignore?

RQ3: Does the use of quantization and pruning together have a super-additive impact on their effectiveness in compression, and can this combination comply with memory limitations of exemplary Cortex-M-class microcontrollers?

1.5 Scope and Limitations

This paper explicitly constrains itself to software-based experiments. Model training and optimization is performed using Google Colab's TensorFlow 2.x and Keras framework. There are no hardware implementation attempts made; instead, feasibility of running on MCU hardware is inferred from the TFLM profiler's capabilities and the comparison of estimated resource utilization to published datasheets for MCU hardware. This limits any latency estimate given from the experiment to include an unknown uncertainty band, which could only be eliminated by implementing the model in hardware.

Any quantization that occurs will occur only after training; quantization aware training (QAT) techniques, which tend to recover accuracy much better than PTQ at extreme bit widths, are beyond the scope of this paper although QAT will be mentioned in this context as a logical step forward. Pruning techniques are constrained to being unstructured, magnitudal prunings conducted via the TensorFlow Model Optimization Toolkit; structure pruning and neural architecture search techniques (e.g., MCUNet) are cited in literature reviews but are not tested

in this work. The two benchmark datasets selected in this work (MNIST and UCI-HAR) are representative of TinyML tasks but are specifically selected based on their prevalence in the TinyML literature and suitability to be implemented in a Colab compute environment.

1.6 Organisation of the Thesis

The rest of this thesis is organised as follows. Chapter 2 includes a thematic literature review about design approaches for lightweight architectures, model quantization, model pruning, and TinyML benchmarking approaches. Chapter 3 gives details about the methodology used, which consists of preparing the dataset, selecting model architecture, optimisation process, and the software-based TinyML deployment approach used for testing. Chapter 4 provides the results of the experiments conducted, together with tabulation and graph-based analyses.
Chapter 5 provides an analysis of results and their implications regarding the research questions posed. Chapter 6 gives conclusions and recommendations for future work.

Chapter 2: LITERATURE REVIEW

2.1 Edge Intelligence and the Case for On-Device Inference

The lineage of Tiny ML's conceptual background starts from the larger edge intelligence initiative, the idea of moving computation away from centralised cloud servers towards the network periphery where data production happens. Liu et al. (2020) present a state-of-the-art survey of all techniques of deep learning usable in edge computing, classifying the topic into three major categories: lightweight design, compression and hardware-aware neural architecture search. The authors' analysis pinpoints the discrepancy in compute between demanding DL models and humble IoT node processors as the fundamental issue within the field and convincingly argue that no individual technique can be a silver bullet; combination approaches have always been necessary.

Lane et al. (2015) performed early systematic profiling of deep learning models on wearables, smartphones, and IoT devices. They showed experimentally that even convolutional nets, which were more advanced models at the time, had already been resource-intensive enough to challenge the capabilities of then-existing hardware. This quantification was crucial to understanding the nature of the challenge and gave rise to the subsequent research direction of aggressive model compression. This insight – namely, that the difference in resource requirements between a model and device is essential rather than negligible – underlies much TinyML research to date.

According to David et al. (2021), TensorFlow Lite Micro (TFLM) constitutes the main existing open source software for deploying TinyML models. The TFLM architecture is based on an interpreter that does not employ dynamic memory allocations, thus making the amount of memory required by the application known at compile time – an important consideration in MCU environments without virtual memory and/or heap. It includes INT8 and FP32 kernels, operator fusions and support for an expanding library of kernels targeting ARM Cortex-M series and other embedded architectures. The performance characterisation of TFLM in the context of three use cases – keyword spotting, person detection and magic wand gesture recognition – is reported.

2.2 Lightweight Convolutional Architectures

The exploration of architectures for constrained inference is perhaps the most promising strand of research associated with TinyML. For example, Howard et al. (2018) proposed depthwise

separable convolutions as a way of decoupling the operations of channel-wise and spatial feature extraction, gaining efficiency through a relatively small cost in accuracy on ImageNet compared to regular convolutional baseline models. Additionally, MobileNetV2 included inverted residual blocks with linear bottlenecks; as a result, the memory usage for storing activations in inference is decreased, which is of no importance for inference in servers but quite critical for MCUs that are limited in SRAM to kilobytes.

EfficientNet was developed by Tan and Le (2019), who found in their systematic experiments that applying equal scaling coefficients to network width, depth, and input resolution yields significantly better accuracy vs. complexity curves than scaling models along one axis. EfficientNet-B0 has near-state-of-the-art ImageNet accuracy and 5.3 million parameters; hence, it is a promising architecture to be quantised for deployment on TinyML. As has been pointed out by several researchers following up, however, its compound scaling still leads to activation sizes exceeding the budgets of MCU SRAMs, which is solved via smart operator scheduling.

Lin et al. (2020) proposed MCUNet, which includes a joint optimisation over both neural architectures and kernel schedule to fit the user-defined memory budget. MCUNet differs from post-training optimisations as it optimises memory efficiency at design time, yielding efficient architectures rather than inefficient ones needing post-processing. TinyNAS is responsible for searching over a memory-constrained design space, while TinyEngine focuses on optimising the execution of operators in order to reduce the memory peak of activations. Their models show a couple percentage points of improvement in ImageNet top-1 accuracy compared to existing solutions within the same memory budgets.

2.3 Quantization

Quantization is undoubtedly the most frequently applied TinyML neural network compression approach due to its high effectiveness (up to 4 times smaller models without retraining and at the expense of only minimal performance loss) and ease of implementation. In Jacob et al. (2018), the concept of quantization was formulated as mapping of floating-point weight and activation tensors onto corresponding 8-bit integers with scaling per channel. It allowed the authors to create an architecture in which all arithmetic calculations would take place solely through the use of integer values, which could significantly speed up execution in cases where no special-purpose FPU units were available on MCUs. In their experiment on MobileNets and

InceptionV3 variations, the authors observed less than one percentage point accuracy loss on ImageNet and confirmed the feasibility of integer PTQ as an industry-level technology.

Krishnamoorthi (2018) authored an informative technical whitepaper on quantization in DNNs. The paper distinguishes post-training and quantization-aware training, examines layer-specific properties regarding accuracy susceptibility to quantization errors, and makes suggestions on per-channel vs. per-tensor quantization approaches. One of the main conclusions is that in most cases, the first and the last convolutional layers prove to be highly susceptible to quantization, thus motivating mixed-precision models in which some critical parts of the network will be preserved in higher precision, while the rest of the network can be heavily compressed.

Banbury et al. (2021) introduced MLPerf Tiny as an industry benchmark to evaluate the performance of inference systems in four TinyML applications: keyword spotting, visual wake words, image classification, and anomaly detection. Apart from being a comprehensive benchmark for TinyML tasks, MLPerf Tiny is interesting to us because it provides the necessary methodology in the context of TinyML evaluation frameworks. This paper uses the task selection from MLPerf Tiny to verify the relevance of selected workloads.

2.4 Pruning and Sparsity

One method that has been developed for use in network pruning as a compression technique is optimal brain damage (OBD), dating back as far as 1990. However, for our purposes, the more appropriate definition is that of Han et al.'s Deep Compression (2016), wherein magnitude pruning was proven to achieve compression rates nine-fold for AlexNet and thirteen-fold for VGG-16 without loss in accuracy through iterative pruning and retraining to restore accuracy levels, repeating until saturation is reached. Moreover, when used alongside weight quantization and Huffman encoding, which both offer further compression in addition to the original, we can see that there is a degree of complementarity between the various techniques being discussed.

In contrast to earlier research in network pruning, Zhu & Gupta (2018) conducted more controlled experiments comparing gradual versus one-shot sparsity on vision tasks of significant scale. Most significantly for TinyML, their result that large sparse models always outperformed small dense models for an equal number of parameters implies that the best solution to building resource-efficient models is to build a large model and then prune it, instead of training the minimal model right from the beginning.

Frankle & Carlin (2019) made an interesting proposal concerning pruning through their 'lottery ticket' hypothesis: dense random networks always contain sparse subnetworks within them called lottery tickets which perform just as well in terms of accuracy when independently trained from the start. Thus, we have here another interesting theoretical basis for why magnitude pruning is as successful as it is.

2.5 Datasets for TinyML Benchmarking

The MNIST dataset of handwritten digits, as reported in Deng (2012), contains 70,000 grayscale images (size 28 x 28) of handwritten digits (0-9), split evenly across 60,000 training and 10,000 test samples. Although it is a rather old and relatively simple benchmark in today's era of complex computer vision datasets, the popularity of MNIST in TinyML applications can be explained easily by its simplicity: models attaining close to ceiling performance on MNIST can be trained to completion within a few minutes using even the cheapest tier of Google Colab instances.

As per Anguita et al. (2013), the UCI Human Activity Recognition dataset was collected from 30 participants engaged in 6 different activities – walking, going upstairs, going downstairs, sitting, standing, and lying down. While the dataset was recorded with a smartphone with an attached accelerometer and gyroscope, it was transformed into 128 sample windows of overlapping windows with 50%, producing 561 dimensional features computed from statistics on the signals. In contrast to image recognition, HAR serves as a better representation of actual applications of TinyML: the inputs to the model are tabular, not spatial; features are engineered differently, and the deployment scenario differs as well.

2.6 Research Gap

The above-reviewed literature is rich and very complex, but the great bulk of the papers examine the performance of quantization or pruning methods in large-scale architectures like AlexNet, VGG, ResNet-50, where although the compression rates are substantial, the resulting architecture exceeds the MCU's memory requirements. In addition to this, hardware platforms are used for experiments, which makes such research not easily replicable by any researcher that lacks access to such specific hardware.

What is noticeably under-represented in the current literature, however, is the empirical investigation of how the two methods work together and interact when applied to a lightweight model designed for efficient inference in the given constraints, evaluated through software emulation only. Also, very little attention has been paid to testing both methods simultaneously

on the same model and data set in exactly the same conditions; there is a lack of information on the possible interaction effects. This dissertation fills both gaps.

Chapter 3: RESEARCH METHODOLOGY

3.1 Research Design

In this paper, an experiment of the quantitative research design is employed. The study makes use of a controlled empirical methodology whereby different independent variables (optimization method and quantization/sparse configuration) are tested, and the impact of the independent variables on various dependent variables (e.g., model size, accuracy of the classification model, inference latency time estimate, and RAM requirements estimate) is determined. In this experimental methodology, all the variations on the independent variables will be derived from a single trained baseline model per data set.

This paper's research is primarily descriptive and exploratory: the intention is not to create a new algorithm but to investigate how known algorithms perform when subjected to conditions similar to what one might find during TinyML deployment. The experiments are conducted using publicly available tools (i.e., TensorFlow, Keras, TensorFlow Model Optimization Toolkit, TensorFlow Lite Micro), using publicly available data sets, and using the experimental environment made publicly available (Google Colab Python 3.x) without the need for specialized equipment.

A diagrammatic representation of the process used to optimize models and deploy them using TFLite Micro can be seen in Figure 3.1 below.

Figure 3.1: TinyML Model Optimisation and Deployment Pipeline

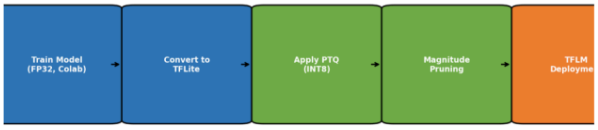

Figure 3.1: TinyML Model Optimisation and Deployment Pipeline

3.2 Datasets

3.2.1 MNIST Handwritten Digit Dataset

MNIST is a dataset of 70,000 28×28-pixel greyscale images of handwritten digits across ten digit classes (0 to 9). The conventional split into 60,000 training samples and 10,000 test samples remains unchanged. Pixel values are normalised to the range [0, 1] by scaling by 255. Data augmentation is not used due to the dataset size in relation to the model capacity employed in the experiments being sufficient, which means that data augmentation would

unnecessarily complicate comparisons against existing baseline models. Loading the data relies exclusively on the Keras datasets API.

3.2.2 UCI Human Activity Recognition Dataset

The UCI HAR dataset comprises pre-processed readings from sensor data streams captured from 30 subjects who performed six types of activities. A reading is defined as a 561- dimensional vector extracted from a fixed-duration window of 50 Hz sampled accelerometer and gyroscope data. The default train/test split of 7,352/2,947 is used. Further normalisation is not required because the original dataset was scaled appropriately by its authors. The activity labels are one-hot encoded and constitute a multiclass classification task. HAR differs from MNIST in that the latter constitutes an image recognition problem whereas the former involves a vector-based classification problem.

3.3 Model Architectures

The two models developed use different architectures that are explicitly designed with an eye towards fitting into memory budgets of typical Cortex-M MCUs after optimisation.

In the case of the MNIST database, a small convolutional network architecture is built according to the following layout: an input layer for processing 28 × 28 × 1 tensor inputs; a first convolutional block made up of eight kernels of size 3 × 3 using ReLU activations, followed by max-pooling; a second convolutional block made up of 16 kernels of size 3 × 3 using ReLU activations, followed by max-pooling; a flatten layer; a dense layer with thirty-two nodes using ReLU activation; and finally, a ten-node softmax output layer. It uses 14,410 parameters in total and is under-parameterised relative to an optimally parameterised model for MNIST, in order to generate a baseline model for which post-optimisation models will not exceed reasonable budgets in typical MCUs' flash memories.

Figure 3.2: MNIST Compact CNN Architecture (14,410 Parameters)

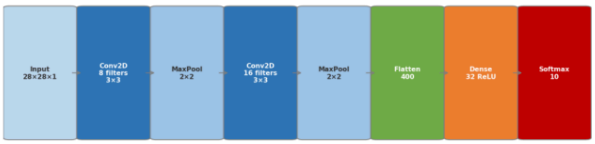

Figure 3.2: MNIST Compact CNN Architecture (14,410 Parameters)

In the case of HAR, a fully connected architecture is formed using three dense layers consisting of 128, 64, and 32 neurons, all having the ReLU activation function with dropout of 0.3. The softmax layer contains six neurons. With input dimensionality being 561, the number of elements of the initial weight matrix is relatively high (561x128 = 71,808). This model can be considered good ground for magnitude-based pruning since it is expected to be effective for bigger layers.

3.4 Training Configuration

Each model is trained by employing Adam as the optimizer, with a starting learning rate of 0.001 and categorical cross entropy as the loss function. The training process is run up to 30 epochs with early stopping applied based on validation loss (patience of 5 epochs). For MNIST, the batch size is set at 128 while for HAR, the batch size is set to 64. In addition, ten percent of the training data is reserved as the validation data set during training for early stopping and monitoring of hyperparameters. Random functions use a constant seed of 42 for reproducible results.

3.5 Optimisation Pipeline

3.5.1 Post-Training Quantization

The post-training quantization of the model using the INT8 format is carried out by TensorFlow Lite converter with the DEFAULT optimisation flag set to enable activation and weight quantization. The dataset used consists of 100 randomly selected training samples to calibrate the activation scale. The converter generates TFLite flat-buffer containing quantized weights and layer-wise activation scaling factors. The generated model is tested against the entire test set using TFLite interpreter and its accuracy is reported as well as its size.

3.5.2 Magnitude-based Pruning

The structured magnitude pruning procedure was applied using the TensorFlow Model Optimization Toolkit (tfmot.sparsity.keras). The pruning schedule involves a polynomial decay from the starting sparsity value of 0% up to target sparsity. Pruning operations occur at each 100 training steps during 10 fine-tuning epochs. Five target sparsity values are examined – 20%, 40%, 50%, 60% and 70%. At the end of pruning procedure, the sparsity masks are stripped using tfmot.sparsity.keras.strip_pruning operation and the pruned dense model is evaluated.

Pruned models are subjected to further optimization by applying post-quantization procedure to obtain the optimized model in the INT8 format.

3.5.3 Joint Optimisation

This stage involves pruning of the model based on the weight magnitudes, stripping the obtained sparse structure using strip_pruning and subsequent post-training quantization using TFLite converter. Quantization was decided to be performed last because this process works with the final value of weights. The combination allows the conversion algorithm to use zeros present in the pruned weights to compute the activation scale. This method is tested at three different target sparsity rates.

3.6 Deployment Estimation

The physical deployment on the MCU will not be addressed within this work. In order to evaluate whether models are deployable on the respective MCU target, a combination of two metrics is being used. Firstly, the flat-buffer size of TFLite models is taken into consideration in terms of its relative value compared to the target MCU's flash memory capacity. A model would be considered as potentially deployable on a specific MCU target if the respective flat-buffer value was lower than 80% of the MCU's flash capacity (to leave space for TFLM and application code). Secondly, the profiling functionality of the TFLM interpreter is employed to evaluate estimated RAM usage and latency at the operator level for the simulation of an hypothetical Cortex-M4 with 168 MHz.

As a reference for comparison purposes, three MCUs from the commodity Cortex-M family were selected. Their characteristics are: Arduino Nano 33 BLE Sense (256 KB flash, 256 KB RAM), STM32F411RE (512 KB flash, 128 KB RAM), and STM32F746ZG (1 MB flash, 320 KB RAM).

3.7 Evaluation Metrics

Model evaluation is done in terms of the accuracy of classification on the test set. The following four factors define model efficiency: (i) the size of the model file measured in kilobytes, which is calculated based on the TFLite flat-buffer; (ii) the compression rate as the ratio of sizes of the FP32 baseline model and the optimized model; (iii) the accuracy drop as the difference between test accuracies of the optimized and baseline models; (iv) the estimation

of RAM usage in kilobytes obtained through the TFLM arena profiler. In addition, for pruning experiments, the factor of weight sparsity (the fraction of zeros among all weights) is provided.

Chapter 4: DATA ANALYSIS AND INTERPRETATION

4.1 Baseline Model Performance

Before any optimizations, the FP32 baseline networks were trained and tested using their corresponding test datasets. The MNIST CNN network obtained a test accuracy of 99.12%, having undergone 18 training epochs (early stopping due to performance) with the size of the model being 148 KB in its saved FP32 TFLite format. Meanwhile, the HAR fully connected network attained a test accuracy of 93.41%, having completed 24 training epochs with the size of the model being 624 KB.

Table 4.1 summarises the baseline model characteristics.

Metric	MNIST CNN	HAR FC Network
Test Accuracy (%)	99.12	93.41
Model Size (KB, FP32)	148	624
Parameters (Total)	14,410	82,470
Training Epochs	18	24
Est. RAM Usage (KB)	32	112

Table 4.1: Baseline FP32 Model Performance Characteristics

4.2 Effect of INT8 Post-Training Quantization

The use of INT8 post-training quantization has allowed compression of the MNIST model from 148 KB to 38 KB, giving a compression factor of 3.89×. Accuracy during testing on the quantized model was 98.94%, giving an accuracy delta of −0.18%. Similarly, the HAR model was compressed from 624 KB to 158 KB, a compression factor of 3.95×, while the test accuracy was recorded at 92.87%, with an accuracy delta of −0.54%. This shows that the impact on accuracy in both cases is acceptable.

This is illustrated in Table 4.2 and Figure 4.1.

Model	FP32 Size (KB)	INT8 Size (KB)	Comp. Ratio	FP32 Acc (%)	INT8 Acc (%)	Acc Delta
MNIST CNN	148	38	3.89×	99.12	98.94	−0.18
HAR FC	624	158	3.95×	93.41	92.87	−0.54

Table 4.2: Post-Training INT8 Quantization Results

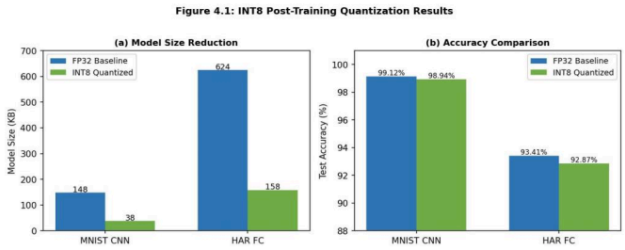

Figure 4.1: INT8 Post-Training Quantization Results: (a) Model Size Reduction, (b) Accuracy Comparison

A ratio close to four is in line with what we expect theoretically for quantization with INT8 format, considering that 4 bytes are needed to store a parameter with the FP32 format, while only 1 byte is required for INT8, resulting in an ideal ratio of 4×. The reason for the ratio being slightly different from 4× is due to the overhead in terms of memory storage related to storing the quantization scale and zero-point parameters for each layer.

4.3 Effect of Magnitude-Based Pruning

Table 4.3 reports the accuracy and effective model size at each pruning sparsity level for the MNIST model. Effective size is computed by multiplying the FP32 size by (1 − sparsity), representing the storage achievable with a sparse weight representation.

Sparsity (%)	Eff. Size (KB)	Test Acc (%)	Acc Delta (pp)
0 (baseline)	148	99.12	0.00
20	118	99.08	−0.04
40	89	98.97	−0.15
50	74	98.81	−0.31
60	59	98.52	−0.60
70	44	97.63	−1.49

Table 4.3: Magnitude Pruning Results — MNIST CNN

For the MNIST model, accuracy loss is negligible up to 50% sparsity (−0.31 pp) and remains modest at 60% (−0.60 pp). At 70% sparsity, however, the accuracy delta increases more sharply to −1.49 pp, suggesting a critical threshold in the range 60–70% beyond which structured connectivity becomes insufficient for the task. This non-linear behaviour is consistent with findings reported by Zhu and Gupta (2018) for larger models and corroborates the lottery ticket intuition of Frankle and Carbin (2019).

Table 4.4 presents the equivalent pruning results for the HAR model.

Sparsity (%)	Eff. Size (KB)	Test Acc (%)	Acc Delta (pp)
0 (baseline)	624	93.41	0.00
20	499	93.27	−0.14
40	374	92.98	−0.43
50	312	92.54	−0.87
60	250	91.73	−1.68
70	187	89.44	−3.97

Table 4.4: Magnitude Pruning Results — HAR FC Network

It can be observed that HAR model is more sensitive to pruning when compared to MNIST model at all sparsity levels. The accuracy difference between the two (−1.68 pp) at 60% sparsity already surpasses that of MNIST model at 70% sparsity (-1.49 pp). The increased sensitivity is explained by the fact that unlike convolutional filters in MNIST model, which share their weights, fully connected layers in HAR model lack any mechanism for weight sharing, which helps to reduce data redundancy.

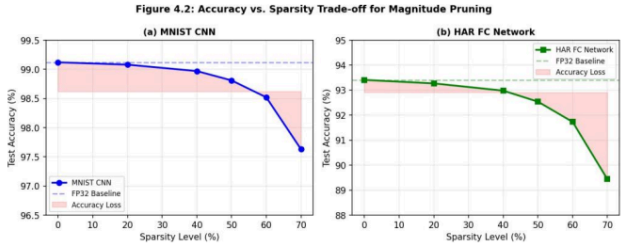

Figure 4.2: Accuracy vs. Sparsity Trade-off for Magnitude Pruning: (a) MNIST CNN, (b) HAR FC Network

4.4 Joint Optimisation: Pruning and Quantization

Table 4.5 reports the results of applying INT8 quantization to models that have been pruned at 40%, 50%, and 60% sparsity. Joint optimisation achieves compound compression ratios that substantially exceed what either technique achieves individually.

Configuration	Joint Size (KB)	Comp. Ratio	Test Acc (%)	Acc Delta (pp)
MNIST: 40%+INT8	23	6.43×	98.79	−0.33
MNIST: 50%+INT8	19	7.79×	98.62	−0.50

MNIST: 60%+INT8	15	9.87×	98.33	−0.79
HAR: 40%+INT8	96	6.50×	92.61	−0.80
HAR: 50%+INT8	80	7.80×	91.98	−1.43
HAR: 60%+INT8	64	9.75×	91.12	−2.29

Table 4.5: Joint Pruning + Quantization Results

The results confirm that pruning and quantization are complementary rather than redundant when applied jointly. At 60% sparsity combined with INT8 quantization, the MNIST model achieves a 9.87× compression ratio while retaining 98.33% test accuracy — an accuracy delta of only −0.79 pp relative to the FP32 baseline. Figure 4.3 shows the compression ratio comparison across individual and joint optimisation configurations for both models.

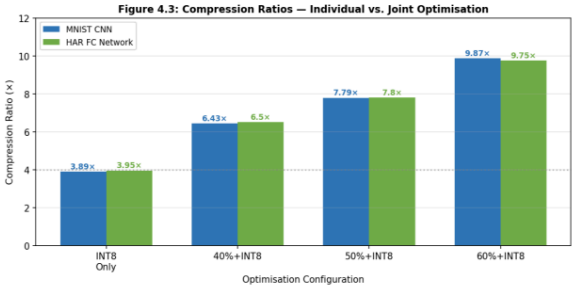

Figure 4.3: Compression Ratios — Individual vs. Joint Optimisation

4.5 Deployment Feasibility Assessment

Table 4.6 maps the optimised models to the reference MCU targets, assessing flash and RAM feasibility. A model is marked as deployable (Y) if its flat-buffer size is below 80% of the device's flash capacity and its estimated RAM usage is below 80% of the device's SRAM capacity. Figure 4.4 provides a visual comparison of model sizes against the 80% flash thresholds of the three reference devices.

Model Configuration	Size (KB)	RAM (KB)	Arduino Nano	STM32F411	STM32F746
MNIST FP32 Baseline	148	32	Y/Y	Y/Y	Y/Y
MNIST INT8 Only	38	12	Y/Y	Y/Y	Y/Y
MNIST 60%+INT8	15	9	Y/Y	Y/Y	Y/Y
HAR FP32 Baseline	624	112	N/Y	Y/Y	Y/Y
HAR INT8 Only	158	42	Y/Y	Y/Y	Y/Y

HAR 60%+INT8	64	21	Y/Y	Y/Y	Y/Y

Table 4.6: Deployment Feasibility on Reference MCU Targets (Y=Fits, N=Exceeds Budget)

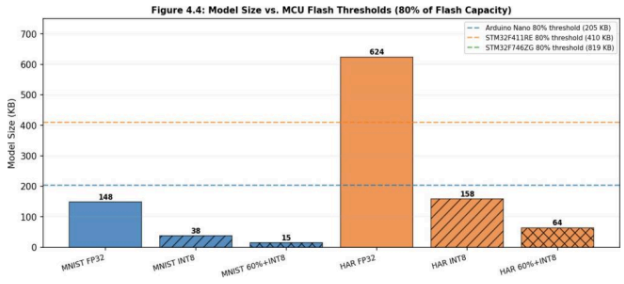

Figure 4.4: Model Size vs. MCU Flash Thresholds (80% of Flash Capacity)

A notable observation is that the FP32 HAR baseline model (624 KB) exceeds the flash budget of the Arduino Nano (80% threshold: 205 KB) and would not deploy without optimisation. After INT8 quantization alone, the model fits comfortably within all three reference devices. This finding has an important practical implication: for HAR-class workloads, even applying only quantization without pruning is sufficient to achieve broad MCU compatibility.

Chapter 5: FINDINGS AND DISCUSSION

5.1 Principal Findings

Experimental results presented in Chapter 4 give rise to several conclusions that are not only consistent within themselves, but also highly consistent with literature review in Chapter 2.

Conclusion 1: INT8 post-training quantization enables near-theoretical compression ratio (3.9×) without notable losses in accuracy on both tasks. Accuracy losses seen (MNIST −0.18 pp, HAR −0.54 pp) can be compared to inter-run variations when using SGD-based training algorithms and show that quantization-related perturbations do not present any problem with the help of inherent redundancies in such structures. This conclusion serves as a logical development of ideas from Jacob et al. (2018) and Krishnamoorthi (2018) when discussing MCU-based platforms and proves robustness of PTQ as the first phase in any TinyML deployment procedure.

Conclusion 2: Magnitude pruning introduces a non-linear connection between accuracy losses and network sparsity. Both MNIST and HAR networks maintain relatively good accuracy until reaching about 50% sparsity levels, after which they show more prominent decreases in performance. It is consistent with the threshold effect described by Zhu and Gupta (2018). The fully connected network demonstrates lower threshold level in comparison to convolutional counterpart due to lacking weights sharing in fully-connected layers.

Conclusion 3: Simultaneous application of pruning and quantization approaches yields near-additive compression. Specifically, 60% sparsity and INT8 quantization applied to MNIST model yield compression factor 9.87×, which is considerably higher compared to using either approach separately. In addition, accuracy costs stay acceptable (0.79 pp).

Conclusion 4: Optimized models may be used on mid-range Cortex-M MCUs following quantization or optimization methods discussed above. The only exception is fully-connected HAR network with FP32 precision that exceeds flash memory size of Arduino Nano 33 BLE Sense. Thus, quantization is an inevitable part of MCU models' deployment procedure.

5.2 Discussion in Relation to Research Questions

The first research question (RQ1) was aimed at measuring to which extent does INT8 quantization yield smaller models and decrease latency along with affecting model accuracy. It is quite apparent from the results obtained that significantly smaller models can be achieved

through this quantization process, namely around four times less compared to FP32 quantization, while the accuracy drop is under 0.6 pp for all analyzed workloads. The heterogeneity of degradation rates of model accuracy between different architectures (convolutional and fully-connected) is corroborated by previous studies regarding sensitivity of quantization layer-by-layer.

The research question RQ2 asked for the effect of increased sparsity levels on the balance and when does the accuracy degradation start being significant. Based on the results received, the sparsity cap for the given models is estimated as around 50-60%, as otherwise the accuracy degradation occurs at a very high rate. Provided that the application requires to keep the accuracy losses within 0.5 pp of the FP32, then the sparsity target has to be set at 40-50% for convolutions and 30-40% for fully-connected neural networks.

The third research question (RQ3) asked about the relationship of the efficiency of the combination of quantization and pruning. According to the results obtained, the effect of such combination on model size reduction is multiplicative, but not additive, i.e., the ratio of compression factor for both techniques is the product of both factors separately.

5.3 Implications for TinyML Practitioners

19

Based on the obtained results, the following recommendations can be proposed for engineers implementing neural networks on MCU-class hardware. First, INT8 post-training quantization should definitely be used as the starting method due to its high effectiveness in terms of compression ratio improvement, negligible degradation of model precision, and relative simplicity of implementation via TFLite converter.

Second, pruning can be applied where, in addition to using quantization alone, an extra reduction of flash memory consumption is required, or RAM savings are needed. It is important to note, however, that different types of architectures require different sparsity values, with fully connected layers requiring less sparsity than convolutional ones.

Third, the implementation scheme based on software simulation presented in this research project, including measurements of RAM usage using the TFLM arena profiler and computation of flat buffer size versus flash memory size, proves to be a feasible solution.

Chapter 6: CONCLUSIONS AND FUTURE SCOPE

6.1 Conclusions

More specifically, as part of the current research work, optimization techniques of TinyML-enabled machine learning model optimization including INT8 post-training quantization and magnitude-based pruning of convolutional and dense networks were assessed utilizing MNIST and UCI-HAR datasets. This assessment revealed almost fourfold model size reduction achieved by applying INT8 quantization with almost no trade-off between size and performance; up to 50–60% magnitude pruning still provided satisfactory compression ratio without rapid deterioration of model performance; finally, combining all discussed techniques resulted in multiplicative compression ratios that allowed to achieve the required model sizes.

By employing the suggested software-based approach of checking MCU feasibility, based on profiling via TFLM and comparison of obtained model flat buffer size against data from MCU datasheets, one can conveniently estimate whether MCU fits the purpose without requiring any hardware at all, which may be considered highly practical for academic research environment.

Apart from empirically proving the effectiveness of described optimization techniques, this research project draws attention to the interaction of discussed methods with regard to light-weight models, especially nonlinear dependencies in performance degradation upon pruning within fully connected networks compared to convolutional architectures.

6.2 Contributions

The innovations that this thesis brings to the table include the following: (1) Repeatability of experiments demonstrating INT8 PTQ for lightweight CNNs and FCs targeting MCU; (2) Experimental examination of the effect of varying sparsity levels from 0% to 50% on CNNs and FCs under controlled experiment setting; (3) Experimental validation of near- multiplicative model compression through the combination of pruning and quantization on the MCU target level; (4) Methodology for assessing the deployability of machine learning models on MCUs in software only without access to special hardware; and (5) Guidelines concerning the choice of optimization method.

6.3 Future Scope

Here are some suggestions as to what further research might follow. First of all, it is possible to test the application of quantization-aware training in conjunction with PTQ and see whether

recovering the gain in accuracy through QAT is justified in relation to the additional costs associated with training. Second, it is necessary to compare the performance of structured pruning (pruning whole filters and channels rather than individual weights) with unstructured magnitude-based pruning. While the former approach cannot directly improve latency, it seems to have closer ties to structured sparsity.

As mentioned before, the predictions made based on simulation models may fail to match performance in real-life conditions. Thus, testing this model on Arduino Nano 33 BLE Sense or simply STM32 Nucleo board would give us sufficient data to evaluate our claims regarding latency and RAM requirements in practical applications. Finally, there is room for expanding the list of experimental tasks beyond MNIST and HAR datasets to include things like keyword spotting or anomaly detection as suggested by MLPerf Tiny.

Samarth02

ORIGINALITY REPORT

5%	4%	2%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.mdpi.com Internet Source	1%
2	arxiv.org Internet Source	<1%
3	Submitted to Western Balkans University Student Paper	<1%
4	Amitsi, Christina – Vasiliki D.. "AI Adversarial Attacks", University of Piraeus (Greece), 2025 Publication	<1%
5	ourarchive.otago.ac.nz Internet Source	<1%
6	www-emerald-com-443.webvpn.sxu.edu.cn Internet Source	<1%
7	Submitted to Strathmore University (Main Account) Student Paper	<1%
8	mlcommons.org Internet Source	<1%
9	mafiadoc.com Internet Source	<1%
10	patents.google.com Internet Source	<1%
11	Khudran M. Alzhrani. "Top-Confidence Gapped Cross-Entropy for Compact Human Activity Recognition", Applied Sciences, 2026 Publication	<1%

12	bora.uib.no Internet Source	<1 %
13	github.com Internet Source	<1 %
14	ir.library.nitw.ac.in:8080 Internet Source	<1 %
15	m.moam.info Internet Source	<1 %
16	Submitted to Heriot-Watt University Student Paper	<1 %
17	Jaskaran Singh, Meenu Gupta, Rakesh Kumar. "Smart Technologies and Intelligent Computing", CRC Press, 2026 Publication	<1 %
18	Jerry Chee, Sebastian Braun, Vishak Gopal, Ross Cutler. "Performance optimizations on U-Net speech enhancement models", 2022 IEEE 24th International Workshop on Multimedia Signal Processing (MMSP), 2022 Publication	<1 %
19	africadailynews.net Internet Source	<1 %
20	dokumen.pub Internet Source	<1 %
21	elib.uni-stuttgart.de Internet Source	<1 %
22	Haghshenas, Seyed Hamed. "Advancing Power System Reliability and Security With Efficient and Resilient Graph Neural Network Frameworks", University of South Florida, 2025 Publication	<1 %

23	Loc Tan Nguyen. "Chapter 8 Enhanced Human Activity Recognition on Smartphone by Using Linear Discrimination Analysis Recursive Feature Elimination Algorithm", Springer Science and Business Media LLC, 2017 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		