Accelerating CNN Inference on MCUs with Quantized Early-Exit Networks

Gunju Park , Seungtae Hong, Jeong-Si Kim
Electronics and Telecommunications Research Institute, Daejeon, Republic of Korea
Emails : parkgj@etri.re.kr , sthong@etri.re.kr , sikim00@etri.re.kr

***Abstract*—On-device AI on microcontrollers (MCUs) is severely hampered by their limited memory and computational resources. To address this, we propose a complete pipeline for creating and deploying efficient, dynamic neural networks. Our methodology features a mixed training strategy to stably attach early-exit branches to a pretrained CNN backbone, followed by INT8 post-training quantization (PTQ). We validated our approach with TensorFlow-Lite on a host CPU using a Modified MobileNetV2 architecture. The results demonstrate a clear and controllable trade-off between latency and accuracy; we achieve up to a 1.54x inference speedup compared to a standard quantized model, with a peak SRAM usage of only 67KB. This work confirms that quantized early-exit networks are a practical and effective solution for accelerating inference on resource-constrained devices.**

***Index Terms*—Quantization, Lightweight CNN, Early Exit Network**

I. Introduction

The proliferation of low-power microcontrollers (MCUs) has ignited the field of Tiny Machine Learning (TinyML), aiming to embed intelligence directly onto edge devices [1]. This on-device approach offers significant advantages, including low latency, enhanced privacy, and connectivity independence. However, a fundamental conflict exists between the ever-growing size of deep neural networks and the severe resource constraints of MCUs, which are limited to kilobytes of RAM and milliwatts of power [1]. This ”model-hardware gap” presents the primary obstacle to realizing the full potential of ubiquitous AI.

To bridge this gap, static optimization techniques such as architecting efficient networks like MobileNetV2 [2] and applying model compression through INT8 quantization [3] have become standard practice. While effective, these methods adhere to a static inference paradigm, where the entire computational graph is executed for every input, regardless of its complexity. This often leads to redundant computation for simpler samples.

To address this inefficiency, we leverage dynamic inference through an Early-Exit mechanism [4], [5]. This strategy augments a backbone network with intermediate classifiers, allowing inference to terminate prematurely for inputs that can be classified with high confidence at an early stage. Consequently, the average computational cost and latency are significantly reduced. Despite the promise of dynamic networks, a streamlined methodology to effectively train, quantize, and evaluate them for resource-constrained targets is currently lacking. This paper introduces a practical pipeline to fill this gap. Our main contributions are threefold:

1) A **mixed training strategy** (disjoint-then-joint) to effectively attach and fine-tune early-exit branches on a pretrained backbone model.
2) A complete pipeline for applying **post-training quantization (PTQ)** to the entire multi-exit dynamic network to make it MCU-friendly.
3) A comprehensive **performance analysis** using TensorFlow Lite, quantifying the trade-offs between accuracy and inference speedup.

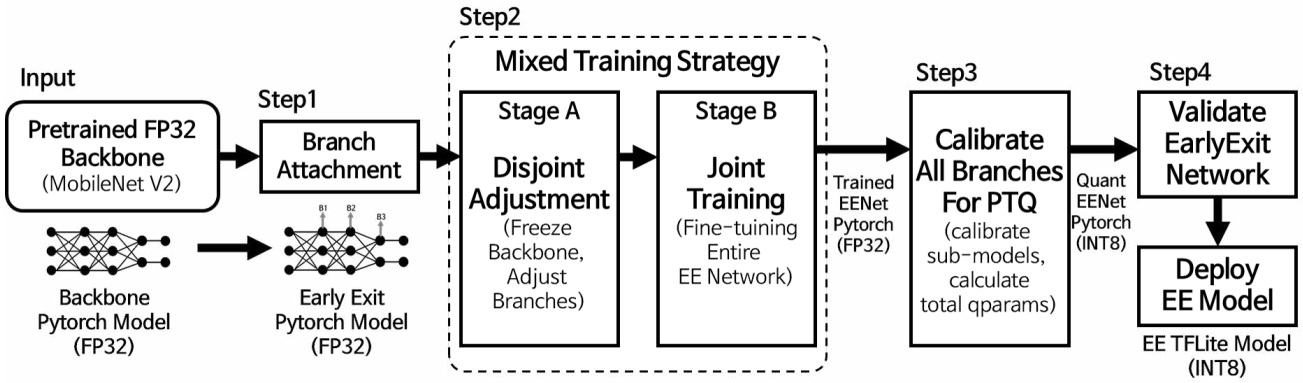

Fig. 1: Overview of the Proposed Pipeline

II. Proposed Methods

Our objective is to transform a standard, pretrained FP32 CNN into a quantized, inference-efficient dynamic network suitable for resource-constrained environments. The proposed pipeline, illustrated in Fig. 1, consists of a mixed training strategy followed by a post-training quantization stage.

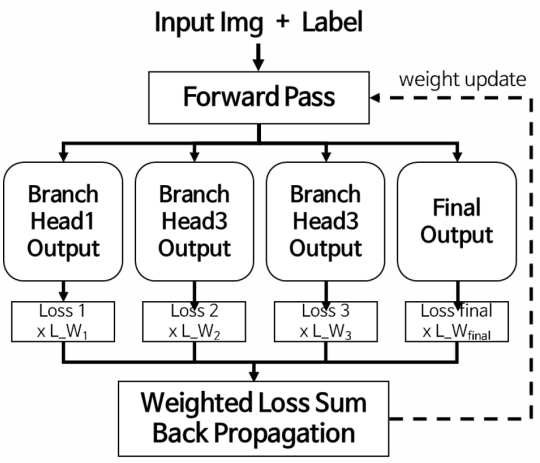

Fig. 2: The joint training process where the total loss is a weighted sum of losses from all intermediate branches and the final output.

A. Mixed Training for Early-Exit Branches

Attaching new classification branches to a pretrained backbone, a process explored in post-trained early-exit networks [6], can disrupt its well-learned features if done naively. To ensure a stable and effective learning process, we adopt a two-stage mixed training strategy that combines disjoint and joint training regimes, a method proven effective for early-exit models [7].

Stage A: Disjoint Training Initially, we freeze the weights of the pretrained backbone and train only the newly attached early-exit branches. This stage allows the branches to learn a reasonable initial mapping from the backbone's feature space without causing catastrophic forgetting or instability in the main network.

Stage B: Joint Training Once the branches are stabilized, we unfreeze the entire network and fine-tune all weights—both the backbone and the branches—end-to-end with a lower learning rate. This joint training phase allows the backbone and the exit branches to co-adapt, leading to a more holistic optimization and maximizing the accuracy of each potential exit point [7]. During this stage, the total loss for backpropagation is calculated as a weighted sum of the individual losses from each exit branch and the final output.

$$L_{\text{total}} = W_{\text{final}} \cdot L_{\text{final}} + \sum_{n=1}^{N} W_n \cdot L_n \tag{1}$$

Here, L_n is the loss from the n-th early-exit branch, L_final is the loss from the original final output, and W_n and W_final are the corresponding scalar weights that control the influence of each classifier. As illustrated in Fig. 2, this joint loss function ensures that the gradients from all classifiers contribute to the update of the shared backbone parameters, encouraging the network to learn feature representations that are beneficial for all exit points.

B. Post-Training Quantization for Early Exits

To meet the strict memory and computational constraints of MCUs, we apply INT8 post-training quantization to the fully trained FP32 model. This process converts all floating-point weights and activations to 8-bit integer representations, which is critical for enabling fast, energy-efficient integer-only arithmetic on target hardware [8]. During this stage, a small, representative calibration dataset is used to determine the optimal quantization parameters (scale and zero-point) for every tensor in the network, including those in the backbone and each exit branch, thereby minimizing the accuracy degradation that can result from the reduction in precision.

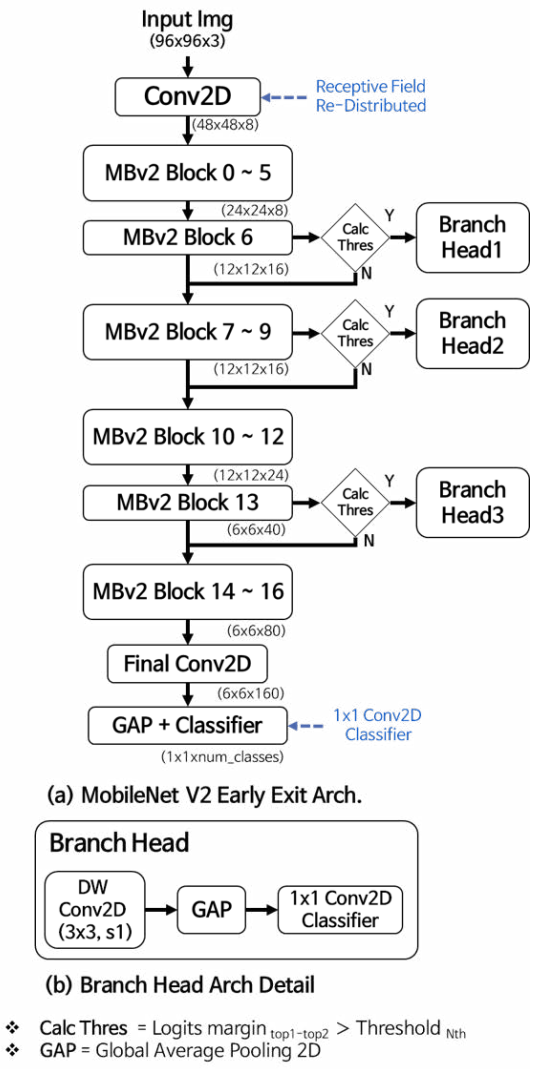

Fig. 3: MobileNetV2 Arch with Early-Exit Branches

III. Experiments and Results

In this section, we present a comprehensive evaluation of our proposed methodology. We aim to quantify the performance gains of the quantized early-exit network against a standard baseline and analyze the inherent trade-off between inference speed and accuracy.

A. Experimental Setup

Model and Dataset: We use a MobileNetV2 [2] architecture augmented with three early-exit branches, as depicted in Fig. 3. The model is trained and evaluated on the Visual Wake Words (VWW) dataset [9], a standard benchmark for TinyML vision tasks.

Training and Evaluation: The model was trained in PyTorch, with the final FP32 network achieving validation accuracies of 79.27% (B1), 85.41% (B2), 87.32% (B3), and 87.65% (Final Exit). For performance analysis, quantized models were evaluated using the TensorFlow Lite (TFLite) interpreter on a host CPU to measure relative inference latency and estimate memory footprint.

Baseline Model: The baseline is a standard INT8 quantized MobileNetV2 without early exits, which achieves 81.09% Top-1 accuracy on the VWW dataset.

Early-Exit Threshold: The inference policy is controlled by a confidence percentile threshold, p, derived from a calibration dataset. For example, p75 sets the threshold at the 75th percentile of confidence scores, meaning inputs exit early if their confidence is in the top 25% of calibration scores.

B. Performance Evaluation

The core results of our evaluation are summarized in Table I. The data clearly illustrates the trade-off between accuracy and inference acceleration. By adjusting the exit threshold p, we can navigate this trade-off to suit different application requirements. At p50, the policy is aggressive, allowing 73.05% of samples to exit early, which results in a maximum speedup of 1.54x over the baseline at the cost of a slight drop in accuracy. The p75 threshold offers a balanced compromise, achieving a competitive accuracy of 80.51% while still providing a significant 1.15x speedup. A conservative p95 threshold forces most samples through the entire network, recovering accuracy to 80.66% but offering negligible acceleration.

TABLE I: Accuracy and Performance Trade-off

Model Conf.	Val Acc. (%)	EE Rate (%)	Speedup
FP32 (PyTorch)	87.65	-	-
INT8 (Baseline)	81.09	-	1.00x
Our EE (p50)	**79.45**	73.05	**1.54x**
Our EE (p75)	**80.51**	29.56	1.15x
Our EE (p95)	**80.66**	1.33	1.01x

Furthermore, the estimated memory requirements confirm the model's suitability for deployment on typical MCUs. Our model requires a Flash Memory footprint of 260 KB, which is comparable to the baseline, and a Peak SRAM Usage of only 67 KB, keeping it well within the constraints of low-cost microcontrollers.

IV. Conclusion

In this paper, we presented an end-to-end pipeline for creating and evaluating efficient, quantized early-exit networks specifically targeted for microcontroller-based applications. We introduced a mixed training strategy to ensure stable and effective learning of intermediate branches on a pretrained backbone and demonstrated the application of post-training quantization to the entire dynamic network.

Our TFLite-based evaluation confirmed the efficacy of this approach, revealing a clear and controllable trade-off between inference latency and model accuracy. The results show that our method can achieve up to a 1.54x theoretical speedup over a conventional quantized baseline while maintaining a low memory footprint suitable for typical MCUs. This work validates that dynamic inference is not just a theoretical concept but a practical and potent strategy for optimizing neural network performance in the highly constrained world of TinyML. Future work could explore Quantization-Aware Training (QAT) to further improve accuracy or automate the optimal placement of exit branches within a given architecture.

Acknowledgment

This research was partly supported by the Challengeable Future Defense Technology Research and Development Program through the Agency For Defense Development(ADD) funded by the Defense Acquisition Program Administration(DAPA) in 2022(No.915062201,60%) and Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT)(No.RS-2024-00339187,Core Technology Development of On-device Robot Intelligence SW Platform,40%)

References

[1] P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2019.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[3] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[4] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: A very deep network with branching built in," in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 2244–2248.

[5] Y. Kaya, S. Hong, and T. Dumitras, "Shallow-deep networks: Understanding and mitigating network overthinking," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019, pp. 3301–3310.

[6] A. Lahiany and Y. Aperstein, "Pteenet: Post-trained early-exit neural networks augmentation for inference cost optimization," *IEEE Access*, vol. 10, pp. 69 680–69 687, 2022.

[7] B. Krzepkowski, M. Michaluk, F. Szarwacki, P. Kubaty, J. Pomponi, B. WÄłjcik, K. Adamczewski *et al.*, "Joint or disjoint: Mixing training regimes for early-exit models," *arXiv preprint arXiv:2407.14320*, 2024.

[8] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.

[9] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," *arXiv preprint arXiv:1906.05721*, 2019.