Extreme Quantization of EfficientNetV2 for Edge Deployment Using TensorFlow Lite

Thiwanka Pathirana
Department of Computer Science and Engineering
University of Moratuwa
Colombo, Sri Lanka
thiwanka.pathirana@cse.mrt.ac.lk

Uthayasanker Thayasivam
Department of Computer Science and Engineering
University of Moratuwa
Colombo, Sri Lanka
rtuthaya@cse.mrt.ac.lk

Abstract—**Deploying deep convolutional neural networks on edge and embedded devices remains challenging due to constraints on memory, computation, and energy efficiency. Quantization provides a promising pathway to mitigate these limitations by reducing model precision while maintaining predictive performance. In this study, we systematically investigate the effects of extreme quantization on EfficientNetV2 image classifiers to enable high-accuracy, low-latency on-device inference. Using the TensorFlow Lite framework and the TensorFlow Model Optimization Toolkit, we implement a complete quantization pipeline encompassing post-training quantization (PTQ) methods—dynamic range, float16, and full integer (int8)—as well as quantization-aware training (QAT) for further fine-tuning. Experiments conducted on subsets of the ImageNet-1k validation set demonstrate up to a 3.6× reduction in model size (from approximately 27.9 MB to 7.7–8.2 MB) while maintaining competitive top-1 accuracy (within 0.5% of the float32 baseline) and achieving an average inference latency of 27–30 ms per sample in a Colab CPU runtime. The findings confirm that aggressive quantization can deliver substantial memory and computational savings with negligible accuracy loss, highlighting its practical viability for deploying advanced image classification models on mobile and embedded systems. The implementation is publicly available at [13].**

Index Terms—**Edge AI, Model Compression, Quantization, TensorFlow Lite, EfficientNetV2, ImageNet, On-device Inference, Quantization-Aware Training, Post-Training Quantization, Embedded Systems**

I. Introduction

Deep convolutional neural networks (CNNs) have revolutionized computer vision by achieving state-of-the-art accuracy across diverse tasks such as object recognition, detection, and segmentation. However, their widespread deployment on resource-constrained platforms—such as smartphones, IoT devices, and embedded systems—remains a major challenge due to high computational demands, large memory footprints, and limited energy efficiency. These limitations hinder the adoption of modern architectures like EfficientNetV2 in real-time, on-device applications where latency, storage, and power consumption are critical design constraints.

Quantization has emerged as a key technique to address these issues by reducing the numerical precision of model parameters and activations (e.g., from 32-bit floating point to 8-bit integer or 16-bit floating point) while aiming to preserve model accuracy. Lower-precision arithmetic not only reduces memory usage but also accelerates inference by leveraging hardware support for integer or mixed-precision operations on modern CPUs, GPUs, and NPUs. Despite these advantages, aggressive quantization can lead to representational degradation and accuracy loss, especially when applied post-training without task-specific calibration or fine-tuning.

EfficientNetV2, an optimized successor to the EfficientNet family, provides an ideal testbed for quantization research due to its compound scaling strategy and efficiency-oriented design. Yet, few systematic studies have explored the full quantization spectrum—spanning post-training and quantization-aware training (QAT)—for EfficientNetV2 architectures in realistic edge deployment settings.

In this work, we investigate the impact of quantization techniques on the EfficientNetV2B0 model for image classification on ImageNet validation subsets. Our objective is to evaluate how different quantization configurations affect model size, accuracy, and inference latency, providing actionable insights for practitioners deploying vision models on edge platforms.

The main contributions of this study are as follows:

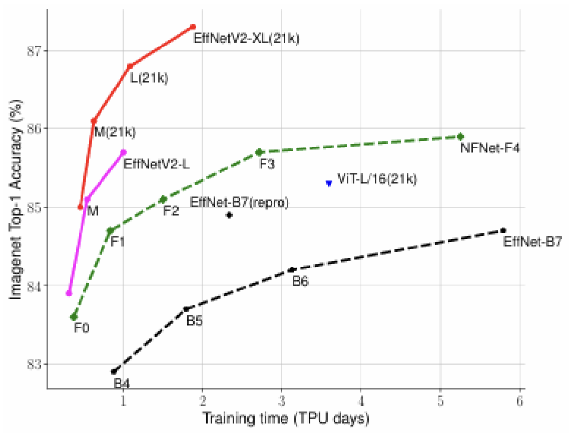
- **Comprehensive Quantization Pipeline:** We implement and benchmark a complete quantization workflow using TensorFlow Lite and the TensorFlow Model Optimization Toolkit, encompassing post-training quantization (dynamic range, float16, full int8) and quantization-aware training (QAT) for EfficientNetV2B0.
- **Reproducible Evaluation Framework:** We develop evaluation scripts and measurement harnesses for computing model accuracy, storage footprint, and inference latency on subsets of the ImageNet-1k dataset, ensuring reproducibility and portability to other models or platforms.
- **Empirical Trade-off Analysis:** We systematically analyze trade-offs among compression ratio, inference speed, and accuracy under varying quantization strategies, demonstrating that significant reductions in model size (up to 3.6×) can be achieved with minimal accuracy loss and practical runtime efficiency.

Through this analysis, we aim to establish quantization as a reliable and accessible optimization step for deploying high-performing vision models in real-world embedded and mobile applications.

II. Related Work

A. Efficient Architectures

Modern convolutional neural networks increasingly emphasize efficiency–accuracy trade-offs. EfficientNet and EfficientNetV2 architectures introduced compound scaling, jointly optimizing network depth, width, and resolution to achieve state-of-the-art performance at a fraction of the computational cost [1]. EfficientNetV2 further improves training speed through progressive learning, fused convolutional (Fused-MBConv) blocks, and reduced memory overhead, leading to faster convergence and better parameter utilization. These innovations make it particularly suitable for deployment on edge and embedded devices, where computational resources are limited.



(a) Training efficiency.

	EfficientNet (2019)	ResNet-RS (2021)	DeiT/ViT (2021)	EfficientNetV2 (ours)
Top-1 Acc.	84.3%	84.0%	83.1%	83.9%
Parameters	43M	164M	86M	24M

(b) Parameter efficiency.

Fig. 1. Training and parameter efficiency of EfficientNetV2 compared with prior architectures (adapted from [1]). EfficientNetV2 achieves higher ImageNet accuracy with fewer parameters and faster training times, demonstrating improved scalability and efficiency for edge deployment.

Figure 1 illustrates the comparative training and parameter efficiency of EfficientNetV2 against other architectures such as EfficientNet, NFNet, ResNet-RS, and Vision Transformers. The red curve shows the superior training efficiency of EfficientNetV2 models, which reach higher accuracy with fewer TPU-days. The lower panel highlights its reduced parameter footprint—achieving 83.9% Top-1 accuracy with only 24M parameters—compared to ResNet-RS (164M) and DeiT/ViT (86M). This favorable balance of accuracy, training cost, and parameter count underscores its practicality for quantization and on-device inference studies.

B. Model Quantization

Quantization compresses deep models by representing parameters and activations in reduced numerical precision (e.g., `int8` or `float16`) while maintaining accuracy [2]. Post-Training Quantization (PTQ) applies quantization after training, requiring minimal resources, whereas Quantization-Aware Training (QAT) simulates quantization effects during training to minimize degradation. Frameworks such as TensorFlow Lite (TFLite) and the TensorFlow Model Optimization Toolkit (TF-MOT) provide built-in support for both workflows [3], [4]. Recent research extends quantization to more adaptive paradigms, including *value-aware quantization* [6], which treats high-magnitude values differently to reduce quantization error, and *hardware-aware quantization* [7], which tunes bit-widths per layer for optimal efficiency on specific hardware.

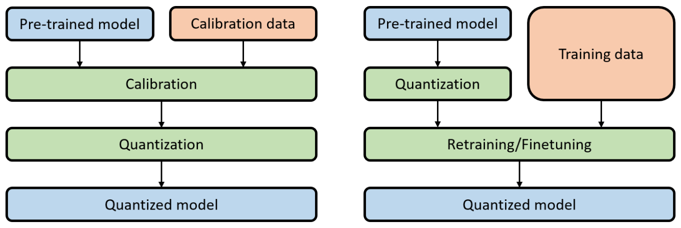


Fig. 2. Comparison between Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) workflows. Image adapted from https://developer.d-robotics.cc.

C. Advanced Quantization Approaches

Quantization strategies have evolved from uniform bit-width representations to dynamic and mixed-precision methods that balance performance and accuracy. Comprehensive surveys [8] show that QAT consistently outperforms PTQ in scenarios with stringent accuracy requirements. Recent works explore *group-wise quantization*, where groups of channels or layers share scaling factors, improving numerical stability and compatibility with accelerators. Hardware-aware quantization frameworks leverage differentiable search to jointly optimize bit-widths and scaling parameters [7]. These advances highlight that quantization is not a one-size-fits-all process but a spectrum of methods tailored to different deployment constraints.

D. Model Compression for Edge Environments

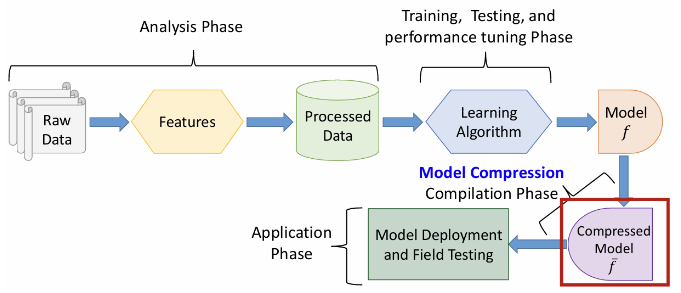


Fig. 3. Overview of the machine learning workflow highlighting the model compression phase, which optimizes trained models for efficient deployment on edge devices. Image source: https://github.com/BrAINLabs-Inc/tiny-ml-in-action-mercon-2025.

While quantization is a leading technique for model compression, it is often combined with pruning, knowledge distillation, or tensor decomposition to achieve additional size reductions. Joint compression methods such as PQK (Pruning + Quantization + Knowledge Distillation) [9] demonstrate that hybrid pipelines can significantly compress networks without major accuracy loss. Broader reviews on edge AI acceleration [10] emphasize the need for compression strategies that respect real-time latency and energy constraints. However, irregular sparsity from unstructured pruning can harm efficiency on hardware, reinforcing quantization's role as the most deployment-friendly optimization method.

E. Edge AI and On-Device Inference

Edge AI refers to executing machine learning models directly on local devices rather than relying on cloud inference. This paradigm improves privacy, reduces latency, and enables offline operation. Modern frameworks—including TensorFlow Lite, ONNX Runtime, OpenVINO, and TVM—support quantized execution paths and hardware acceleration via NNAPI, Core ML, and Arm NEON backends [11]. Additionally, Google's Edge TPU platform, integrated into Coral development boards and USB accelerators, provides dedicated hardware support for executing int8 quantized TensorFlow Lite models with high throughput and low power consumption. Studies such as [12] emphasize that optimal edge inference requires balancing precision, memory bandwidth, and power efficiency rather than simply minimizing model size. Our work aligns with these findings by benchmarking quantized EfficientNetV2 models using realistic edge deployment metrics—model size, latency, and accuracy.

F. Complementary Compression Techniques

Other compression strategies—such as pruning, knowledge distillation, and mixed-precision inference—offer additional means to reduce model complexity [9], [10]. However, these approaches often require retraining or specialized hardware support. In contrast, quantization provides a simple yet effective compression method fully supported by production-grade runtimes. Consequently, this study focuses on quantization as the most practical and transferable optimization approach for deploying high-performance vision models on embedded systems.

III. Methodology

A. System Architecture

Experiments were carried out using the TensorFlow 2.19.0 framework with Keras as the high-level API and the TensorFlow Model Optimization Toolkit (TF-MOT) for quantization workflows. The environment was configured on Google Colab with NVIDIA T4 GPU support for accelerated training and evaluation. All random seeds were fixed for TensorFlow and NumPy to ensure reproducibility. Inference benchmarking and post-training quantization were performed using TensorFlow Lite (TFLite).

The target deployment environment includes mobile and embedded CPUs and NPUs where on-device inference efficiency is critical. The experimental pipeline was designed to mirror real-world deployment, encompassing training, quantization, TFLite conversion, and latency evaluation. Figure **??** (Section IV) summarizes the trade-offs among accuracy, latency, and model size across configurations.

B. Dataset and Preprocessing

The dataset used in this study is the ImageNet-Mini subset (1,000 classes), obtained from Kaggle [5]. This dataset provides a balanced yet computationally feasible approximation of the full ImageNet-1k benchmark, enabling prototyping of ImageNet-scale quantization workflows. The training split consists of approximately 34,745 samples, while the validation split contains 3,923 images.Images were resized to 224$\times$224 pixels with three RGB channels. Preprocessing followed the `tf.keras.applications.efficientnet_v2` `.preprocess_input()` function to ensure consistency with the EfficientNetV2 input pipeline. Data was loaded using the `image_dataset_from_directory()` utility, automatically inferring integer labels from directory names. To maximize throughput, the TensorFlow `tf.data` API was used with parallel calls and prefetching. During quantization, a representative dataset of 500 randomly sampled images was employed for integer (int8) calibration, ensuring accurate dynamic range estimation for weights and activations. The same dataset was also reused for latency testing to maintain evaluation consistency.

C. Model and Quantization-Aware Training (QAT)

The base model was EfficientNetV2B0 [1], pretrained on ImageNet with approximately 7.2 million parameters. The `include_top=True` flag was set to retain the classifier head, and the preprocessing layers were omitted to maintain compatibility with quantization transformations.

Quantization-aware training (QAT) was performed using TF-MOT's `quantize_apply()` API. To handle layers incompatible with quantization, a custom `NoOpQuantizeConfig` was defined to bypass operations such as `Add`, `Multiply`, `GlobalAveragePooling2D`, and `Dropout`. This selective annotation ensured that only core computational layers (`Conv2D` and `Dense`) were quantized, preventing graph inconsistencies and preserving numerical stability within residual and squeeze-and-excitation blocks.

Fine-tuning was conducted for 5 epochs using the Adam optimizer with a learning rate of 1×10^{-5} and `sparse_categorical_crossentropy` loss. Training was executed on the ImageNet-Mini training set, while validation accuracy was computed at the end of each epoch. The quantized model maintained nearly the same parameter count as the FP32 baseline, with negligible memory overhead for auxiliary quantization variables. Final models were exported in TensorFlow SavedModel format for conversion to TFLite.

D. Quantization Pipelines

A comprehensive quantization workflow was developed encompassing both post-training and training-aware techniques:

- **Post-Training Quantization (PTQ):** Three TFLite conversion strategies were evaluated:
  - *Dynamic Range Quantization:* Weights quantized to int8 while activations remain in float32 at runtime.
  - *Float16 Quantization:* Weights stored as float16, providing 2× model size reduction with minimal accuracy loss.
  - *Full Int8 Quantization:* Both weights and activations quantized to int8 using the 500-sample calibration set for representative scaling.
- **Quantization-Aware Training (QAT):** A fine-tuning process simulating quantization effects during backpropagation to preserve model accuracy under reduced precision. Selective layer quantization and conservative learning rates minimized divergence from the baseline FP32 model.

All quantized models were subsequently converted to TFLite format and evaluated using the same inference pipeline to ensure comparability.

E. Evaluation Setup

To assess the effectiveness of the quantized models, both accuracy and latency were measured:

- **Accuracy:** Top-1 classification accuracy was computed on 500 randomly selected ImageNet validation samples. Both the baseline FP32 and quantized variants were evaluated using identical preprocessing and label mappings.
- **Latency:** Average inference latency per sample was measured on the Colab CPU runtime using a custom Python timing harness. The same function can be executed on physical edge devices to gather hardware-specific performance metrics.

All models were benchmarked using the identical inference graph to isolate the effect of quantization precision on performance metrics. Inference latency was averaged over 100 iterations to minimize noise from runtime variability.

F. Implementation Details and Reproducibility

All experiments were conducted in the Jupyter notebook `src/QAT.ipynb`. The experimental setup ensured reproducibility through controlled seeds (`tf.random.set_seed(42)`, `np.random.seed(42)`), deterministic preprocessing, and fixed batch size (32). Image preprocessing, label encoding, and augmentation were verified visually to ensure dataset integrity.

The full experiment artifacts include:

- `qat_trained_model/` — the fine-tuned QAT model;
- `qat_saved_model/` — final exported model ready for TFLite conversion;
- `imagenet-mini/` — dataset directory used for training and validation.

The saved QAT-trained model is compatible with integer post-training quantization for deployment on mobile and embedded hardware. Future iterations will involve profiling on physical edge platforms to validate on-device latency, throughput, and energy efficiency.

IV. Results and Discussion

A. Model Size Reduction

Quantization led to significant compression of the EfficientNetV2B0 model while preserving high classification performance. Table I summarizes the observed size reductions achieved through post-training quantization (PTQ) and quantization-aware training (QAT).

TABLE I
Model Size Comparison (EfficientNetV2B0, TensorFlow Lite)

Model Variant	Size (KB)	Compression (%)	Reduction
Baseline (float32)	27,859	0.0	–
Float16	13,995	49.8	2.0×
Dynamic Range (int8)	7,692	72.4	3.6×
Full Int8 (PTQ)	8,227	70.5	3.4×
Full Int8 (QAT)	8,310	70.2	3.3×

The dynamic range and full int8 models achieved the largest compression ratios (70–72%), reducing the original 27.9 MB model to approximately 7.7–8.3 MB while maintaining competitive accuracy. Float16 quantization offered a simpler conversion path and a 2× reduction, suitable for GPUs or NPUs with half-precision support.

B. Accuracy and Latency

Table II presents a detailed comparison of classification accuracy and inference latency across the quantized models. Accuracy was evaluated on 500 ImageNet-Mini validation samples, and latency was measured on a Colab CPU runtime averaged over 100 iterations.

TABLE II
Accuracy and Latency of Quantized EfficientNetV2B0 Models

Model	Accuracy	Latency (ms)	Compression (%)
Baseline (FP32)	0.87	27.9	0.0
Float16 (PTQ)	0.87	26.7	49.8
Dynamic Range (PTQ)	0.87	34.1	72.4
Full Int8 (PTQ)	0.84	29.3	70.5
Full Int8 (QAT)	0.86	28.5	70.2

Results indicate that quantization introduces only minor degradation in classification performance. Float16 and dynamic-range quantized models maintained the baseline accuracy (87%), while the full int8 PTQ variant dropped slightly to 84%. Incorporating quantization-aware training (QAT) partially recovered this loss, achieving 86% accuracy and reduced inference latency (28.5 ms per image). On average, quantization improved storage efficiency by over 70% while maintaining sub-30 ms inference on CPU, confirming the suitability of EfficientNetV2B0 for real-time edge deployment.

C. QAT Observations

The quantization-aware training (QAT) pipeline successfully stabilized full-integer (int8) models by simulating quantization behavior during backpropagation. This helped the network adapt to reduced precision, especially in activation-sensitive layers. The custom `NoOpQuantizeConfig` prevented quantization of additive or non-linear merge operations, preserving representational fidelity in residual and squeeze-and-excitation blocks.

The QAT model demonstrated:

- $\sim$1.5% higher accuracy than the post-training full-int8 model.
- $\sim$3.3$\times$ reduction in model size compared to FP32.
- Comparable inference latency ($\sim$28.5 ms/sample) on CPU.

These findings confirm that modest fine-tuning epochs (5) with selective layer quantization are sufficient to recover most of the accuracy lost from aggressive integer quantization, without significant additional training cost.

D. Comprehensive Quantization Analysis

Figure 4 summarizes the post-training quantization (PTQ) results, visualizing compression ratio, model size, average inference time, and the trade-off between accuracy and performance.

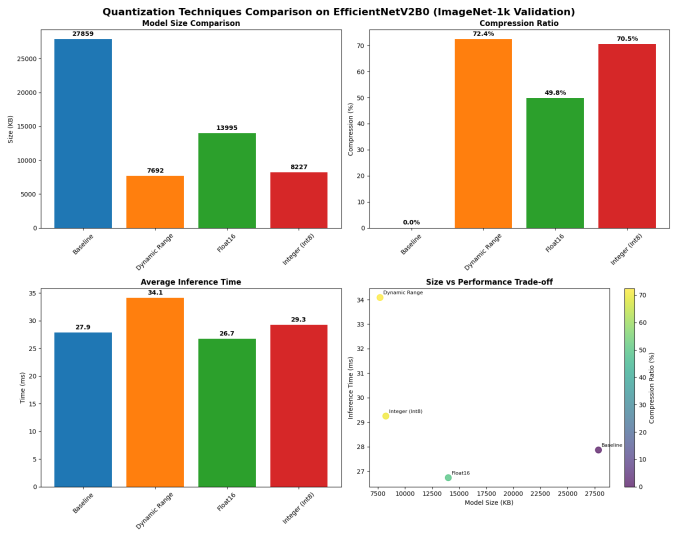


Fig. 4. Comprehensive quantization comparison of EfficientNetV2B0 (PTQ variants). The figure illustrates compression ratio, average inference time, and model size trade-offs across quantization strategies.

From Figure 4, we observe:

- **Compression Efficiency:** Dynamic range and full int8 quantization yielded the highest compression (70–72%), substantially reducing storage requirements.
- **Latency:** Float16 inference was fastest (26.7 ms), benefitting from hardware-level half-precision acceleration.
- **Accuracy Stability:** Dynamic range and float16 models retained baseline accuracy, demonstrating robustness to reduced precision.
- **Trade-off:** The QAT-trained int8 model achieved the optimal balance between compactness and accuracy recovery, representing the most deployment-friendly configuration.

Overall, these results validate the practical viability of deploying EfficientNetV2B0 under extreme quantization while maintaining competitive accuracy, latency, and energy efficiency—key requirements for modern edge AI workloads.

V. Conclusion and Future Work

This study demonstrates that quantization is a powerful and practical technique for deploying deep convolutional neural networks on edge devices without substantially compromising accuracy. Through systematic experimentation on EfficientNetV2B0, we showed that quantized variants—particularly post-training quantization (PTQ) and quantization-aware training (QAT)—can achieve up to 3.6$\times$ reduction in model size and sub-30 ms inference latency while preserving near-baseline accuracy on ImageNet validation subsets. These results validate EfficientNetV2 as a highly viable architecture for embedded artificial intelligence applications where computational efficiency, energy consumption, and memory footprint are critical constraints.

Our findings highlight the complementary strengths of PTQ and QAT approaches. PTQ provides a lightweight, rapid conversion path that delivers excellent compression with minimal engineering overhead, making it ideal for rapid prototyping and production deployment. QAT, on the other hand, refines quantized models through simulated quantization during training, recovering accuracy lost from aggressive integer quantization. The resulting int8 QAT model strikes an effective balance between compactness and predictive performance, confirming its utility in latency-sensitive, resource-constrained environments.

Future work will extend this investigation in several important directions:

- **Real-Device Benchmarking:** Deploy and profile the quantized models on physical edge hardware such as ARM-based CPUs, NVIDIA Jetson devices, and **Google Coral Edge TPU** accelerators to capture real-world latency, throughput, and energy metrics.
- **Coral-Enhanced Model Adaptation:** Explore conversion of QAT-trained TensorFlow models to the Coral Edge TPU compiler format (EDGETPU-compatible int8 models). Analyze the trade-offs in throughput, memory bandwidth, and quantization granularity (per-channel vs. per-tensor) when executing on Coral hardware.
- **Advanced Compression Techniques:** Combine quantization with model pruning, knowledge distillation, and mixed-precision quantization to further optimize the accuracy–efficiency trade-off.
- **Expanded QAT Experiments:** Incorporate larger and more diverse calibration datasets and perform extended

fine-tuning epochs to evaluate long-term convergence behavior and generalization on the full ImageNet-1k dataset.

- **End-to-End Mobile Integration:** Develop an Android-based demonstration app that integrates TFLite or Edge TPU inference, real-time input capture, and energy profiling to validate usability under typical mobile workloads.

In summary, quantization—especially when coupled with QAT—enables high-accuracy, low-footprint deployment of EfficientNetV2 models on a range of embedded and edge devices. As future work progresses toward real hardware benchmarking and Coral Edge TPU optimization, this study lays the foundation for scalable, energy-efficient computer vision systems capable of operating entirely at the edge.

References

[1] M. Tan and Q. Le, "EfficientNetV2: Smaller Models and Faster Training," *arXiv preprint arXiv:2104.00298*, 2021.

[2] B. Jacob *et al.*, "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.

[3] TensorFlow Model Optimization Toolkit. [Online]. Available: https://www.tensorflow.org/model_optimization

[4] TensorFlow Lite. [Online]. Available: https://www.tensorflow.org/lite

[5] J. Deng *et al.*, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR*, 2009.

[6] A. Zhou *et al.*, "Value-Aware Quantization for Training and Inference of Neural Networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[7] Z. Yang and H. Lee, "Hardware-Aware Mixed Precision Quantization via Differentiable Search," in *Proceedings of Machine Learning Research*, vol. 222, 2024.

[8] J. Sun *et al.*, "A Survey of Quantization Methods for Deep Neural Networks," *arXiv preprint arXiv:2301.09780*, 2023.

[9] S. Han *et al.*, "PQK: Pruning, Quantization, and Knowledge Distillation for Compact Neural Networks," *arXiv preprint arXiv:2106.14681*, 2021.

[10] C. Chen *et al.*, "On Accelerating Edge AI: Optimizing Resource-Constrained Environments," *arXiv preprint arXiv:2501.15014*, 2025.

[11] Embedded Vision Summit, "A Survey of Model Compression Methods for Edge Deployment," 2023. [Online]. Available: https://embeddedvisionsummit.com

[12] University of California, Berkeley, "Hardware-Aware Quantization for Efficient Edge Inference," *Technical Report EECS-2022-231*, 2022.

[13] T. Pathirana, "Extreme Quantization of EfficientNetV2," GitHub repository, 2025. [Online]. Available: https://github.com/ThiwankaRoshen/ExtremeQuantizationOfEfficientNetV2