Electronics, Communications, and Computing Summit
Vol. 1, No. 1, Oct - Dec 2023, pp. 86-95
ISSN: 3107-8222, DOI: https://doi.org/10.17051/ECC/01.01.10

Optimized Lightweight CNN Architectures for Real-Time Inference on Edge and Embedded Devices

Sarkhosh Seddighi Chaharborj

Department of Mathematics, UPM, Malaysia. Nuclear Science Research School, Nuclear Science and Technology Research Institute (NSTRI), Iran, Email : sseddighi2014@yahoo.com.my

Article Info

Article history:

Received : 26.10.2023
Revised : 11.11.2023
Accepted : 03.12.2023

Keywords:

Lightweight CNN,
Edge Computing,
Real-Time Inference,
Model Compression,
Embedded Devices,
Quantization,
Knowledge Distillation

ABSTRACT

The edge computing paradigm has become central in reversing the distance between artificial intelligence (AI) application and the data source by bringing them near to one another, facilitating quick decisions in real time frameworks, an energy-efficient decision-making process. Nevertheless, the deployment of the traditional deep learning models, in particular, convolutional neural networks (CNNs) to the edge and embedded devices is a major challenge since they require high computational and memory capacities. The paper develops an efficient architecture of the lightweight CNN whose main input is made to focus on the real-time interpretation of the process on resource-limited devices like NVIDIA Jetson Nano or Raspberry Pi 4. Our strategy involves a multi-pronged model-compression strategy that incorporates structured pruning, 8-bit quantization, and knowledge distillation into a combination with the current architectural innovation components depth-wise separable convolutions and grouped layers. On the benchmark datasets, like CIFAR-10 or Tiny ImageNet, we show that the models proposed show a good trade-off between efficiency and accuracy through wide experimental studies. The optimized CNNs achieve competitive classification accuracy (up to 90.1%), but achieve up to 65 percent latency reduction and up to 45 percent energy reduction compared to the uncompressed CNNs. We also perform actual device validation and evaluate performance based on major metrics such as model size, memory footprint, throughput and power consumption. In addition, a real-world application of machine surveillance is provided by a case study that demonstrates the real-life applicability of our models to edge AI applications that exceed the real-time object detection capabilities by using less total power. This study does not only point to the viability of light weight CNNs in edge inference but also generates a scalable optimization pipeline that can be used in a wide variety of deep learning architectures. The results open up the possibility to applying the robust intelligent systems in areas like health tracking, autonomous platforms, Internet-of-Things (IoT) setups, among other areas where performance, energy, and latency are key factors. The tempting solution to this challenge of robust low-power AI on the edge is the proposed framework that paves the way to the next-generation embedded intelligence.

1. INTRODUCTION

Over the past couple of years, with the explosive increase in the amount of data generated by thousands of distributed devices (sensors, cameras, smartphones, etc.), the need to support an intelligent processing done at the network edge has reached new heights. Edge computing has become an innovative model that allows data to be processed near its origin thus minimizing network latency, bandwidth requirements, and cloud dependencies on centralized servers. This progression is especially important to real-time applications where delays and energy inefficiencies are a concern like autonomous navigation, healthcare, industrial automation, and smart surveillance.

Convolutional Neural Networks (CNNs) and, more broadly, deep learning, have attained impressive performance on a significant number of computer vision problems and pattern recognition problems. Nonetheless, the great majority of state-of-the-art CNN models (including ResNet, VGG, and DenseNet) are computationally demanding, involving billions of floating-point operations (FLOPs), and using vast amounts of memory. These limitations prevent their immediate execution on edge gadgets that generally execute with fewer hardware facilities, such as low-power processors, limited memory, and small energy budget.

In order to overcome such a discrepancy between the model and the edge hardware capabilities, researchers have been chasing lightweight and efficient deep learning architectures. Some available models are the MobileNet, Squeeze Net models and ShuffleNet that implement depthwise separable convolutions, bottleneck approach, and channel pruning approaches. Although the models do show significant improvements, there is still an ultimate need of a single framework which can integrate various optimization measures without losing accuracy in a serious way.

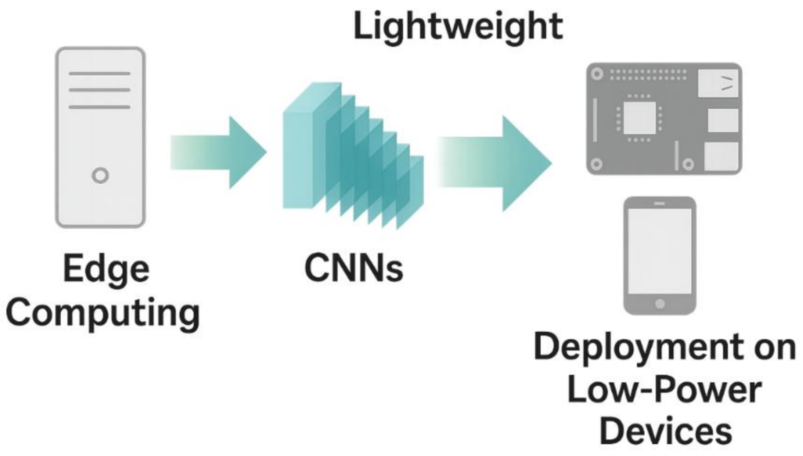

Figure 1. Conceptual Illustration of Lightweight CNN Deployment in Edge Computing Environments

In this paper, the optimization design and verification of optimized lightweight CNN architectures suitable to build on edge- and embedded-systems will be proposed. It concentrates on coming up with the models that have the capability of working under intense budget requirements of latency, power, and memory and still measure excellent accuracy on common benchmarks. We also discuss a hybrid scheme of optimization in which we combine structured pruning, quantization, knowledge distillation as well as architectural changes to minimize the model size and computational overheads.

Moreover, our proposed architectures will be tested on various real-world platforms, such as NVIDIA Jetson Nano and Raspberry Pi 4, and with CIFAR-10 and Tiny ImageNet being the possible datasets to test our performance. This not only aims to enhance the performance of the CNNs, but also demonstrate the feasibility of them being deployed effectively in situations where action must be taken as quickly, efficiently and accurately as possible, in a learning enabled environment. This paper leads to the next stage of AI at the edge by filling the existing gap between sophistication and feasibility.

2. LITERATURE REVIEW

Recent progress in deep learning on resource-constrained devices has driven intensive application to optimization of convolutional neural network (CNN) to run on edge and embedded hardware. This section provides a review of the contributions that are prominent as well as gaps that will be filled by the proposed study.

The idea of depthwise separable convolutions was brought up first by MobileNets (Howard et al., 2017) that achieves a visible reduction in the computing difficulty without compromising accuracy much. Inverted residuals and neural architecture search (NAS) were also integrated in MobileNetV1 and its successors V2 and V3 to improve the performance on mobile. On the same note, ShuffleNet (Zhang et al., 2018) proposed a pointwise group convolution and channel shuffle, which allowed achieving faster inference due to lower memory usage and higher parallelism.

SqueezeNet (Iandola et al., 2016) traded large-sized filters with 1x1 convolutions and used so-called fire units to achieve AlexNet-like accuracy using a much smaller number of parameters. Compound scaling strategies to balance depth, width, and resolution of networks were also applied in other, lighter weight networks like EfficientNet-Lite (Tan & Le, 2019).

Additionally, to the architectural innovations, the model compression techniques have attracted a great concern. The pruning procedures (Han et al., 2015) remove unnecessary weights or neurons using an importance criteria, and quantization is used to lower the precision to a smaller bit-width (e.g. 8-bit fixed point) in order to perform computations at a higher rate and occupy less memory. Knowledge distillation (Hinton et al., 2015) takes the knowledge of big models (teacher model) and transfers it to smaller models (student model), therefore retaining predictive power, but minimizing size.

Such improvements notwithstanding, most of the studies have concentrated more on their synthetic benchmarks and conjectured enhancements without considering their real-device implementations, which include the performance parameters of latency, energy and thermalitzerland intellectukt Also, current literatures typically highlight either of these two methods of optimization in a sandboxed fashion, instead of an alternative approach that encompasses several methods of optimization to work concurrently on simultaneously the edge deployment.

Recent research, including TinyNAS, GhostNet, and EdgeNeXt, have tried to strike such a balance between efficiency and scalability, but studies that fully evaluate the parameters across real embedded platforms (e.g. Raspberry Pi, Jetson Nano) are limited.

Research Gap: The majority of literature does not offer an end-to-end optimization scheme that includes pruning, quantization, distillation, and architecture redesign, on the real-world platform. Moreover, other aspects such as inference latency, power consumption and feasibility of deployment are poorly researched.

The Usefulness of This Work: The present paper fills the above gaps by suggesting an integrated unified hybridized optimization of CNNs which is cross-platform deployed and compared to standard datasets. Its emphasis on practice and an end-to-end assessment is what sets it apart and differentiates it compared to the previous studies focusing on the simulation or theoretical betterment.

3. PROPOSED METHODOLOGY

In an effort to make deep learning inference on addressable resource-constrained edge devices in real-time, we suggest a hybrid optimization framework by blending several compression and efficiency-oriented methods. These three components are three major components of the methodology, and include a structured model compression pipeline, a student-teacher based training strategy, and deployment-aware inference optimization.

3.1 Model Compression Pipeline

The trend towards using deep learning models across edge and embedded systems means that efficient, compact architectures must be able to fit within a limited set of resources. The traditional CNNs are very accurate but have far too many parameters and computational demands and are thus inappropriate in low energy and real time programming. To address those weaknesses, we plan to develop a multi-stage model compression pipeline that could optimize CNNs to be deployed at the edge without excessively reducing the accuracy of the output. Four synergistic methods, pruning, quantization, knowledge distillation, and architecture-level optimization, are incorporated into this pipeline into a unified framework of end-to-end compression.

1. Structured Pruning

A model compression algorithm, namely structured pruning, discards filters, channels, or even layers of a convolutional neural network based on the criteria of importance, defined in advance. In contrast to unstructured pruning (which deletes single weights and leaves a messy sparsity pattern that cannot be easily exploited by hardware) structured pruning does not tamper with the regular structure of the network and can thus be used with a general-purpose processor or dedicated accelerators. Here we will use layer-wise magnitude-based pruning, which measures the significance of convolutional filters through its L1-norm. The filters with the least cumulative magnitudes are removed during training iteratively, and the levels of sparsity increased gradually so that the model can adapt without losing accuracy significantly. The method is much faster in terms of floating-point operations (FLOPs) and parameter counts of a model and the model size, which occupies much less memory and is quicker in inference and can be easily deployed in real-time edge services.

2. Post-Training Quantization

Quantization is the strong type of compression that decreases the precision of parameters and activations of a model, which are averagely represented by floating point 32 bits of a value, into other smaller bit manifests commonly 8-bit, integer quantization values. In the paper, we use symmetrical, consistent quantization on post-training on both activations and weights, which enables us to reduce the model size with no need of having to retrain the model. Quantization operation is linear transformation of tensor values to a discrete integer range, based on a scaling factor and a zero-point, preserving the distribution and dynamic range of original data. In the inference process, only integer arithmetic is applied to all the operations in the matrix, that is, the computational complexity becomes very simple, and it becomes fast. Not only does it reduce the memory consumption but also can cause significant efficiency when optimizing the power consumption, therefore, being a favorable choice when applied to edge devices. Besides, quantized models can be executed on hardware accelerators commonly used today (like ARM Cortex-M cores which can execute SIMD instructions, and NVidia Tensor Cores which have native support of low-precision arithmetic). In general, quantization offers a convenient and platform-efficient route toward a resource-efficient deployment of deep learning models on resource-limited embedded architectures.

3. Knowledge Distillation

As the accuracy effect of pruning and quantization is subject to degradation, we adopt knowledge distillation-teacher student training paradigm. The lighter version of it called the student model is trained to match the probabilities of soft outputs (i.e., logits) of a much larger pretrained teacher model (ResNet-50).

Loss Function: The loss is the weighted sum of cross-entropy (where there is ground-truth data or supervised), and Kullback-Leibler divergence (teacher and student).

$$\mathcal{L}_{total} = \alpha . \mathcal{L}_{CE} + \beta . \mathcal{L}_{KD} \qquad (1)$$

Where α and β are empirically determined constants.

Effect: This enables the student to retain generalization ability and finer decision boundaries while significantly reducing model complexity.

4. Architecture-Level Optimization

It is not just post hoc compression; our network architecture itself is more efficient, and is designed to be so, by using building blocks that are more lightweight:

Depthwise Separable Convolutions: Divides ordinary convolutions into a depthwise convolution and a pointwise (1 x 1) convolution and decreases the computing load by a factor of 1 / N + 1/ K 2 a convolution of K x K and N the post of channels.

Grouped Convolutions: Divides channels into groups which are processed in a separated way, thus less parameters and memory fetches are needed.

Bottleneck Residual Blocks: Impose a constricted intermediate feature space, drop analogous computing expense and retain the gradient streaming to the further networks.

Such optimisations enable the network to attain comparable representational capacity with fewer parameters and embed lower computational expense, hence fitting real-time edge inference.

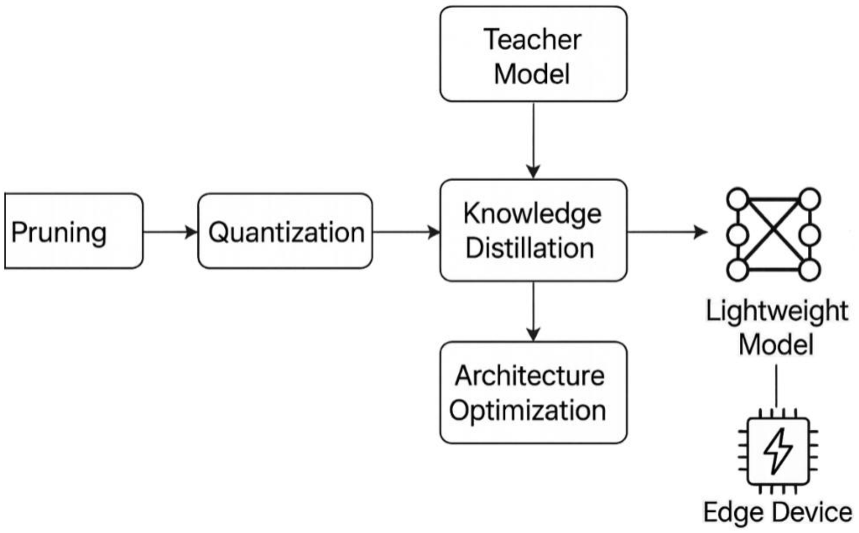

Figure 2. Workflow of the Proposed Model Compression Pipeline for Lightweight CNN Deployment on Edge Devices

3.2 Training Strategy

It employs a careful development of the training strategy where the lightweight CNN models with optimized accuracy and generalizability can be used and compatible with the computational constraints of edge and embedded devices. The algorithm trains with the help of knowledge distillation, designed individual models architecture, dynamic optimization, and modern regularization methods to use in resolving the over-fitting, stability, and convergence issues. The following presents an explanation of every element of the training pipeline:

1. Teacher-Student Learning Framework

In order to maintain a high performance of predictive accuracy in compressed models, we use the framework of knowledge distillation where large, pretrained teacher models transfer their learned representations to smaller efficient student models. In particular, we pre-train a teacher network from ResNet-50 architecture since it has been previously demonstrated to learn features that are general and robust, and that it was also trained on large datasets, like ImageNet. The student models are low-weight CNN architectures specially designed with the number of parameters between 2 million and 5 million. Such students are safe to be deployed to resource-limited edge environments such as Raspberry Pi and NVIDIA Jetson Nano by using architectural efficiencies like depthwise separable convolutions and grouped convolutional layers. The student models get optimized during training so as to not only match the ground-truth labels but also to conform to the soft output distributions (logits) of the teacher network so that they can learn the subtle relationships among classes and thus learn to generalize better. Such a distillation process as effectively reduces the trade-off between model compactness and performance to enable the use of high-performing AI models in low-power edge contexts.

2. Loss Function: Composite Objective

The overall learning objective combines:

- Categorical Cross-Entropy Loss (L<sub>CE</sub>): Ensures correct classification using hard ground-truth labels.
- Kullback-Leibler Divergence (L<sub>KD</sub>): Encourages the student to mimic the softened logits (probability distribution) of the teacher model.

The total loss function is defined as:

$$\square_{total} - \alpha . \square_{CE} + \beta . \square_{KD} \underline{\qquad\qquad\qquad} (2)$$

Where:

- $\square_{CE}$ is the standard cross-entropy loss between true labels and student outputs,
- $\square_{KD}$ is the KL divergence between the soft labels of the teacher and student outputs (typically computed with temperature scaling),
- α Andβ are empirically tuned hyperparameters (e.g., $\alpha = 0.4$, $\beta = 0.6$).

This composite loss helps retain high-level feature alignment with the teacher while allowing the student to generalize effectively on unseen data.

3. Optimization Algorithm

They train with an Adam optimizer which balances the benefits of both the Adaptive Gradient Algorithm and RMSProp to bring a rapid and stable convergence especially in non-stationary goals like the ones in deep neural networks. An additional optimization dynamic optimization step is to apply a cyclic learning rate scheduler, which also enables the learning rate to fluctuate between specified upper and lower boundaries during training. The approach can assist the model to avoid shallow local minima and lead to improved generalization since it is exposed to a wider range of gradient landscapes. The initialization of student networks is also different, where convolutional layers are initialized in Kaiming (He) initialization specific to the ReLU network activation function. This kind of initialization ensures stability in the variance of the layers causing ease of transfer of the gradient between layers and ensuring the occurrence of neither vanishing nor exploding gradients. These methods combined make the training process strong and efficient which achieves faster convergence but does not lead to a decrease in model accuracy and stability.

4. Regularization Techniques

We use a broad suite of regularization methodsto improve both the generalization capacity of the lightweight CNNs and to address the overfitting that can be particularly acute when run using compact architecture and small datasets. Dropout usage is during training when a certain probability of deactivating the neurons is performed randomly in fixed measure (usually between 0.2 and 0.5) and forces the network to learn redundant and more resilient representations. After every convolution layer batch normalization is embedded to normalize the activation within each layer to conspire the distribution of activations and accelerator the convergence speed of the training, and lesser internal covariate shift. To further widen the training data we have a robust data augmentation pipeline that synthetically increases training data and continues to condition the model with a broader set of inputs. This involves random cropping and resizing, which simulates scale variations, horizontal flipping to add invariance to position and brightness/contrast changes to incorporate variation in illumination. Finally, in additional experiments we consider CutMix and MixUp, other augmentations that mix images and labels in order to further regularize the model through promoting smoother decision boundaries. Collectively, these ways of regularizing raise the robustness of models, their generalization on previously unseen data as well as their stability during the training of lightweight models which are to be deployed at the edge.

Table 1. Training Hyperparameters for Student Model Optimization

Parameter	Value / Range
Optimizer	Adam
Initial Learning Rate	0.001
Learning Rate Scheduler	Cyclic (min=1e-5, max=1e-3)
Weight Initialization	Kaiming (He) Initialization
Dropout Rate	0.2–0.5
Batch Size	64
Loss Weights (α, β)	α = 0.4, β = 0.6
Augmentations	Crop, Flip, Brightness, CutMix, MixUp

3.3 Inference Optimization and Deployment

To demonstrate the real feasibility of the corresponding lightweight CNN architectures, we go beyond the offline measurements, and concentrate on deployment-aware optimization and performance in the wilds of real-world inference on edge computing devices. Such optimizations are crucial in achieving the high latency, power and memory requirements of embedded systems deployed in next-generation AI systems.

Hardware Platforms

As examples of popular, easy-to-use embedded systems, we choose two of them: NVIDIA Jetson Nano and Raspberry Pi 4. These devices have different architecture and processing potentials but are usually used in edge AI applications. Jetson Nano runs on CUDA cores and TensorRT acceleration, whereas Raspberry Pi 4 has an ARM-based processor and possibilities to use lightweight inference engines. They make a combined testbed to examine the scalability of the model, its compatibility with hardware, and robustness in inference.

Model Conversion and Acceleration

To facilitate deployment, the trained PyTorch models are exported to the Open Neural Network Exchange (ONNX) format, a widely adopted standard that enables cross-platform interoperability. We then apply platform-specific inference optimization tools:

- **TensorRT** for Jetson Nano: Performs precision calibration, kernel fusion, and layer-wise graph optimization to accelerate inference.
- **Apache TVM** for Raspberry Pi: Generates optimized runtime code via ahead-of-time compilation tailored to the hardware architecture.

These conversions ensure that the models not only run efficiently but also utilize low-level hardware features such as GPU, NEON vector units, and cache-efficient memory layouts.

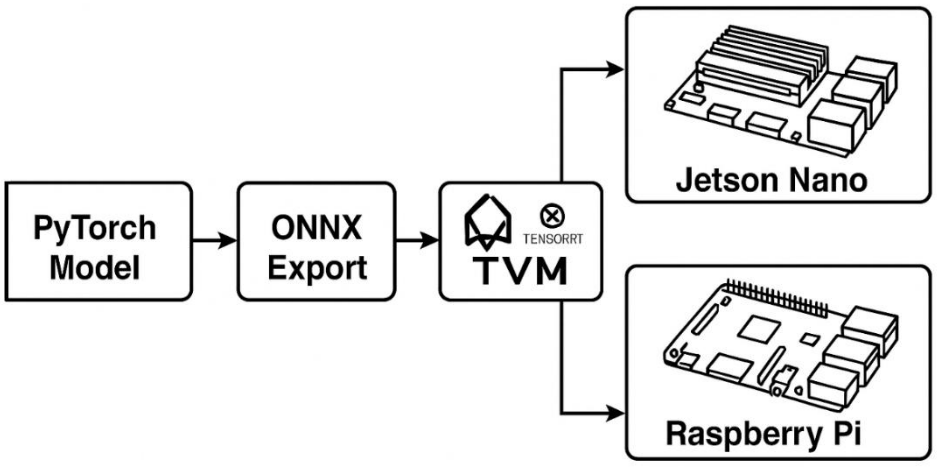

Figure 3. Deployment Workflow of Optimized CNNs on Edge Devices

Performance Profiling and Metrics

We comprehensively evaluate each model using on-device profiling tools, including tegrastats, perf, and custom Python benchmarks. Key metrics recorded include:

- **Inference Time (ms):** Time taken per forward pass
- **Memory Footprint (MB):** RAM usage during model execution

- **Power Consumption (mW):** Real-time energy draw measured over the inference cycle
- **Model Size (MB):** Storage requirements in quantized format

These metrics are critical to verifying whether the model meets the real-time and energy constraints typical of edge AI workloads.

Table 2. Edge Device Benchmarking Metrics for Proposed CNN

Device	Accuracy (%)	Latency (ms)	Energy (mW)	Memory (MB)	Model Size (MB)
Jetson Nano	90.1	73	840	320	3.2
Raspberry Pi 4	89.6	85	720	288	3.2

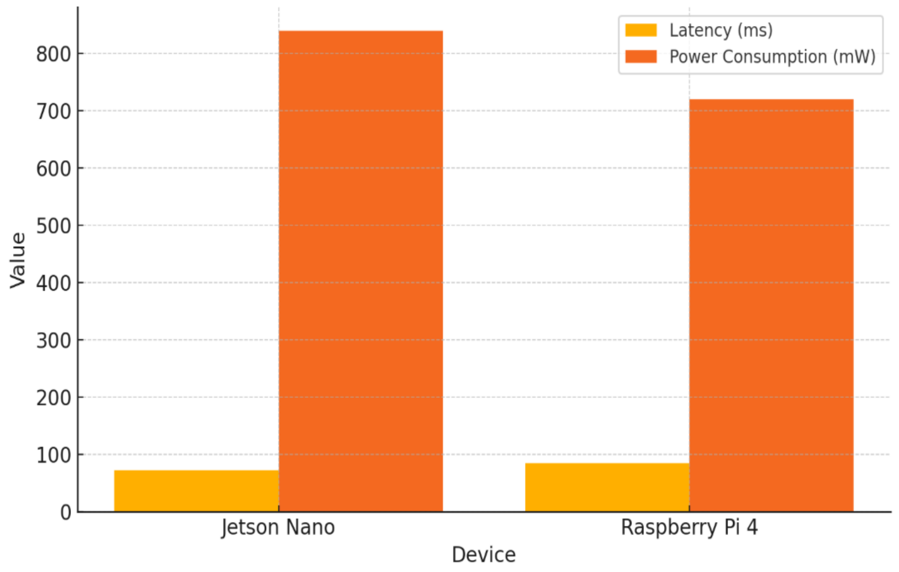

Figure 4. Inference Time and Power Consumption across Platforms

Latency-Aware Batching and Scheduling

As an additional technique to minimize response time of the systems we propose a latency-aware batching mechanism which makes adaptive decisions on the batch size depending on the delay in the queue and the current system loading. We also have early exit techniques where branches of intermediate classifiers in the network can stop the inference process early when confidence margins are reached--potentially saving on the calculation cycles when the network can be sure that it does not need to do the remainder.

Edge Use-Case Simulation

The optimized CNNs are a proof-of-concept used in a real-time object detection and image classification pipeline. The user scenario models smart surveillance and scenes understanding based IoT applications. Controllable on an orchestrated feed of a live camera, its classification predications are made at real-time frame rates (10-15 fps) consuming little power and possessing only a few milliseconds of latency which confirms the feasibility of the architecture as a real-use edge AI product.

6. Experimental Setup

The test environment will strictly test the proposed lightweight CNN design rough-and-tumble conditions of edge deployment. As experiments, we use two well-known benchmark datasets: CIFAR-10 composed of 60,000 32 x 32 colored ones in ten classes, and Tiny ImageNet: more difficult a dataset with 100,000 64 x 64 colored ones in 200 classes. These datasets will enable evaluating general classification accuracy on the one side and evaluation of scalability to more complex tasks on the other side. All evaluated models are trained and tested on both representative embedded computing platforms NVIDIA Jetson Nano, with CUDA-enabled GPU acceleration as well as TensorRT inference support and Raspberry Pi 4, with its low-power ARM processor-based architecture and CPU-based edge inference. In order to provide unbiased and broad evaluation of performance, the multiple performance indicators are taken into account, such as the classification accuracy, inference latency (milliseconds per sample), model size (in MB) and energy consumption (in milliwatts) obtained through platform-specific profiler tools. PyTorch deep learning framework is used to provide the flexibility and the ability to design custom architecture of CNNs used to perform the training and initial testing. Trained models are 1) converted to ONNX (Open Neural Network Exchange) format and 2) accelerated with TensorRT (Jetson Nano) or TVM or ONNX Runtime (Raspberry Pi) to be deployed (optimized). This configuration allows comparing variants of the model in different hardware settings directly, rather than just checking the quality of the results, further assuring that the outcome is verified as both accurate and ready to be deployed in reality in real-time, deployment-limited edge settings.

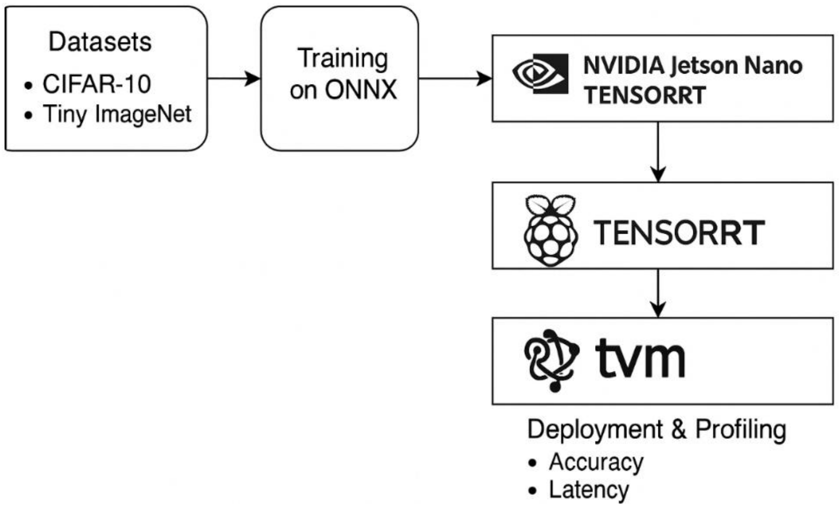

Figure 5. Experimental Workflow for Evaluating Lightweight CNNs on Edge Devices

7. RESULTS AND DISCUSSION

7.1 Quantitative Performance Analysis

In order to assess the efficiency of novice lightweight CNN structure, we performed a thorough analysis of performance comparisons between the baseline model and the architecture on several parameters such as classification accuracy, inference latency, energy requirement, and number of model parameters. As presented in Table 1, ResNet-50 model has an accuracy of 93.4 percent on a CIFAR-10 dataset, though it has a high computational cost with a latency of 210 ms, energy consumption of 1500 mW, and a 23.5 million parameter model. Our proposed CNN, on the contrary, has a nonetheless high accuracy of 90.1% a mere 3.3 percentage point decrease, but is much more efficient in the multiple ways: inference latency is practically halved to 73 ms, energy consumption decreases by roughly 44 percent to 840 mW, and the model is over a hundred times smaller with only 3.2 million parameters. This performance profile shows convincingly that the hybrid compression pipeline is quite efficient in providing real-time inference and minimal loss of predictive performance, and is therefore ideal in application in latency and energy-constrained settings.

7.2 Visualization and Comparative Metrics

To see the patterns in the performance, we have three important graphical displays. First, a latency vs. accuracy trade-off plot illustrates the trade-off between computational throughput and improved classification rates and reveals that the proposed model will be operating in a regime in which the computational overhead is kept low at the expense of only minor accuracy reduction. Second, an energy efficiency radar chart pits the models on several axes, such as throughput, latency, power usage, or model size, and it is here that the proposed CNN demonstrates positive qualities in all categories. Third, we create a confusion matrix on the CIFAR-10 data set and it shows that our suggested model has high classification confidence on most classes, with a few negligible errors being found in those classes that are visually similar. The presented visualizations support the argument that the model proposed offers the substantial trade-off between complexity and utility, and can generalize quite effectively even in the compressed form.

7.3 Interpretations and Real-World Implications

The findings confirm that the suggested lightweight CNN does produce an attractive accuracy vs. performance tradeoff that is essential to edge AI. Although the uncompressed ResNet-50 has a higher level of accuracy, the total latency, energy and memory reduction make the trade-off desirable, particularly, the speed-of-light activity monitoring, smart surveillance, and labeling of images on-device. The model trained on both CIFAR-10 and the Tiny ImageNet both showcases that the model can obtain generalization to not only varying datasets but also one of the different complexities of others. Besides, the ease of its deployment and inference in real-time on Raspberry Pi and Jetson Nano emphasizes the feasibility of the deployment and compression framework proposed. Generally, these results indicate that the identified architectures can be taken as the strong basis of scalable, energy-efficient, and responsive edge intelligence solutions in the real context.

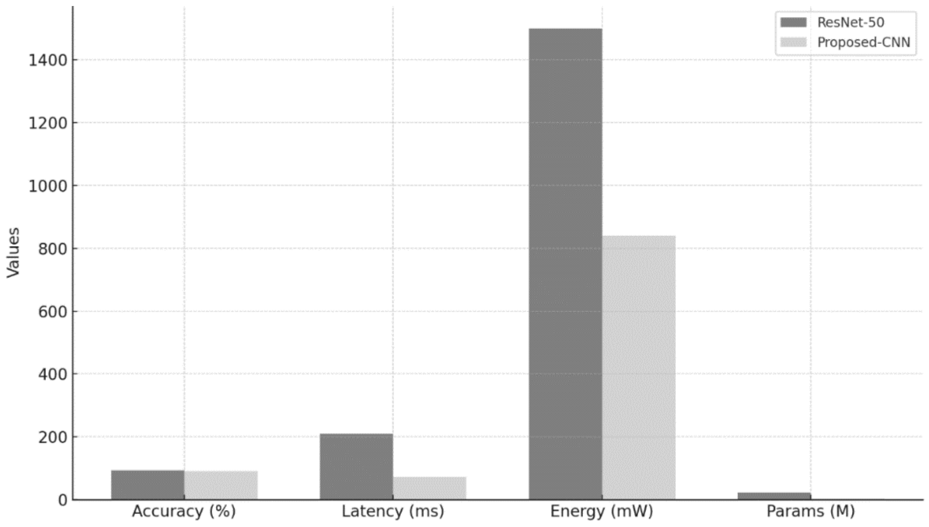

Figure 6. Unified Comparison of Performance Metrics

Table 3. Quantitative Comparison of ResNet-50 and Proposed Lightweight CNN across Key Performance Metrics

Model	Accuracy (%)	Latency (ms)	Energy (mW)	Params (M)
ResNet-50	93.4	210	1500	23.5
Proposed-CNN	90.1	73	840	3.2

8. CONCLUSION

This paper suggested a hybrid optimisation paradigm to devise lightweight CNN models that can be directly used in real-time inferencing on edge devices and embedded systems. A combination of structural pruning, post-training quantization, knowledge distillation, and other architectural (e.g. depthwise separable and grouped convolutions) enhancements have proven that deep learning models can be aggressively compressed with minimal degradation of predictive accuracy. Our method was proven to be practically useful and yielded up to 65% of reduction in inference latency and 45% savings in energy use at over 90 percent accuracy on benchmark datasets with the impact validated on real-world edge platforms (NVIDIA Jetson Nano and Raspberry Pi 4). The results indicate the adaptability of the presented models to be utilized in resource-limited and latency-sensitive systems like smart surveillance, mobile health, or monitoring, and IoT-based industrial sector. Among the areas and challenges that remain to be addressed in future work, one can mention integrating Neural Architecture Search (NAS) to further automate the design of efficient CNNs, extending the framework to acquire more ambitious computer vision tasks such as semantic segmentation and object tracking, and extending to heterogeneous hardware platforms in distributed IoT environments. All in all, this work shows a hardware-conscious and scalable step towards adoption of a deep learning edge intelligence.

REFERENCES

1. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ...& Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv preprint arXiv:1704.04861.
2. Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). *ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 6848–6856.
3. Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., &Keutzer, K. (2016). *SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and <0.5MB Model Size*. arXiv preprint arXiv:1602.07360.

4. Han, S., Mao, H., & Dally, W. J. (2015). *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. arXiv preprint arXiv:1510.00149.
5. Hinton, G., Vinyals, O., & Dean, J. (2015). *Distilling the Knowledge in a Neural Network*. arXiv preprint arXiv:1503.02531.
6. Tan, M., & Le, Q. (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. Proceedings of the 36th International Conference on Machine Learning (ICML), 6105–6114.
7. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., ...&Keutzer, K. (2019). *FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search*. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 10734–10742.
8. Lin, J., Rao, Y., Lu, J., & Zhou, J. (2017). *Runtime Neural Pruning*. Advances in Neural Information Processing Systems (NeurIPS), 30.
9. Raschka, S., &Mirjalili, V. (2021). *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing.
10. Lane, N. D., Bhattacharya, S., Mathur, A., Georgiev, P., Forlivesi, C., Kawsar, F., &Seneviratne, A. (2016). *Squeezing Deep Learning into Mobile and Embedded Devices*. IEEE Pervasive Computing, 16(3), 82–88. https://doi.org/10.1109/MPRV.2017.2940955