Hardware/Software Co-Design for Keyword Spotting on Edge Devices

PURDUE UNIVERSITY FORT WAYNE

Jacob Bushur

Advisor: Dr. Chao Chen

Department of Electrical and Computer Engineering

TensorFlow

Introduction

Digital assistants like Amazon Alexa, Apple Siri, Google Assistant, and Microsoft Cortana have enabled a novel method of interacting with electronic devices. Using automatic speech recognition, digital assistants enable users to operate devices simply by speaking commands. Since 2012, artificial neural networks have substantially improved automatic speech recognition. However, neural networks do not allow large, complex natural language processing models to run on resource-constrained embedded systems like smart speakers and smartphones. Instead, digital assistants perform keyword spotting, where the neural network is only trained to recognize a relatively small set of words.

Even with the reduced scope of keyword spotting compared to natural language processing, designing a neural network to run on edge devices with the performance of commercial digital assistants is not trivial. Companies like Amazon, Apple, Google, and Microsoft employ teams of dedicated developers to design and train their neural networks. However, creating an effective custom neural network architecture is prohibitive for almost everyone else.

The goal of this research project is to explore how the resource constraints of embedded systems affect the performance of existing neural network architectures that have been adapted to perform keyword spotting.

Methodology

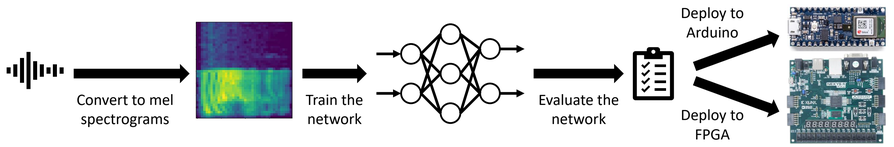

Fig. 1. Methodology Overview.

Implementation:

- The Google Speech Commands dataset [1] was selected for training and evaluation. To prepare the dataset, the 105,829 one-second audio recordings were transformed into 40x40 mel spectrograms.
- Six different neural network architectures were implemented in Python using the TensorFlow machine learning library. These architectures included feedforward fully-connected neural network (FFCN), depthwise separable convolutional neural network (DS-CNN) [2], ResNet [3], DenseNet [4], Inception [5], and CENet [6].
- Each network was adapted from its original paper to accept a 40x40 mel spectrogram as input and classify it as one of 35 keywords.

Training:

A total of 70 unique neural network models of different architectures and sizes were trained for 100 epochs.

Evaluation:

Post-training quantization was applied to each model, creating a version where the weights and operations used 8-bit signed integers instead of 32-bit floating-point numbers. Both model versions were evaluated for size and accuracy. The TensorFlow Lite benchmark tool was also used to measure the models' memory consumption and inference latency.

Deployment:

After evaluation, one of the DS-CNN models was deployed on an Arduino Nano 33 BLE and on a Digilent Nexys4 FPGA using CFU Playground.

Table 1. Neural Network Memory and Latency Statistics. For each set of neural network models, the peak memory usage and average inference latency were measured.

Architecture	Quantization	Peak Memory (MB)			Inference Latency (us)		
		Max	Min	Average	Max	Min	Average
FFCN	None	5.730	5.680	5.702	18.130	15.038	16.482
FFCN	8-bit Integer	5.094	5.023	5.062	8.663	7.264	8.116
DS-CNN	None	6.129	4.996	5.674	402.205	49.570	209.870
DS-CNN	8-bit Integer	5.836	4.988	5.464	351.975	72.441	198.077
ResNet	None	6.418	5.645	6.016	641.389	193.195	193.195
ResNet	8-bit Integer	6.180	5.738	5.954	486.624	239.679	239.679
DenseNet	None	7.027	5.629	6.332	617.123	160.345	160.345
DenseNet	8-bit Integer	6.582	5.762	6.208	588.441	224.248	224.248
Inception	None	6.961	5.910	6.450	770.405	205.215	205.215
Inception	8-bit Integer	6.742	5.750	6.341	649.030	266.512	266.512
CENet	None	8.816	5.883	6.839	892.887	140.050	140.050
CENet	8-bit Integer	7.527	5.770	6.413	966.854	241.022	241.022

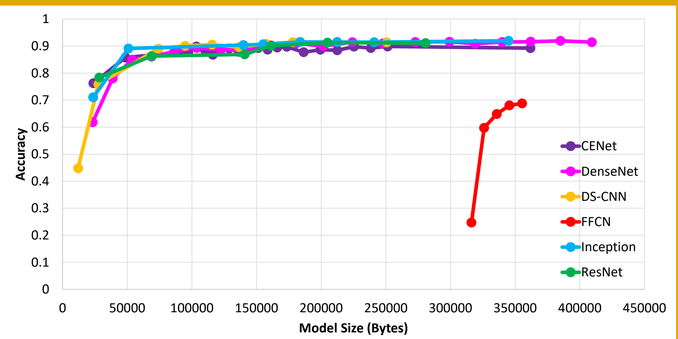

Fig. 2. Neural Network Accuracy vs. Model Size. Models used 32-bit floating-point values for weights and operations (i.e., no quantization).

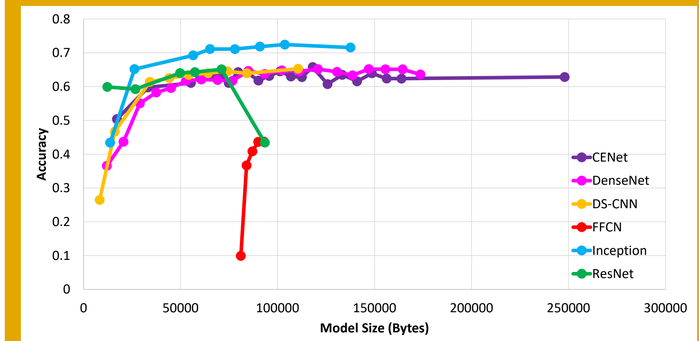

Fig. 3. Neural Network Accuracy vs. Mode Size. Models were converted after training to use 8-bit signed integers for weights and operations via post-training quantization.

Results

Evaluation:

Accuracy

Every unquantized model (except for FFCN models) above 50kB performed similarly, reaching approximately 90% accuracy. However, performance diverged as the model size dropped below 50kB. Inception, CENet, ResNet, and DS-CNN models clustered around 75% accuracy, beating a similarly sized DenseNet model. Integer quantized models behaved differently. At model sizes above 25kB, the Inception architecture outperformed every other architecture by approximately 5% in accuracy. However, at model sizes below 25kB, ResNet was the clear frontrunner, beating the next closest model by about 10% in accuracy.

Memory Consumption

For non-quantized models, CENet consumed the most memory, followed by Inception, DenseNet, ResNet, FFCN, and DS-CNN. This order held for the quantized models except for FFCN and DS-CNN, whose order was flipped.

Inference Latency

Regardless of quantization, every network remained below the 1ms mark. However, the quantized models were generally slower than their non-quantized counterparts. This was likely caused by the extra multiplication and addition operations required with quantization and by the CPU's floating-point unit (FPU) accelerating floating-point operations on the computer running the benchmark.

Deployment:

The xxd command was used to convert the smallest quantized DS-CNN model into an array of bytes which was then inserted into the keyword spotting example in the Arduino Harvard_TinyMLx library. An FPGA custom function unit (CFU) for a RISC-V soft CPU was also successfully generated using CFU Playground.

Conclusion

In conclusion, six different neural network architectures were implemented in Python using TensorFlow. For each architecture, models were trained and evaluated in terms of accuracy, memory consumption, inference latency, and size. Additionally, the smallest DS-CNN model was converted to a byte array and run on an Arduino 33 BLE. An FPGA custom function unit was also generated using CFU Playground.

This research serves as a guide to hardware/software co-design techniques for implementing efficient on-device keyword spotting on resource-constrained systems. This work can be extended to evaluate neural network architectures for other edge device applications such as visual wake word detection and anomaly detection.

References & Acknowledgements

[1] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *arXiv*, Apr. 2018, doi: https://doi.org/10.48550/ARXIV.1804.03209.
[2] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv*, Apr. 2017, doi: https://doi.org/10.48550/ARXIV.1704.04861.
[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv*, Dec. 2015, doi: https://doi.org/10.48550/ARXIV.1512.03385.
[4] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," *arXiv*, Aug. 2016, doi: https://doi.org/10.48550/ARXIV.1608.06993.
[5] C. Szegedy *et al.*, "Going Deeper with Convolutions," *arXiv*, Sep. 2014, doi: https://doi.org/10.48550/ARXIV.1409.4842.
[6] X. Chen, S. Yin, D. Song, P. Ouyang, L. Liu, and S. Wei, "Small-footprint Keyword Spotting with Graph Convolutional Network," *arXiv*, Dec. 2019, doi: https://doi.org/10.48550/ARXIV.1912.05124.
[7] S. Prakash *et al.*, "CFU Playground: Full-Stack Open-Source Framework for Tiny Machine Learning (tinyML) Acceleration on FPGAs," *arXiv*, Jan. 2022, doi: https://doi.org/10.48550/ARXIV.2201.01863.

Special thanks to the Office of Graduate Studies for supporting this research through the Graduate Research Assistantship and to the Google Cloud Research Credits Program for cloud computing resources.