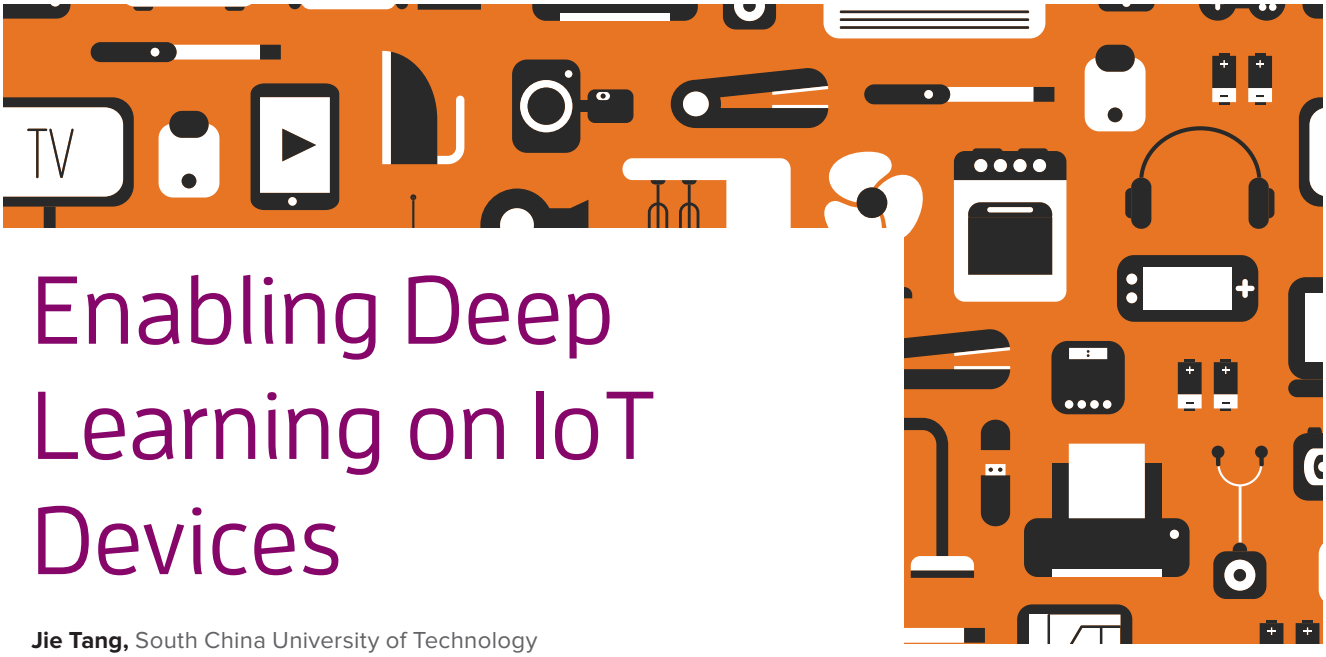

Enabling Deep Learning on IoT Devices

Jie Tang, South China University of Technology
Dawei Sun, Tsinghua University and PerceptIn
Shaoshan Liu, PerceptIn
Jean-Luc Gaudiot, University of California, Irvine

Deep learning can enable Internet of Things (IoT) devices to interpret unstructured multimedia data and intelligently react to both user and environmental events but has demanding performance and power requirements. The authors explore two ways to successfully integrate deep learning with low-power IoT products.

FROM THE EDITOR

Machine learning (ML) is applicable to many of the use cases that surround IoT products. However, if cloud services are necessary to supply ML functionality, it complicates the end-to-end system design that must also allow for low-latency interaction and disconnected operation. This article addresses moving ML to an IoT platform, and examines how the increased power and resources can be mitigated. *–Roy Want*

In recent years, increasing numbers of Internet of Things (IoT) products have appeared on the market that capture environmental data and use traditional machine-learning technologies to understand it. An example is Google's Nest Learning Thermostat, which records temperature data in a structured way and then applies algorithms to understand the patterns of its users' temperature preferences and schedules. However, it doesn't understand unstructured multimedia data, such as audio signals and visual images.

Emerging IoT devices apply more sophisticated deep-learning technologies that use neural networks to capture and understand their environments. Amazon Echo, for example, can understand human voice commands. It converts audio signals into a list of words, and then uses these words to search for relevant information. More recently, Microsoft's Windows IoT team released a face-recognition security system that

0018-9162/17/$33.00 © 2017 IEEE

EDITOR **ROY WANT**
Google; roywant@google.com

utilizes deep-learning technology to perform tasks such as unlocking a door when it recognizes its users' faces.

Deep-learning applications on IoT devices often have demanding real-time requirements. For example, security camera–based object-recognition tasks usually require a detection latency of less than 500 ms to capture and respond to target events—for example, strangers appearing inside a house—in a timely manner. Commercial smart IoT devices often offload intelligence to the cloud. Nonetheless, consistent good-quality network connections, which are only available at limited locations and at high cost, become a challenging constraint for these devices to abide by real-time requirements. A better choice is to enable deep learning on the device, which isn't affected by the connection quality.

However, enabling deep learning on the device side is difficult. Indeed, the main characteristic of mobile IoT devices is low power, which usually means limited computing power and small memory size. On the software side, to reduce the memory footprint, applications usually run on bare metal or use a very lightweight OS with a minimal set of third-party libraries. On the opposite end of the spectrum, deep learning requires high-performance computing and has high power consumption. Furthermore, existing deep-learning libraries usually call on many third-party libraries, which can be difficult to migrate to IoT devices.

The most widely used neural networks for processing deep-learning workloads are *convolution neural networks* (CNNs), which convert unstructured image data into structured object-label data. Generally, CNNs work as follows: first, a convolution layer scans the input image to generate a feature vector; second, an activation layer determines which feature in the vector should be activated for the image under inference; third, a pooling layer reduces the feature vector size; finally, a fully connected layer connects each potential label to all outputs of the pooling layer.

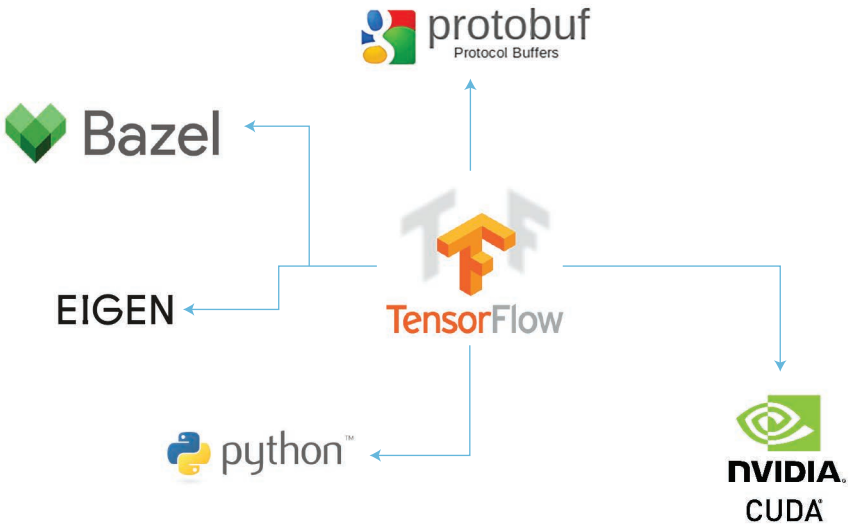

Figure 1. TensorFlow's dependency on third-party libraries. Migrating an existing deep-learning platform such as TensorFlow to Internet of Things (IoT) devices is not an easy process because of its dependency on numerous third-party libraries.

In this article, we describe how CNN inference engines can enable deep-learning tasks on IoT devices.

OFFLOADING TO THE CLOUD

For low-power IoT devices, the question arises as to whether offloading deep-learning workloads to the cloud is a feasible solution in terms of power consumption and performance. To answer this question, we implemented a CNN-based object-inference task on an Nvidia Jetson TX1 device and compared its performance and power consumption to the case of offloading these services to the cloud.[1]

To ascertain whether offloading can reduce power consumption and meet the real-time requirements of object-recognition tasks, we sent images to the cloud and then waited for the responses. Our results showed that, for object recognition, executing locally consumes 7 W compared to 2 W when offloading to the cloud. This shows that offloading is indeed an effective way to reduce power consumption.

However, offloading also leads to a minimum latency of 2 s, and latency can be as high as 5 s—which doesn't meet our real-time requirement of 500 ms. In addition, the high variability in latency makes the service very unreliable (these experiments were run in the US and China with comparable observations). This led us to conclude that with current Internet speeds, offloading to the cloud isn't yet a viable solution for real-time deep-learning tasks.

MIGRATING DEEP-LEARNING PLATFORMS TO EMBEDDED DEVICES

With offloading impractical, one option is to migrate existing deep-learning platforms to IoT devices. To this end, we built an IoT product with object-inference capability by migrating TensorFlow,[2] an open source deep-learning

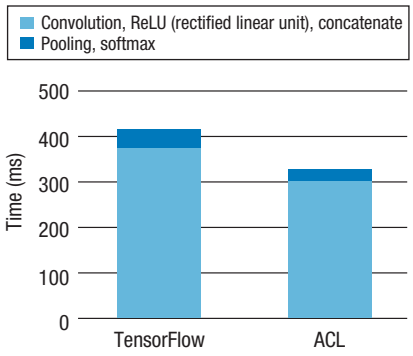

Figure 2. Performance of the SqueezeNet inference engine running on TensorFlow versus the SqueezeNet engine built with the ARM Compute Library (ACL). Building simple inference engines from scratch not only takes less development time but also outperforms existing deep-learning engines, such as TensorFlow.

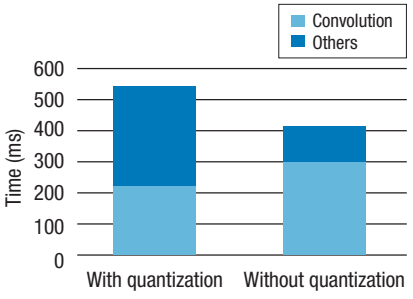

Figure 3. TensorFlow performance with and without vector quantization. Manual optimization on existing deep-learning platforms such as TensorFlow is difficult and might not bring significant performance improvements.

platform developed by Google, to Zuluko, our bare-metal ARM system on a chip (SoC). Zuluko contains four ARM v7 cores running at 1 GHz and 512 Mbytes of RAM, consumes about 3 W at peak power, and costs about $4. We chose TensorFlow as it delivers the best performance on ARM-Linux SoCs based on our research.

We expected to finish the migration within a few days; however, migrating TensorFlow was not easy due to its dependency on third-party libraries (see Figure 1). To reduce resource consumption, most IoT devices run on bare metal, and migrating all these dependencies can be a daunting task. It took a week of intensive effort before we successfully ran TensorFlow on Zuluko. This experience raised the question of whether it's more worthwhile to build a platform from scratch rather than port an existing one. Without basic building blocks such as a convolution operator, this is extremely difficult. In addition, an inference engine built from scratch might not outperform a well-tested deep-learning framework.

BUILDING AN INFERENCE ENGINE FROM SCRATCH

ARM recently announced their Compute Library (ACL; developer.arm.com/technologies/compute-library), a comprehensive collection of software functions implemented for the ARM Cortex-A family of CPU processors and the ARM Mali family of GPUs. Specifically, ACL provides the basic building blocks for CNNs, including activation, convolution, full and local connectivity, normalization, pooling, and softmax functionality. These functions are exactly what's needed to build an inference engine.

We used the ACL building blocks to construct a CNN inference engine with a SqueezeNet architecture,[3] which is suitable for embedded devices because of its low memory footprint. SqueezeNet utilizes a 1 × 1 convolution kernel to reduce the input size of the 3 × 3 convolution layer while maintaining similar inference accuracy. We then compared SqueezeNet engine performance to that of TensorFlow on Zuluko. To ensure a fair comparison, we enabled the ARM NEON vector computation optimization in TensorFlow, and used NEON-enabled building blocks when creating our SqueezeNet engine. By making sure both engines utilized the NEON vector computation, we ensured that any performance difference would have been caused exclusively by the platform itself. As Figure 2 shows, on average, it took 420 ms to process a 227 × 227 RGB image, while the SqueezeNet engine reduced the time to process the same image to 320 ms, a 25-percent speedup.

To better understand the source of the performance gain, we divided the processing time into two groups: the first group included convolution, ReLU (rectified linear unit) activation, and concatenation; the second included pooling and softmax functionality. The breakdown shown in Figure 2 indicates that the SqueezeNet engine outperformed TensorFlow by 23 percent in group 1 and 110 percent in group 2. As for resource utilization, when running on TensorFlow, the average CPU usage was 75 percent and the average memory usage was about 9 Mbytes; when running on the SqueezeNet engine, the average CPU usage was 90 percent and the average memory usage was about 10 Mbytes. There are two reasons for the performance improvement: first, the SqueezeNet engine provides better NEON optimization—because all ACL operators are developed using NEON-intrinsic operators directly—whereas TensorFlow relies on the ARM compiler to provide NEON optimization. Second, the TensorFlow platform itself likely introduces some performance overhead.

As a next step, we wanted to squeeze additional performance out of TensorFlow and see if it could outperform the SqueezeNet inference engine. Vector quantization is a technique that uses 8-bit weights to trade accuracy for performance. In addition, with 8-bit weights, we can use vector computations to process multiple data units with one instruction.[4] However, this optimization comes at a cost: it introduces re-quantization and de-quantization operations. We implemented this optimization in TensorFlow; Figure 3 compares performance with and without optimization. Using vector quantization improved convolution performance by 25 percent but also added significant overhead because of the de-quantization and re-quantization operations; overall,

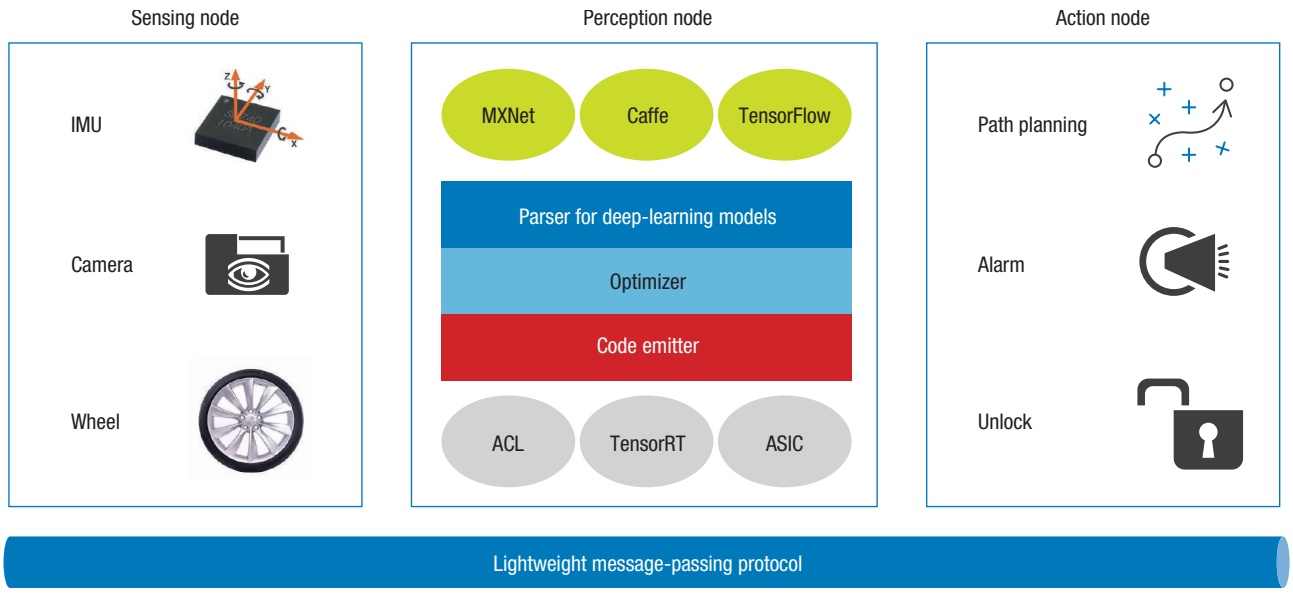

Figure 4. IoT device service architecture. We need a new system architecture to enable deep learning on IoT devices: first, we need to directly compile and optimize deep-learning models to executable codes on the target device; second, we need an extremely lightweight OS to enable multitasking and efficient communications between different tasks. IMU: Inertial measurement unit.

it slowed down the entire inference process by more than 100 ms.

Network connectivity can be easily lost, so we wanted to make sure that some form of intelligence is local to the device, allowing it to continue to operate in the event of an ISP/network failure. Nonetheless, intelligence requires high-performance computing and leads to high power consumption.

While offloading intelligence to the cloud reduces IoT device power consumption, it doesn't meet real-time requirements. Existing deep-learning platforms are built for generality—suitable for both training and inference tasks—which means that these engines aren't optimized for embedded-inference tasks. They also depend on other third-party libraries that aren't readily available on bare-metal embedded systems, making them very hard to migrate.

By constructing embedded CNN inference engines with the ACL building blocks, we can deliver high performance by making full use of the SoC's heterogeneous computing resources. Thus, the question becomes whether it's easier to migrate existing engines or build them from scratch. Our experience shows that if the model is simple, it's much easier to build them from scratch. As the model gets increasingly complicated, we might hit a point where porting an existing engine is more efficient. However, it's highly unlikely that complex models will be needed for embedded-inference tasks. Therefore, we conclude that a built-from-scratch embedded inference engine could be a viable way to bring deep-learning capabilities to IoT devices.

NEXT STEPS

Instead of manually building models from scratch, we need a more convenient way to enable deep learning on IoT devices. One solution is to implement a deep-learning model compiler that optimizes a given model into executable code on the target platform. As shown in the center panel of Figure 4, the front end of such a compiler can parse models from major deep-learning platforms including MXNet, Caffe, TensorFlow, and more. The optimizer can then perform additional optimizations, including model pruning, quantization, and heterogeneous execution. After optimization, the code emitter generates executable code on the target platform, which can be ACL (for ARM devices), TensorRT (for Nvidia GPUs), and other ASIC devices.

The NNVM project (github.com/dmlc/nnvm) is a first step toward this goal. We have successfully extended NNVM to generate code so that we can use ACL to accelerate deep-learning operations on ARM devices. An additional benefit of this approach is that even as the models get more complicated, we can still easily enable them on IoT devices.

Current IoT devices usually perform single tasks because of computing resource limitations. However, we anticipate that we'll soon have low-power IoT devices that are able to perform multiple tasks (for instance, our Zuluko device comprises four cores). To enable these devices, we need an extremely lightweight message-passing protocol to connect different services.

As shown in Figure 4, the basic services for IoT devices include sensing, perception, and action. A sensing node involves processing raw sensor

JIE TANG is an associate professor in the School of Computer Science and Engineering at South China University of Technology. Contact her at cstangjie@scut.edu.cn.

DAWEI SUN is a researcher in the Department of Automation at Tsinghua University and a software engineer at PerceptIn, working on deep learning and cloud infrastructures, autonomous robots, and embedded systems. Contact him at sdw14@mails.tsinghua.edu.cn.

SHAOSHAN LIU is chairman and cofounder of PerceptIn, working on developing the next-generation robotics platform. Contact him at shaoshan.liu@perceptin.io.

JEAN-LUC GAUDIOT is a professor in the Electrical Engineering and Computer Science Department at the University of California, Irvine, and currently serves as the 2017 president of the IEEE Computer Society. Contact him at gaudiot@uci.edu.

data from, for example, a camera, an inertial measurement unit, and wheel odometry. A perception node consumes processed sensor data and produces an interpretation of the captured information, such as object labels and device position. An action node contains a set of rules that determine how to respond when a specific event is detected, such as to unlock a door when the owner's face is detected or to adjust a robot's motion path when an obstacle is detected. Nanomsg (nanomsg .org) is an extremely lightweight message-passing framework that's suitable for this task. Another choice is the Robot Operating System,[5] although we find it too heavy for IoT devices in terms of memory footprint and computing resource requirements.

To efficiently integrate more deep-learning workloads with IoT devices, we have developed our own OS consisting of a sensor interface for consuming standard sensor inputs, a compiler based on NNVM to compile and optimize existing deep-learning models into executable code, and a message-passing framework based on Nanomsg to connect all the nodes.

We hope this article will inspire researchers and developers alike to design smarter IoT systems with more intelligence than was previously thought possible in a small embedded device.

ACKNOWLEDGMENTS

This work is partly supported by South China University of Technology start-up grant no. D61600470, Guangzhou Technology grant no. 201707010148, and the National Science Foundation (NSF) under grant no. XPS-1439165. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

1. J. Tang, Y. Ren, and S. Liu, "Real-Time Robot Localization, Vision, and Speech Recognition on Nvidia Jetson TX1," arXiv, 2017; arxiv.org/abs/1705.10945.
2. M. Abadi et al., "Tensorflow: Large-scale Machine Learning on Heterogeneous Distributed Systems," arXiv, 2016; arxiv.org/abs/1603.04467.
3. F.N. Iandola et al., "SqueezeNet: AlexNet-level Accuracy with 50× Fewer Parameters and< 0.5 MB Model Size," arXiv, 2016; arxiv.org/abs/1602.07360.
4. S. Han, H. Mao, and W.J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," arXiv, 2015; arxiv.org/abs/1510.00149.
5. M. Quigley et al., "ROS: An Open-Source Robot Operating System," *ICRA Workshop Open Source Software*, vol. 3, no. 3.2, 2009; www .willowgarage.com/sites/default /files/icraoss09-ROS.pdf.