# Squeezing Deep Learning into Mobile and Embedded Devices

***Nicholas D. Lane,*** *University College London and Nokia Bell Labs*
***Sourav Bhattacharya and Akhil Mathur,*** *Nokia Bell Labs*
***Petko Georgiev,*** *Google DeepMind*
***Claudio Forlivesi and Fahim Kawsar,*** *Nokia Bell Labs*

In a relatively short time, deep learning principles and algorithms have transformed how the world processes, models, and interprets data.[1] For discriminative learning tasks routinely integrated into mobile and embedded systems—such as recognizing spoken words, objects, and faces—deep networks have been the state of the art for many years. Looking ahead to future device-based applications of learning, deep models are proving pivotal in the development of control algorithms for autonomous cars and drones (for example, for deep reinforcement learning). Deep models are also expanding into the area of core system issues—improving, for example, methods for encryption and compression.[2]

The blending of learning algorithms and mobile computing taking place today is only the beginning. We believe, in particular, that deep learning will play a prominent role in the evolution of smart devices (such as phones, watches, and embedded sensors) moving forward. It is therefore of paramount importance that we advance our understanding of how to simply and efficiently integrate current—and future—deep learning breakthroughs within constrained computing platforms (for more information, see the "Deep Learning under Constrained Devices" sidebar). This, along with continued research into the use of deep neural networks that support the diverse inference needs of sensor systems, will help produce radical improvements in how on-device context modeling and activity recognition is performed.

The emergence of mobile and embedded forms of deep learning has been slowed by the extreme resource overhead that it can easily introduce. Deep networks often contain hundreds of layers of interconnected nodes, and performing a single classification from a frame of sensor data can require computations over potentially hundreds of millions of parameters. Model representations and inference algorithms originally conceived for deep networks can easily overwhelm the resources of constrained platforms. In response to this resource barrier, the past 18 months have seen a surge in the investigation of resource-efficient deep learning for mobile and embedded platforms.

Promising early results are appearing across many domains, including hardware,[3,4] systems,[5,6] and learning algorithms.[7,8] Likely to further accelerate progress is the rate at which existing commercially supported deep learning tools, libraries, and frameworks have begun to address the specific needs of constrained devices (examples include TensorFlow, Caffe2, SNPE, Compute Library from Google, Facebook, Qualcomm, and ARM). These tools are starting to offer building blocks that enable fundamental research in this area by simplifying key steps such as runtime support on Android devices, processor-optimized low/mix precision matrix multiplication, or access to often unavailable heterogeneous device processors such as digital signal processors (DSPs) or GPUs.

In this short article, we provide an overview of the progress we have made toward overcoming a variety of core challenges facing deep learning for mobile and embedded devices, while also attempting to connect our findings to those of the wider community in the area. This discussion is largely focused on improvements seen within on-device execution of deep networks, which assumes the models are trained off-device. This is because execution (that is, inference) is the critical first step toward deep learning support, and it's the focus of almost all existing work, although exploration of on-device training has begun. Finally, given space constraints, we only superficially touch upon the ways in which deep learning is changing the face of activity and context recognition,[9] again limiting our focus to on-device examples.

Published by the IEEE CS ■ 1536-1268/17/$33.00 © 2017 IEEE

## DEEP LEARNING UNDER CONSTRAINED DEVICES

The deep learning revolution has been powered by major advances in training algorithms, leaps in the availability of computing resources (primarily GPUs), and of course increased access to large-scale data. But at the core of any on-device, use of deep learning remains a neural architecture that must be efficiently executed.

### PRIMER ON DEEP LEARNING INFERENCE AND ARCHITECTURES

Although a variety of deep model architectures have been developed, here we briefly describe two popular networks (shown in Figure A): deep neural networks (DNNs) and convolutional neural networks (CNNs). The role of training algorithms is to set the parameters of these neural architectures based on available data. This process is almost always assumed to occur off-device, and so the device itself is concerned with efficient inference.

Under a DNN, inference follows a feed-forward approach that operates on input data segments in isolation. The algorithm starts at the input layer and moves layer-wise sequentially while updating the activation states of all nodes within each layer. The process finishes at the output layer when all nodes in the layer have been updated. Finally, the inferred class is identified as the class corresponding to the output layer node with the greatest activation value. DNNs are often used in familiar mobile sensing tasks, such as spoken keyword spotting or identifying a speaker, but they're also use in extracting high-level human behaviors and contexts from inertial, location,[1] and (again) audio sensors.

Primarily used for vision and image-related tasks, CNNs are an alternative formulation of deep learning models. A CNN model contains one or more convolutional layers, pooling or subsampling layers, and fully connected layers. The objective of these layers is to extract simple representations from the input data and convert the representations into a more complex representation at much coarser resolutions within the subsequent layers. Lastly, fully connected layers often are used to help a CNN make predictions. CNNs can recognize a place type (such as a kitchen), accurately estimate age and gender, or more broadly recognize daily events from even noisy complex images, even those from wearable cameras.[2] Certain designs of CNN architectures like AlexNet or VGG[3] can be specialized to support many distinct tasks, and so their particular performance on constrained devices can become particularly important.

### SYSTEM RESOURCE BOTTLENECKS

Model training is not the only computationally challenging process in deep learning. Even executing the straightforward inferencing step using a parameter-heavy model on a resource-limited device must overcome several challenges, including limited memory, limited computational power, and an unusually large inference time.[4,5] For example, deep models often have millions of parameters, and their storage on limited memory devices quickly becomes infeasible. Under low memory conditions, neural networks are often represented with low-precision parameters (8-bit or 16-bit) or by quantizing the weights of the architecture. Remarkably, even when heavily compressed with such methods, deep architectures can retain much of their accuracy. However, due to runtime memory limits, performing inference might still require frequent paging operations.

Inference time is also impacted by the overall number of computations. The availability of multiple cores and low-power processors on mobile platforms can be used to parallelize partial state updates of nodes to improve the inference time. Moreover, inferences often come with real-time requirements. Local execution of the memory- and computation-optimized models can potentially meet the requirements, overcoming intermittent connectivity problems prevalent in cloud-based systems.

Also, when running deep models continuously on embedded or wearable devices, high energy efficiency is crucial for maintaining a prolonged battery life. The energy consumption, among many things, mainly depends on the amount of computations, the use of low-power processors—such as digital signal processors (DSPs)—and the number of cache accesses. Thus, energy optimization requires a detailed understanding of the deep-model-execution pipeline on heterogeneous hardware platforms.

### REFERENCES

1. J. Zhang et al. "DNN-Based Prediction Model for Spatio-Temporal Data," *Proc. 24th ACM SIGSPATIAL Int'l Conf. Advances in Geographic Information Systems* (GIS), 2016, article no. 92.

2. D. Castro et al., "Predicting Daily Activities from Egocentric Images Using Deep Learning," *Proc. 2015 ACM Int'l Symp. Wearable Computers*, 2015, pp. 75–82.

3. K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *Proc. Int'l Conf. Learning Representations* (ICLR), 2015; https://arxiv.org/pdf/1409.1556.pdf.

4. N.D. Lane et al., "An Early Resource Characterization of Deep Learning on Wearables, Smartphones, and Internet-of-Things Devices," *Proc. 2015 Int'l Workshop on Internet of Things towards Applications* (IoT-App), 2015, pp. 7–12.

5. J. Albericio et al., "Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing," *Proc. 43rd Int'l Symp. Computer Architecture* (ISCA), 2016, pp. 1–13; https://doi.org/10.1109/ISCA.2016.11.

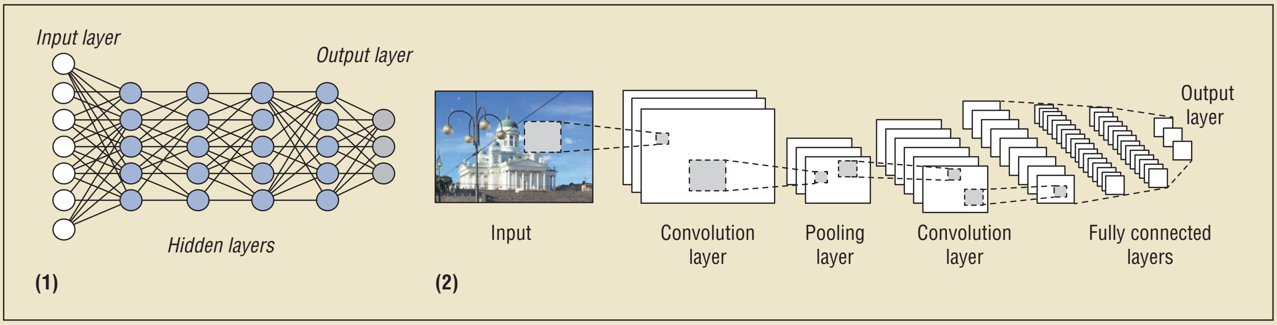


**Figure A. Two popular neural network architectures: (1) deep neural networks (DNNs) and (2) convolutional neural networks (CNNs).**

## EARLY SMARTPHONE SENSING RESULTS

In late 2014, we began our exploration into deep learning, starting with smartphones. These early investigations were motivated by two questions. First, could typical mobile and embedded sensing tasks, such as activity recognition and context sensing, be improved by the same deep learning approaches that were revolutionizing so many other inference domains? Second, how feasible was it to use these notoriously resource-heavy modeling techniques for user devices such as smartphones?

Fast forward to today, and deep networks for activity recognition—and smartphone sensing in general—have become much more mainstream. Researchers are developing powerful methods to train various deep architectures, raising the level of accuracy for models of human behavior.[9] Similarly, the ability to push neural networks into phone DSPs for low-power operation, a core innovation in our 2014 work (discussed next),[10] is an upcoming feature of Google's TensorFlow in partnership with Qualcomm.[11]

### Deep Networks for Activity Recognition and Audio Sensing

We devised early deep learning solutions for well-known smartphone recognition tasks to quantify the benefits for on-device sensing.[10,12] A unique aspect of our approach was our focus on building constrained deep networks suitable for mobile and embedded devices. We wanted to know if deep learning was a viable and transformative replacement for the existing classifiers of mobile context and activities, grounded in shallow learning techniques. A core finding of our work was that for a range of sensing tasks, generic (nontask specific) deep networks could outperform state-of-the-art hand-selected features and shallow models—even when the deep networks were constrained to a size that made them more resource efficient than shallow alternatives.[10]

We then applied these findings to the audio domain and developed DeepEar,[12] a system for training and executing small-footprint deep neural networks (DNNs)—specifically, Restricted Boltzmann Machines (RBMs)—which were able to classify many audio contexts despite being a modest size of 2.3 million parameters each. As Table 1 summarizes, we stress-tested DeepEar, as well as a range of task-specific mobile audio classifiers, and on average, the accuracy was more than 30 percent higher for each task using DeepEar, even though each DNN was designed to execute not only within the CPU but even in the phone's DSP, a critical factor we explain next.

### Low-Power Deep Networks via Heterogeneous Compute

Just as GPUs are a primary enabler for scaling up the training of larger and larger deep networks, we have found that non-CPU heterogeneous processors (such as DSPs) play a key role in scaling down deep networks for constrained devices. The DSPs in phones, for example, are sufficiently energy-efficient to compute on sensor data almost continuously while still supporting a device battery life beyond 24 hours.

Motivated by such efficiencies, we executed our proposed activity and audio targeting deep networks within the constraints of phone DSPs of the time—in

TABLE 1

A comparison of accuracy between our low-resource generic-task deep classifiers and existing hand-designed and task-specific (shallow) classifiers from the literature for various mobile sensing tasks. Note, reported microphone accuracy is lower than might be expected (for example, speaker identification), because experiments were conducted under severe acoustic conditions. (Experimental setup and classifier specifications appear elsewhere.[12,13] For each shallow classifier, we indicate the original venue of publication.)

| Device type | Sensor | Sensing task | Task-specific shallow classifier (%) | Generic-task deep classifier (%) |
|---|---|---|---|---|
| Smartphone | Microphone | Ambient scene detection | 81 (baseline from MobiSys 2009) | 86 |
| Smartphone | Microphone | Stress detection | 62 (UbiComp 2012) | 82 |
| Smartphone | Microphone | Emotion recognition | 72 (UbiComp 2010) | 81 |
| Smartphone | Microphone | Speaker identification | 36 (Pervasive 2011) | 57 |
| Smartwatch | Accelerometer, gyroscope | Gesture recognition | 68 (Activity Recognition in Pervasive Intelligent Environments 2010) | 72 |
| Smartwatch | Accelerometer, gyroscope | Physical activity recognition | 82 (SenSys 2010) | 93 |
| Smartwatch | Light sensor, magnetic sensor, microphone, temperature sensor, proximity sensor | Location detection (indoor/outdoor) | 87 (SenSys 2014) | 94 |

particular, within memory footprints of just 8 Mbytes (using the Hexagon DSP of the Qualcomm Snapdragon 800).[10,12] DeepEar, under the Hexagon DSP, could run for 24 hours while using just 6 percent of a typical phone battery life with interleaved DNNs supporting four different audio tasks. In our follow-up system, DeepX,[5] we showed that by dividing models across a wide range of commodity phone processors (CPUs, DSPs, and GPUs), such efficiency gains were possible for not just small-scale DNNs but also other architectures, including even large image-based deep networks (such as the CNN AlexNet with 61 million parameters).

Our algorithms in DeepX allowed neural networks to be partitioned across different processor types within a local device using a runtime form of model compression that used singular value decomposition (SVD) to cope with processor constraints and minimize inter-processor overhead. Our smartphone prototype (on the Snapdragon 800) showed that this let various well-known deep models execute with efficiencies far in excess of baselines based on single processors or model compression alone (our prototype was up to seven times more energy efficient, for approximately a five percent loss in accuracy).

## VGG AND MORE ON A SMARTWATCH

As techniques for deep learning on phones have matured, we have started studying how these issues manifest under smartwatches. The capabilities (compute and memory) of watches, coarsely speaking, lag phones often by one or two device generations; a typical Android smartwatch has not only 512 Mbytes of RAM and a multicore CPU but also a GPU and DSP. Watches are also natural for performing continuous and diverse behavior and context inferences—unlike phones, which can spend most of the day in pockets and bags. These two factors make it both conceivable and warranted for watches to join phones in performing nontrivial deep learning.

### Transforming Watches from Smart to Deep

As in DeepEar,[12] our first proposed watch,[13] deep learning models were applicable to a range of common watch sensing tasks (shown in Table 1). Just as the DeepEar experiments had done for the smartphone audio domain, we demonstrated that typical inertial and wearable sensor data (such as accelerometer, barometer, and magnetometer data), fed into DNNs suitable in size for watches (around 200,000 parameters), could outperform existing task-agnostic classifiers from the literature.

This result further added to the understanding of feature representation learning by showing that these DNN models, produced by a single (off-watch) training framework, could outperform custom per-task combinations of hand-selected features and shallow models. On average, tasks were more than 7 percent more accurate compared to the best performing manually constructed classifier, while exerting a reasonable overhead.[13] For example, a commodity LG smartwatch could run one such RBM at 3 Hz and still maintain a 32-hour battery life.

### Leveraging Layer Separation and Compression

Most examples of deep models—designed to process images, for example—dwarf the DNNs just described. The well-known VGG architecture can perform object recognition (and many other visual tasks) but at a cost of 138 million parameters or more. To prove the potential of smartwatches to support such demanding deep models, we showed that the VGG can be run locally on commodity smartwatches with a loss of approximately 3 percent accuracy (a tuneable parameter).[7] This was achieved primarily through a method applicable to any CNN, which reduces the computational bottleneck of applying thousands of convolutional kernels through what we call kernel separation. This technique replaces the 2D kernels defined during training with a pair of 1D vertical and horizontal kernels that, when used together, produce a result that approximates that of the original 2D version.

We coupled this optimization with the earlier described SVD-based model-compression technique for the fully connected layers at the end of the CNN, which simplifies the description of how nodes are connected and allows a further reduction in the number of parameters. We studied this approach on commodity watches under a variety of deep models, with VGG being the most resource intensive.[7] VGG, for example, executes in just under 1.2 seconds (a 2.7 times gain over conventional implementations) on LG smartwatches. These results, under some of the heaviest examples of deep models, pair with our low-resource DNN-based findings to show how deep solutions can not only improve over shallow methods but also be adopted in watches.

## OVERCOMING SEVERE EMBEDDED CONSTRAINTS

As we have discussed, resource constraints present nontrivial barriers for deep learning on phones and watches. However, within embedded processors, these issues are magnified to extreme levels. Smartphones can address multiple Gbytes of RAM, but embedded processors, such as the ARM Cortex series, typically are limited to just hundreds or even tens of Kbytes. Similar resource differentials also extend into energy and compute domains. For these reasons, unlike the proliferation of phone-based deep learning in the last 18 months, few examples of deep learning under embedded constraints currently exist.

Toward filling this void, promising results are being seen in the form of binary deep architectures that are composed solely of 1-bit weights[14] instead of 32-bit or 16-bit parameters. Such architectures offer incredibly small models and remove the need for expensive multiplication operations, but their ability to perform well with real-world problems is still an open question. Solutions more closely tied to hardware will also undoubtedly play a key role in the area, such as the unique co-design opportunities for embedded-scale deep models that are built for field-programmable gate arrays processors,[3] or even emerging small form-factor deep learning accelerators (see, for example, https://uploads.movidius.com/1463156689-2016-04-29_VPU_ProductBrief.pdf).

### Sparse Compression for Embedded Processors

Our contribution to the embedded area has been to devise a new form of model compression[7] that enables conventional DNNs to both fit and execute within the embedded processors, such as the ARM Cortex M3, and even the ARM Cortex M0! With this technique, fully connected layers of the DNN are represented using a sparse dictionary. As shown in Figure 1, dense matrices that capture the pair-wise dependencies of nodes (that is, weight matrices) are replaced with a code-book and sparse matrix that, together, closely approximate the dense original. We discovered a sparse-coding formulation that lets this approximation (and therefore the model accuracy) remain high. The dictionary is trained from the initial model representation, and a large saving in computation and memory results because nonzero elements can be ignored.

Compute savings under our approach are even further magnified, because, at execution, high-efficiency sparse matrix multiplication algorithms can be adopted in favor of conventional varieties that assume dense matrices. Although this method is only applicable to fully connected layers, it addresses the central embedded bottleneck of model size and still remains broadly useful, because the operations optimized are a key component to alternatives such as recurrent and convolutional architectures.

### Experiences on the ARM Cortex

To measure the gains of our sparse-coding method for embedded processors, we tested DNNs for two audio tasks: speaker recognition and classification of the acoustic environments. We adopted an existing DNN architecture and training methods designed for low-resource platforms while still maximizing audio task robustness. Our findings showed, for example, that at the expense of 2 percent in accuracy, model compression by sparse coding can reduce these already optimized models by a factor of approximately 17 times for both tasks. In the case of speaker recognition, DNNs executed within our runtime that could leverage the sparsity of model representation showed a tenfold improvement in execution time within both ARM Cortex processors.

These gains make it feasible to run what are normally smartphone-class audio models in severely constrained processors. However, work remains to make deep models of this scale completely practical, because they still can't execute these models in real time—execution is still in the order of tens of seconds even to process a single five-second audio clip.

## LOCAL EXECUTION OF MULTIPLE DEEP MODELS

Virtually all of the progress made thus far in mobile and embedded deep learning assumes that a single model executes on a constrained device. This is natural, because even a single deep model can present considerable technical challenges. However, most devices and applications will need to execute multiple models as part of their daily operations. For example, a wearable camera likely won't just recognize objects; it will also identify people and track facial expressions.

Between-model optimization opportunities exist most often when the collection of models perform related tasks (like image models), because each is trained independently, which lets natural redundancies emerge. For example, models that perform face recognition and object recognition will both learn layers that perform a type of edge detection during training, even though this operation could, in theory, be shared. Optimization opportunities such as this present an important class of performance improvements that has received little attention thus far.

### Multiple Model Inference Pipeline

As a first step in addressing this issue, we designed an inference pipeline for wearables that targets the local execution of multiple image-based CNNs.[15] This

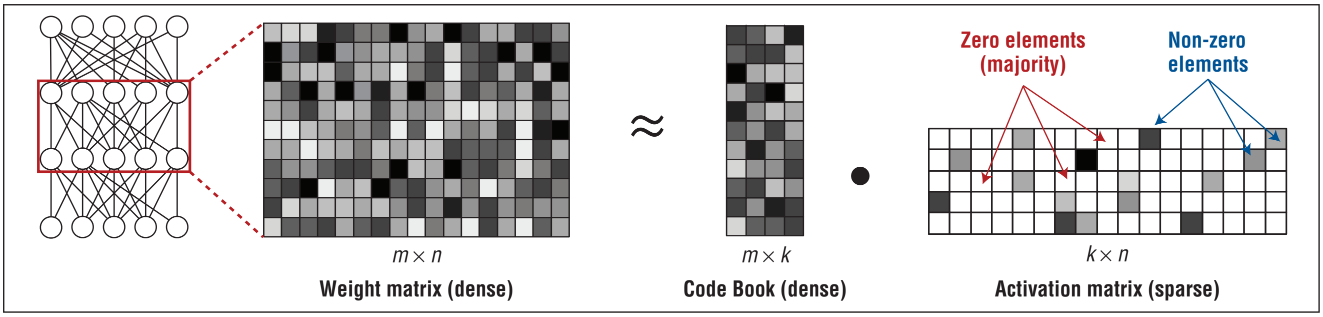


**Figure 1. Illustration of our sparse-coding approach that factorizes dense matrices typically necessary to describe the connectivity between layers. A single dense matrix is approximated with two matrices; one is the weight code-book and the other is the sparse layer connectivity descriptor. We note a similar factorization is used in DeepX (not shown), but sparse coding is replaced by a light-weight singular value decomposition (SVD)-based method.**

pipeline builds on a single fundamental optimization insight—namely, that CNNs are comprised of both *computation-heavy* convolutional layers and *memory-heavy* fully connected layers. Although convolutional layers only lightly tax the memory resources, they are computationally demanding. In contrast, fully connected layers place the exact opposite resource demands.

Due to these orthogonal resource demands of memory and compute, it's possible to schedule and batch layers together from multiple models to better maximize the resources of constrained devices and avoid bottlenecks that prevent multiple deep models from being executed. Our layer-centric execution framework for the inference stage of multiple CNNs focuses on optimal scheduling and batching decisions for device performance with a global view of all models, while still adhering to the layer dependencies of the neural network architecture.

Beyond this core idea, the execution framework incorporates memory caching of frequently used fully connected layers, selective use of SVD-based compression (described earlier), and logic that identifies the visual similarity in consecutive images to avoid unnecessary operations. Although designed for CNNs, the underlying concepts of this pipeline can generalize to other deep architectures.

### DeepEye Wearable Camera

To study this multiple model pipeline, we integrated it within DeepEye—a prototype wearable camera based on a commodity processor (the Qualcomm Snapdragon 410) that offers execution of multiple CNN models without offloading computation to the cloud. DeepEye supports two use cases: lifelogging and vision assistance. Lifelogging seeks to log various everyday user experiences, with DeepEye realizing this through CNNs that can recognize objects, places, and faces and infer important image regions and how memorable an image is for the user. In contrast, vision assistance aims to help users who have low-vision capabilities by applying the same deep models that detect faces or objects, along with additional CNNs that infer age, gender, and emotions.

We compared the performance of DeepEye against the serial execution and single-model optimization alternatives. Experiments revealed that the latency for executing the multimodel inference pipeline is 10.10 seconds and 8.2 seconds for lifelogging and vision assistance, respectively (gains of 1.7 and 1.88 times over baselines, respectively).[15] These gains translate into a battery life of nearly 20 hours (1.4 times gain over the baseline), assuming images are captured every 30 seconds.

Deep learning on constrained devices, such as phones, watches, and even embedded sensors, is already well on its way to becoming mainstream. This is being enabled by a growing community of academic and industrial researchers who are bridging the worlds of machine learning, mobile systems, and hardware architecture.

Looking toward what is next, in the short term, we're likely to see continued leaps in activity and context-recognition accuracy, as insights from deep learning continue to propagate. We're also likely to see not just inference but also training being performed more routinely on devices. More fundamentally, applications of deep learning today are largely limited to classification tasks, yet the broader trend is for these algorithms to perform a wider range of computation. Within constrained devices, the potential definitely exists for them to begin to perform control and decision tasks, as well as more application logic, where their ability to learn and adapt dynamically to complex conditions might overcome some of the more brittle characteristics of sensory system behavior that have proven difficult to overcome.

**IEEE Computer Society Publications Office**
10662 Los Vaqueros Circle
Los Alamitos, CA 90720

**STAFF**

Lead Editor
**Brian Brannon**
bbrannon@computer.org

Content Editor
**Shani Murray**

Manager, Peer Review & Periodical Admin
**Hilda Carman**

Publications Coordinator
**pervasive@computer.org**

Contributors
**Dale Strok and**
**Teri Sullivan**

Director, Products & Services
**Evan Butterfield**

Senior Manager, Editorial Services
**Robin Baldwin**

Senior Business Development Manager
**Sandra Brown**

Digital Marketing Manager
**Marian Anderson**
manderson@computer.org

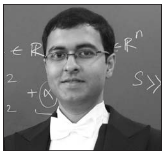
***IEEE Pervasive Computing*** (ISSN 1536-1268) is published quarterly by the IEEE Computer Society. IEEE Headquarters, Three Park Ave., 17th Floor, New York, NY 10016-5997; IEEE Computer Society Publications Office, 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1314, phone +1 714 821 8380; IEEE Computer Society Headquarters, 2001 L St., Ste. 700, Washington, DC 20036. Subscribe to *IEEE Pervasive Computing* by visiting www.computer.org/pervasive.

**Postmaster:** Send undelivered copies and address changes to IEEE Pervasive Computing, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854-4141. Periodicals postage paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8. Printed in the USA.

## REFERENCES

1. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.
2. G. Toderici et al., “Variable Rate Image Compression with Recurrent Neural Networks,” *Proc. Int’l Conf. Learning Representations* (ICLR), 2016; https://arxiv.org/abs/1511.06085.
3. S. Han et al., “ESE: Efficient Speech Recognition Engine for Sparse LSTM on FPGA,” *Proc. 2017 ACM/SIGDA Int’l Symp. Field-Programmable Gate Arrays* (FPGA), 2017, pp. 75–84.
4. R. LiKamWa et al., “RedEye: Analog ConvNet Image Sensor Architecture for Continuous Mobile Vision,” *Proc. Int’l Conf. ACM/IEEE 43rd Ann. Int’l Symp. Computer Architecture* (ISCA), 2016, pp. 255–266.
5. N. Lane et al., “DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices,” *Proc. 15th Int’l Conf. Information Processing in Sensor Networks* (IPSN), 2016, article no. 23.
6. S. Han et al., “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” *Proc. Int’l Conf. Learning Representations* (ICLR), 2016; https://arxiv.org/pdf/1510.00149.pdf.
7. S. Bhattacharya and N.D. Lane, “Sparsifying Deep Learning Layers for Constrained Resource Inference on Wearables,” *Proc. 14th ACM Conf. Embedded Network Sensor Systems* (SenSys), 2016, pp. 176–189.
8. G. Huang et al., “Densely Connected Convolutional Networks,” to appear in *Proc. 13th IEEE Conf. Computer Vision and Pattern Recognition* (CVPR), 2017.
9. N. Hammerla, S. Halloran, and T. Ploetz, “Deep, Convolutional, and Recurrent Models for Human Activity Recognition Using Wearables,” *Proc. Int’l Joint Conf. Artificial Intelligence* (IJCAI), 2016; https://arxiv.org/abs/1604.08880.
10. N.D. Lane and P. Georgiev, “Can Deep Learning Revolutionize Mobile Sensing?” *Proc. 16th Int’l Workshop Mobile Computing Systems and Applications*, (HotMobile), 2015, pp. 117–122.
11. Qualcomm, “TensorFlow Machine Learning Now Optimized for the Snapdragon 835 and Hexagon 682 DSP,” 9 Jan. 2017; www.qualcomm.com/news/snapdragon/2017/01/09/tensorflow-machine-learning-now-optimized-snapdragon-835-and-hexagon-682.
12. N.D. Lane, P. Georgiev, and L. Qendro, “DeepEar: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep Learning,” *Proc. 2015 ACM Int’l Joint Conf. Pervasive and Ubiquitous Computing* (UbiComp), 2015, pp. 283–294.
13. S. Bhattacharya and N.D. Lane, “From Smart to Deep: Robust Activity Recognition on Smartwatches Using Deep Learning,” *Proc. Workshop on Sensing Systems and Applications Using Wrist Worn Smart Devices* (WristSense), 2016; http://ieeexplore.ieee.org/document/7457169.
14. B. McDanel, S. Teerapittayanon, and H.T. Kung, “Embedded Binarized Neural Networks,” *Proc. Int’l Conf. Embedded Wireless Systems and Networks* (EWSN), 2017; www.eecs.harvard.edu/~htk/publication/2017-ewsn-mcdanel-teerapittayanon-kung.pdf.
15. A. Mathur et al., “DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models Using Wearable Commodity Hardware,” to appear in *Proc. 15th Int’l Conf. Mobile Systems, Applications, and Services* (MobiSys), 2017.

**Nicholas D. Lane** is a senior lecturer (associate professor) at University College London and a principal scientist at Nokia Bell Labs. Contact him at niclane@acm.org.



**Sourav Bhattacharya** is a research scientist at Nokia Bell Labs. Contact him at sourav.bhattacharya@nokia-bell-labs.com.

**Akhil Mathur** is a research scientist at Nokia Bell Labs. Contact him at akhil.mathur@nokia-bell-labs.com.

**Petko Georgiev** is a research engineer at Google DeepMind. Contact him at pig20@cam.ac.uk.

**Claudio Forlivesi** is a research engineer at Nokia Bell Labs. Contact him at claudio.forlivesi@nokia-bell-labs.com.

**Fahim Kawsar** leads the Internet of Things research at Nokia Bell Labs. Contact him at fahim.kawsar@gmail.com.