Hardware for Machine Learning: Challenges and Opportunities

(Invited Paper)

Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman, Zhengdong Zhang
Massachusetts Institute of Technology
Cambridge, MA 02139

***Abstract*—Machine learning plays a critical role in extracting meaningful information out of the zetabytes of sensor data collected every day. For some applications, the goal is to analyze and understand the data to identify trends (e.g., surveillance, portable/wearable electronics); in other applications, the goal is to take immediate action based the data (e.g., robotics/drones, self-driving cars, smart Internet of Things). For many of these applications, local embedded processing near the sensor is preferred over the cloud due to privacy or latency concerns, or limitations in the communication bandwidth. However, at the sensor there are often stringent constraints on energy consumption and cost in addition to throughput and accuracy requirements. Furthermore, flexibility is often required such that the processing can be adapted for different applications or environments (e.g., update the weights and model in the classifier). In many applications, machine learning often involves transforming the input data into a higher dimensional space, which, along with programmable weights, increases data movement and consequently energy consumption. In this paper, we will discuss how these challenges can be addressed at various levels of hardware design ranging from architecture, hardware-friendly algorithms, mixed-signal circuits, and advanced technologies (including memories and sensors).**

I. Introduction

This is the era of *big data*. More data has been created in the past two years than the entire history of the human race [1]. This is primarily driven by the exponential increase in the use of sensors (10 billion per year in 2013, expected to reach 1 trillion by 2020 [2]) and connected devices (6.4 billion in 2016, expected to reach 20.8 billion by 2020 [3]). These sensors and devices generate hundreds of zetabytes (10^{21} bytes) of data per year — petabytes (10^{15} bytes) per second [4].

Machine learning is needed to extract meaningful, and ideally actionable, information from this data. A significant amount of computation is required to analyze this data, which often happens in the cloud. However, given the sheer volume and rate at which data is being generated, and the high energy cost of communication and often limited bandwidth, there is an increasing need to perform the analysis locally near the sensor rather than sending the raw data to the cloud. Embedding machine learning at the edge also addresses important concerns related to privacy, latency and security.

II. Applications

Many applications can benefit from embedded machine learning ranging from multimedia to medical space. We will provide a few examples of areas that researchers have investigated; however, this paper will primarily focus on computer vision, specifically image classification, as a driving example.

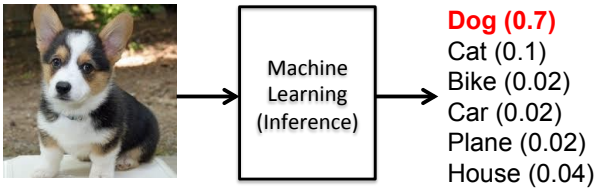


Fig. 1. Image classification.

A. Computer Vision

Video is arguably the biggest of the big data. It accounts for over 70% of today's Internet traffic [5]. For instance, over 800 million hours of video is collected daily worldwide for video surveillance [6]. In many applications (e.g., measuring wait times in stores, traffic patterns), it would be desirable to use computer vision to extract the meaningful information from the video right at the image sensor rather than in the cloud to reduce the communication cost. For other applications such as autonomous vehicles, drone navigation and robotics, local processing is desired since the latency and security risk of relying on the cloud are too high. However, video involves a large amount of data, which is computationally complex to process; thus, low cost hardware to analyze video is challenging yet critical to enabling these applications.

In computer vision, there are many different artificial intelligence (AI) tasks [7]. In this paper, we focus on image classification (Fig. 1), where the entire image is provided and the task is to determine which class of objects is in the image.

B. Speech Recognition

Speech recognition has significantly improved our ability to interact with electronic devices, such as smartphones. While currently most of the processing for applications such as Apple Siri and Amazon Alexa voice services is in the cloud, it is desirable to perform the recognition on the device itself to reduce latency and dependence on connectivity, and to increase privacy. Speech recognition is the first step before many other AI tasks such as machine translation, natural language processing, etc. Low power hardware for speech recognition is explored in [8, 9].

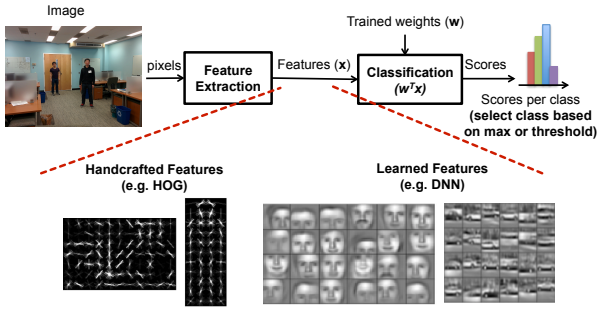


Fig. 2. Inference pipeline.

C. Medical

There is a strong clinical need to be able to monitor patients and to collect long-term data to help either detect/diagnose various diseases or monitor treatment. For instance, constant monitoring of ECG or EEG signals would be helpful in identifying cardiovascular diseases or detecting the onset of a seizure for epilepsy patients, respectively. In many cases, the devices are either wearable or implantable, and thus the energy consumption must be kept to a minimum. Using embedded machine learning to extract meaningful physiological signal and process it locally is explored in [10–12].

III. Machine Learning Basics

Machine learning is a form of artificial intelligence (AI) that can perform a task without being specifically programmed. Instead, it *learns* from previous examples of the given task during a process called *training*. After learning, the task is performed on new data through a process called *inference*. Machine learning is particularly useful for applications where the data is difficult to model analytically.

Training involves learning a set of weights from a dataset. When the data is labelled, it is referred to as supervised learning, which is currently the most widely-used approach. Inference involves performing a given task using the learned weights (e.g., classify an object in an image)[1]. In many cases, training is done in the cloud. Inference can also happen in the cloud; however, as previously discussed, for certain applications this is not desirable from the standpoint of communication, latency and privacy. Instead it is preferred that the inference occur locally on a device near the sensor. In these cases, the trained weights are downloaded from the cloud and stored in the device. Thus, the device needs be programmable in order to support a reasonable range of tasks.

A typical machine learning pipeline for inference can be broken down into two steps as shown in Fig. 2: Feature Extraction and Classification. Approaches such as deep neural networks (DNN) blur the distinction between these steps.

A. Feature Extraction

Feature extraction is used to transform the raw data into meaningful inputs for the given task. Traditionally, feature extraction was designed through a *hand-crafted* process by experts in the field. For instance, for object recognition in computer vision, it was observed that humans are sensitive to edges (i.e., gradients) in an image. As a result, many well-known computer vision algorithms use image gradient-based features such as Histogram of Oriented Gradients (HOG) [13] and Scale Invariant Feature Transform (SIFT) [14]. The challenge in designing these features is to make them robust to variations in illumination and noise.

B. Classification

The output of feature extraction is represented by a vector, which is mapped to a score using a classifier. Depending on the application, the score is either compared to a threshold to determine if an object is *present*, or compared to the other scores to determine the object *class*.

Techniques often used for classification include linear methods such as support vector machine (SVM) [15] and Softmax, and non-linear methods such as kernel-SVM [15] and Adaboost [16]. In many of these classifiers, the computation of the score is effectively a dot product of the features ($\vec{x}$) and the weights ($\vec{w}$) (i.e., $\sum_i w_i x_i$). As a result, much of the hardware research has been focused on reducing the cost of a multiply and accumulate (MAC).

C. Deep Neural Networks (DNN)

Rather than using hand-crafted features, the features can be learned directly from the data, similar to the weights in the classifier, such that the entire system is trained end-to-end. These *learned* features are used in a popular form of machine learning called deep neural networks (DNN), also known as deep learning [17]. DNN delivers higher accuracy than hand-crafted features on a variety of tasks [18] by mapping inputs to a high-dimensional space; however, it comes at the cost of high computational complexity.

There are many forms of DNN (e.g., convolutional neural networks, recurrent neural networks, etc.). For computer vision applications, DNNs are composed of multiple convolutional (CONV) layers [19] as shown in Fig. 3. With each layer, a higher-level abstraction of the input data, called a feature map, is extracted to preserve essential yet unique information. Modern DNNs are able to achieve superior performance by employing a very deep hierarchy of layers.

Fig. 4 shows an example of a convolution in DNNs. The 3-D inputs to each CONV layer are 2-D feature maps ($W \times H$) with multiple channels (C). For the first layer, the input would be the 2-D image itself with three channels, typically the red, green and blue color channels. Multiple 3-D filters (M filters with dimension $R \times S \times C$) are then convolved with the input feature maps, and each filter generates a channel in the output 3-D feature map ($E \times F$ with M channels). The same set of M filters is applied to a batch of N input feature maps. Thus there are N input feature maps and N output feature maps. In addition, a 1-D bias is added to the filtered result.

The output of the final CONV layer is processed by fully-connected (FC) layers for classification purposes. In FC layers,

[1]Machine learning can be used in a discriminative or generative manner. This paper focuses on the discriminative use.

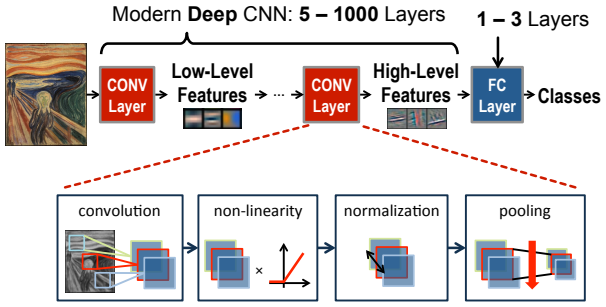


Fig. 3. Deep Neural Networks are composed of several convolutional layers followed by fully connected layers.

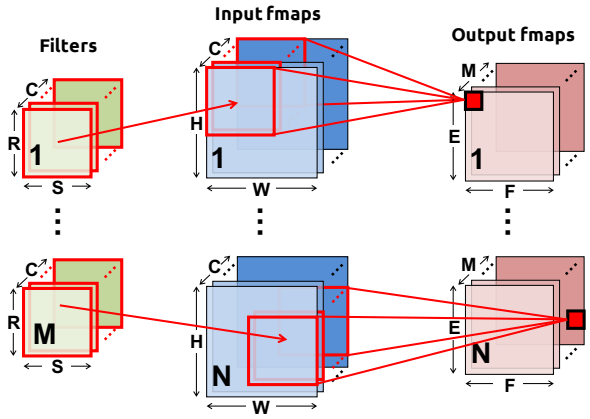


Fig. 4. Computation of a convolution in DNN.

the filter and input feature map are the same size, so that there is a different weight for each input pixel. The number of FC layers has been reduced from three to one in most recent DNNs [20, 21]. In between CONV and FC layers, additional layers can be optionally added, such as the pooling and normalization layers [22]. Each of the CONV and FC layers is also immediately followed by an activation layer, such as a rectified linear unit (ReLU) [23]. Convolutions account for over 90% of the run-time and energy consumption in DNNs.

Table I compares modern DNNs, with a popular neural net from the 1990s, LeNet-5 [24], in terms of number layers (depth), number of filters weights, and number of operations (i.e., MACs). Today's DNNs are several orders of magnitude larger in terms of compute and storage. A more detailed discussion on DNNs can be found in [25].

D. Complexity versus Difficulty of Task

It is important to factor in the difficulty of the task when comparing different machine learning methods. For instance, the task of classifying handwritten digits from the MNIST dataset [28] is much simpler than classifying an object into one of a 1000 classes as is required for the ImageNet dataset [18](Fig. 5). It is expected that the size of the classifier or network (i.e., number of weights) and the number of MACs will be larger for the more difficult task than the simpler task and thus require more energy. For instance, LeNet-5[24] is

TABLE I
SUMMARY OF POPULAR DNNS [20, 21, 24, 26, 27]. ACCURACY MEASURED BASED ON TOP-5 ERROR ON IMAGENET [18].

Metrics	LeNet 5	AlexNet	VGG 16	GoogLeNet (v1)	ResNet 50
Accuracy	n/a	16.4	7.4	6.7	5.3
CONV Layers	2	5	16	21	49
Weights	2.6k	2.3M	14.7M	6.0M	23.5M
MACs	283k	666M	15.3G	1.43G	3.86G
FC Layers	2	3	3	1	1
Weights	58k	58.6M	124M	1M	2M
MACs	58k	58.6M	124M	1M	2M
Total Weights	60k	61M	138M	7M	25.5M
Total MACs	341k	724M	15.5G	1.43G	3.9G

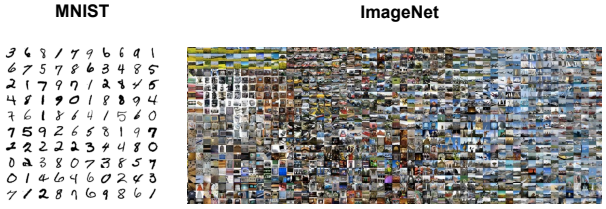


Fig. 5. MNIST (10 classes, 60k training, 10k testing) [28] vs. ImageNet (1000 classes, 1.3M training, 100k testing)[18] dataset.

designed for digit classification, while AlexNet[26], VGG-16[27], GoogLeNet[20], and ResNet[21] are designed for the 1000 class image classification.

IV. CHALLENGES

The key metrics for embedded machine learning are accuracy, energy consumption, throughput/latency, and cost.

The accuracy of the machine learning algorithm should be measured on a sufficiently large dataset. There are many widely-used publicly-available datasets that researcher can use (e.g., ImageNet).

Programmability is important since the weights need to be updated when the environment or application changes. In the case of DNNs, the processor must also be able to support different networks with varying number of layers, filters, channels and filter sizes.

The high dimensionality and need for programmability both result in an increase in computation and data movement. Higher dimensionality increases the amount of data generated and programmability means that the weights also need be read and stored. This poses a challenge for energy-efficiency since data movement costs more than computation [29]. In this paper, we will discuss various methods that reduce data movement to minimize energy consumption.

The throughput is dictated by the amount of computation, which also increases with the dimensionality of the data. In this paper, we will discuss various methods that the data can be transformed to reduce the number of required operations.

The cost is dictated by the amount of storage required on the chip. In this paper, we will discuss various methods to reduce storage costs such that the area of the chip is reduced, while maintaining low off-chip memory bandwidth.

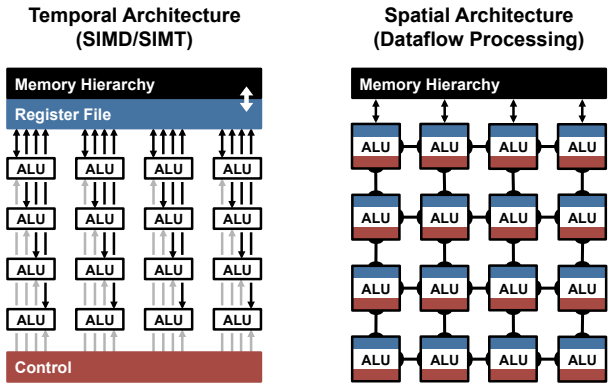


Fig. 6. Highly-parallel compute paradigms.

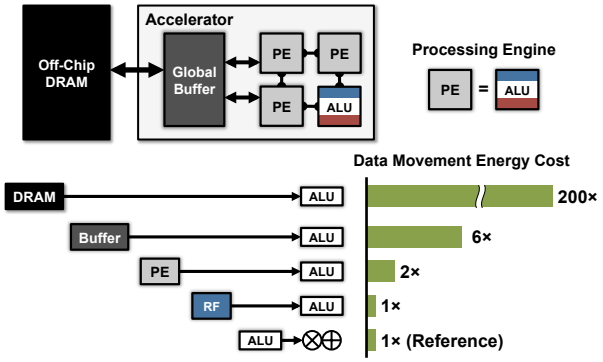


Fig. 7. Memory hierarchy and data movement energy [34].

Finally, training requires a significant amount of labeled data (particularly for DNNs) as well as computation for multiple iterations of back-propagation to determine the value of the weights. There is on-going research on training in the cloud using CPUs, GPUs, FPGAs and ASICs. However, this is beyond the scope of this paper.

Currently, state-of-the-art DNNs consume orders of magnitude higher energy than other forms of embedded processing (e.g., video compression). We must exploit opportunities at multiple levels of hardware design to address all these challenges and close this energy gap.

V. Opportunities in Architectures

The MAC operations in both the feature extraction (CONV layer in a DNN) and classification (for both DNN and hand-crafted features) can be easily parallelized. Two common highly-parallel compute paradigms are shown in Fig. 6 with multiple arithmetic logic units (ALU).

A. CPU and GPU Platforms

CPUs and GPUs use temporal architectures such as SIMD or SIMT to perform the MACs in parallel. All the ALUs share the same control and memory (register file). On these platforms, all classifications are represented by a matrix multiplication. The CONV layer in a DNN can also be mapped to a matrix multiplication using the Toeplitz matrix. There are software libraries designed for CPUs (e.g., OpenBLAS, Intel MKL, etc.) and GPUs (e.g., cuBLAS, cuDNN, etc.) that optimize for matrix multiplications. The matrix multiplication is tiled to the storage hierarchy of these platforms, which are on the order of a few megabytes at the higher levels.

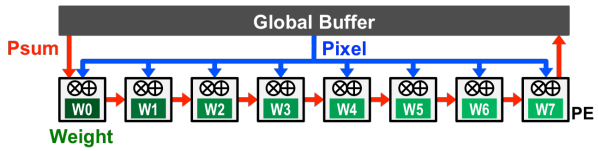
The matrix multiplications on these platforms can be further sped up by applying transforms to the data to reduce the number of multiplications. Fast Fourier Transform (FFT) [30, 31] is a well known approach that reduces the number of multiplications from $O(N_o^2 N_f^2)$ to $O(N_o^2 log_2 N_o)$, where the output size is $N_o \times N_o$ and the filter size is $N_f \times N_f$; however, the benefits of FFTs decrease with filter size. Other approaches include Strassen [32] and Winograd [33], which rearrange the computation such that the number of multiplications scale from $O(N^3)$ to $O(N^{2.807})$ and $2.25\times$ for a 3×3 filter, respectively, at the cost of reduced numerical stability, increased storage requirements, and specialized processing depending on the size of the filter.

B. Accelerators

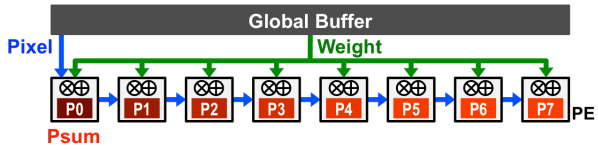
Accelerators provide an opportunity to optimize the data movement (i.e., dataflow) in order to minimize accesses from the expensive levels of the memory hierarchy as shown in Fig. 7. In particular, for DNNs we investigate dataflows that exploit three forms of data reuse (convolutional, filter and image). We use a spatial architecture (Fig. 6) with local memory (register file) at each ALU processing element (PE) on the order of 0.5 – 1.0kB and a shared memory (global buffer) on the order of 100 – 500kB. The global buffer communicates with the off-chip memory (e.g., DRAM). Data movement is allowed between the PEs using an on-chip network (NoC) to reduce accesses to the global buffer and the off-chip memory. Three types of data movement include input pixels, filter weights and partial sums (i.e., the product of pixels and weights) that are accumulated for the output.

Recent work [35–46] has proposed solutions for DNN acceleration, but it is difficult to compare their performance directly due to differences in implementation and design choices. The following taxonomy (Fig. 8) can be used to classify these existing DNN dataflows based on their data handling characteristics [34]:

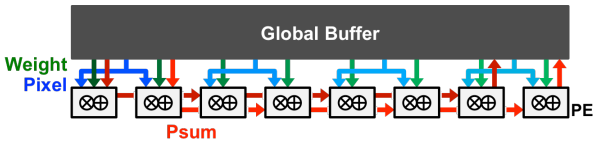
- **Weight stationary (WS):** The weights are stored in the register file at the PE and remains stationary to minimized the movement cost of the weights (Fig. 8(a)). The inputs and partial sums must move through the spatial array and global buffer. Examples are found in [35–40].
- **Output stationary (OS):** The outputs are stored in the register file at the PE and remains stationary to minimized the movement cost of the partial sums (Fig. 8(b)). The inputs and weights must move through the spatial array and global buffer. Examples are found in [41–43].
- **No local reuse (NLR):** While small register files are efficient in terms of energy (pJ/bit), they are inefficient in terms area (μm^2/bit). In order to maximize the storage capacity, and minimize the off-chip memory bandwidth, no local storage is allocated to the PE and instead all



(a) Weight Stationary



(b) Output Stationary



(c) No Local Reuse

Fig. 8. Dataflows for DNNs.

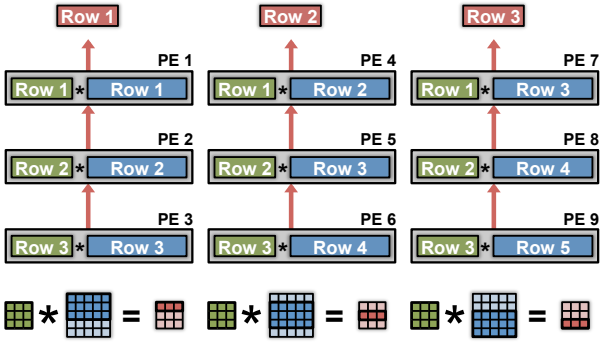
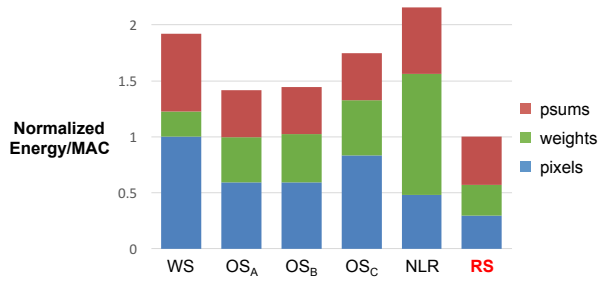


Fig. 9. Row Stationary Dataflow [34].

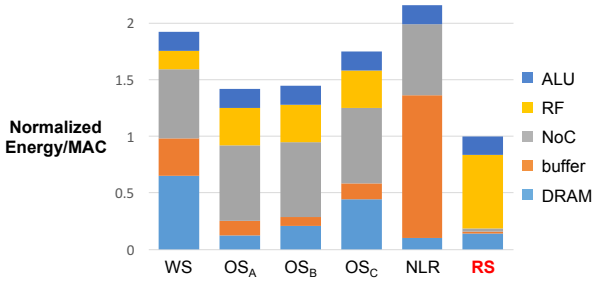
that area is allocated to the global buffer to increase its capacity (Fig. 8(c)). The trade-off is that there will be increased traffic on the spatial array and to the global buffer for all data types. Examples are found in [44–46].

- **Row stationary (RS):** In order to increase reuse of all types of data (weights, pixels, partial sums), a row stationary approach is proposed in [34]. A row of the filter convolution remains stationary within a PE to exploit 1-D convolutional reuse within the PE. Multiple 1-D rows are combined in the spatial array to exhaustively exploit all convolutional reuse (Fig. 9), which reduces accesses to the global buffer. Multiple 1-D rows from different channels and filters are mapped to each PE to reduce partial sum data movement and exploit filter reuse, respectively. Finally, multiple passes across the spatial array allow for additional image and filter reuse using the global buffer. This dataflow is demonstrated in [47].

The dataflows are compared on a spatial array with the same number of PEs (256), area cost and DNN (AlexNet). Fig. 10 shows the energy consumption of each approach. The row stationary approach is 1.4× to 2.5× more energy-efficient



(a) Across types of data



(b) Across levels of memory hierarchy

Fig. 10. Energy breakdown of dataflows [34].

than the other dataflows for the convolutional layers. This is due to the fact that the energy of all types of data is reduced. Furthermore, both the on-chip and off-chip energy is considered.

VI. Opportunities in Joint Algorithm and Hardware Design

There is on-going research on modifying the machine learning algorithms to make them more hardware-friendly while maintaining accuracy; specifically, the focus is on reducing computation, data movement and storage requirements.

A. Reduce Precision

The default size for programmable platforms such as CPUs and GPUs is often 32 or 64 bits with floating-point representation. While this remains the case for training, during inference, it is possible to use a fixed-point representation and substantially reduce the bitwidth for energy and area savings, and increase in throughput. Retraining is typically required to maintain accuracy when pushing the weights and features to lower bitwidth.

In hand-crafted approaches, the bitwidth can be drastically reduced to below 16-bits without impacting the accuracy. For instance, in object detection using HOG, each 36-dimension feature vector only requires 9-bit per dimension, and each weight of the SVM uses only 4-bits [48]; for object detection using deformable parts models (DPM) [49], only 11-bits are required per feature vector and only 5-bits are required per SVM weight [50].

Similarly for DNN inference, it is common to see accelerators support 16-bit fixed point [45, 47]. There has been significant

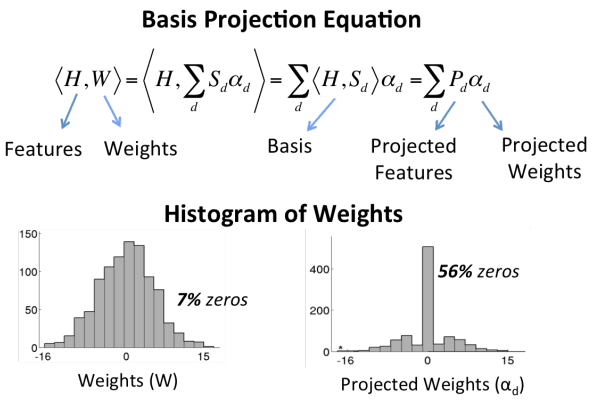


Fig. 11. Sparse weights after basis projection [50].

research on exploring the impact of bitwidth on accuracy [51]. In fact, recently commercial hardware for DNN reportedly support 8-bit integer operations [52]. As bitwidths can vary by layer, hardware optimizations have been explored to exploit the reduced bitwidth for 2.56× energy savings [53] or 2.24× increase in throughput [54] compared to a 16-bit fixed point implementation. With more significant changes to the network, it is possible to reduce bitwidth down to 1-bit for either weights [55] or both weights and activations [56, 57] at the cost of reduced accuracy. The impact of 1-bit weights on hardware is explored in [58].

B. Sparsity

For SVM classification, the weights can be projected onto a basis such that the resulting weights are sparse for a 2× reduction in number of multiplications [50] (Fig. 11). For feature extraction, the input image can be made sparse by pre-processing for a 24% reduction in power consumption [48].

For DNNs, the number of MACs and weights can be reduced by removing weights through a process called pruning. This was first explored in [59] where weights with minimal impact on the output were removed. In [60], pruning is applied to modern DNNs by removing small weights. However, removing weights does not necessarily lead to lower energy. Accordingly, in [61] weights are removed based on an energy-model to directly minimize energy consumption. The tool used for energy modeling can be found at [62].

Specialized hardware has been proposed in [47, 50, 63, 64] to exploit sparse weights for increased speed or reduced energy consumption. In Eyeriss [47], the processing elements are designed to skip reads and MACs when the inputs are zero, resulting in a 45% energy reduction. In [50], by using specialized hardware to avoid sparse weights, the energy and storage cost are reduced by 43% and 34%, respectively.

C. Compression

Data movement and storage are important factors in both energy and cost. Feature extraction can result in sparse data (e.g., gradient in HOG and ReLU in DNN) and the weights used in classification can also be made sparse by pruning. As a result, compression can be applied to exploit data statistics to reduce data movement and storage cost.

Various forms of lightweight compression have been explored to reduce data movement. Lossless compression can be used to reduce the transfer of data on and off chip [11, 53, 64]. Simple run-length coding of the activations in [65] provides up to 1.9× bandwidth reduction, which is within 5-10% of the theoretical entropy limit. Lossy compression such as vector quantization can also be used on feature vectors [50] and weights [8, 12, 66] such that they can be stored on-chip at low cost. Generally, the cost of the compression/decompression is on the order of a few thousand kgates with minimal energy overhead. In the lossy compression case, it is also important to evaluate the impact on performance accuracy.

VII. Opportunities in Mixed-Signal Circuits

Most of the data movement is in between the memory and processing element (PE), and also the sensor and PE. In this section, we discuss how this is addressed using mixed-signal circuit design. However, circuit non-idealities should also be factored into the algorithm design; these circuits can benefit from the reduced precision algorithms discussed in Section VI. In addition, since the training often occurs in the digital domain, the ADC and DAC conversion overhead should also be accounted for when evaluating the system.

While spatial architectures bring the memory closer to the computation (i.e., into the PE), there have also been efforts to integrate the computation into the memory itself. For instance, in [67] the classification is embedded in the SRAM. Specifically, the word line (WL) is driven by a 5-bit feature vector using a DAC, while the bit-cells store the binary weights ± 1. The bit-cell current is effectively a product of the value of the feature vector and the value of the weight stored in the bit-cell; the currents from the column are added together to discharge the bitline (BL or BLB). A comparator is then used to compare the resulting dot product to a threshold, specifically sign thresholding of the differential bitlines. Due to the variations in the bitcell, this is considered a weak classifier, and boosting is needed to combine the weak classifiers to form a strong classifier [68]. This approach gives 12× energy savings over reading the 1-bit weights from the SRAM.

Recent work has also explored the use of mixed-signal circuits to reduce the computation cost of the MAC. It was shown in [69] that performing the MAC using switched capacitors can be more energy-efficient than digital circuits despite ADC and DAC conversion overhead. Accordingly, the matrix multiplication can be integrated into the ADC as demonstrated in [70], where the most significant bits of the multiplications for Adaboost classification are performed using switched capacitors in an 8-bit successive approximation format. This is extended in [71] to not only perform multiplications, but also the accumulation in the analog domain. It is assumed that 3-bits and 6-bits are sufficient to represent the weights and input vectors, respectively. This enables the computation to move closer to the sensor and reduces the number of ADC conversions by 21×.

To further reduce the data movement from the sensor, [72] proposed performing the entire convolution layer (including convolution, max pooling and quantization) in the analog domain at the sensor. Similarly, in [73], the entire HOG feature is computed in the analog domain to reduce the sensor bandwidth by 96.5%.

VIII. Opportunities in Advanced Technologies

In the previous section, we discussed how data movement can be reduced by moving the processing near the memory or the sensor using mixed-signal circuits. In this section, we will discuss how this can be achieved with advanced technologies.

The use of advanced memory technologies such as embedded DRAM (eDRAM) and Hybrid Memory Cube (HMC) are explored in [46] and [74], respectively, to reduce the energy access cost of the weights in DNN. There has also been a lot of work that investigates integrating the multiplication directly into advanced *non-volatile* memories by using them as resistive elements. Specifically, the multiplications are performed where the conductance is the weight, the voltage is the input, and the current is the output (note: this is the ultimate form of weight stationary, as the weights are always held in place); the addition is done by summing the current using Kirchhoff's current law. In [75], memristors are used to compute a 16-bit dot product operation with 8 memristors each storing 2-bits; a 1-bit×2-bit multiplication is performed at each memristor, where a 16-bit input requires 16 cycles to complete. In [76], ReRAM is used to compute the product of a 3-bit input and 4-bit weight. Similar to the mixed-signal circuits, the precision is limited, and the ADC and DAC conversion overhead must be considered in the overall cost, especially when the weights are trained in the digital domain. The conversion overhead can be avoided by training directly in the analog domain as shown for the fabricated memristor array in [77].

Finally, it may be feasible to embed the computation into the sensor itself. This is useful for image processing where the bandwidth to read the data from the sensor accounts for a significant portion of the system energy consumption. For instance, an Angle Sensitive Pixels sensor can be used to compute the gradient of the input, which along with compression, reduces the sensor bandwidth by 10× [78]. A sensor that outputs gradients can also reduce the computation and energy consumption of subsequent processing engine [48, 79].

IX. Hand-crafted versus Learned Features

Hand-crafted approaches give higher energy efficiency at the cost of reduced accuracy as compared with learned features such as DNNs. For hand-crafted features, the amount of computation is less, and reduced bit-width is supported. Furthermore, less data movement is required since the weights are not required for the features. The classification weights for both approaches must however remain programmable. Fig. 12 compares the energy consumption of HOG feature extraction versus the convolution layers in AlexNet and VGG-16 based measured results from fabricated 65nm chips [50] and [47], respectively. Note that HOG feature extraction consumes around

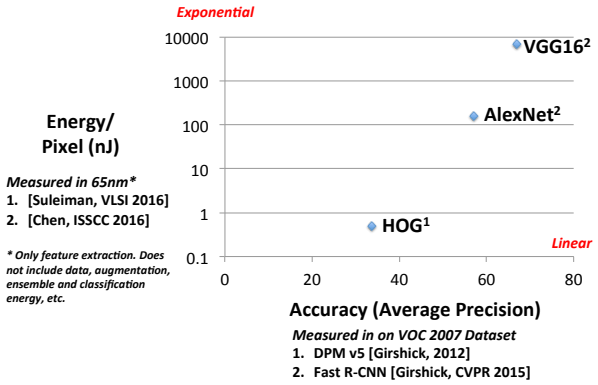


Fig. 12. Energy vs. accuracy comparison of hand-crafted and learned features.

the same energy as video compression (under 1nJ/pixel [80] for real-time high definition video), which servers as a good benchmark of what is acceptable for energy consumption near the sensor; however, DNNs currently consume several orders of magnitude more. A more detailed comparison can be found in [81]. We hope that the many design opportunities that we have highlighted in this paper will help close this gap.

X. Summary

Machine learning is an important area of research with many promising applications and opportunities for innovation at various levels of hardware design. During the design process, it is important to balance the accuracy, energy, throughput and cost requirements.

Since data movement dominates energy consumption, the primary focus of recent research has been to reduce the data movement while maintaining accuracy, throughput and cost. This means selecting architectures with favorable memory hierarchies like a spatial array, and developing dataflows that increase data reuse at the low-cost levels of the memory hierarchy. With joint design of algorithm and hardware, reduced bitwidth precision, increased sparsity and compression are used to minimize the data movement requirements. With mixed-signal circuit design and advanced technologies, computation is moved closer to the source by embedding computation near or within the sensor and the memories.

One should also consider the interactions between these different levels. For instance, reducing the bitwidth through hardware-friendly algorithm design enables reduced precision processing with mixed-signal circuits and non-volatile memory. Reducing the cost of memory access with advanced technologies could result in more energy-efficient dataflows.

Acknowledgment

Funding provided by DARPA YFA, MIT CICS, TSMC University Shuttle, and gifts from Texas Instruments and Intel.

References

[1] B. Marr, "Big Data: 20 Mind-Boggling Facts Everyone Must Read," Forbes.com, October 2015.
[2] "For a Trillion Sensor Road Map," TSensorSummit, October 2013.

[3] "Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015," Gartner.com, November 2015.

[4] "Cisco Global Cloud Index: Forecast and Methodology, 2015 - 2020," Cisco, June 2016.

[5] "Complete Visual Networking Index (VNI) Forecast," Cisco, June 2016.

[6] J. Woodhouse, "Big, big, big data: higher and higher resolution video surveillance," technology.ihs.com, January 2016.

[7] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[8] M. Price, J. Glass, and A. P. Chandrakasan, "A 6 mW, 5,000-Word Real-Time Speech Recognizer Using WFST Models," *IEEE J. Solid-State Circuits*, vol. 50, no. 1, pp. 102–112, 2015.

[9] R. Yazdani, A. Segura, J.-M. Arnau, and A. Gonzalez, "An ultra low-power hardware accelerator for automatic speech recognition," in *MICRO*, 2016.

[10] N. Verma, A. Shoeb, J. V. Guttag, and A. P. Chandrakasan, "A micro-power EEG acquisition SoC with integrated seizure detection processor for continuous patient monitoring," in *Sym. on VLSI*, 2009.

[11] T.-C. Chen, T.-H. Lee, Y.-H. Chen, T.-C. Ma, T.-D. Chuang, C.-J. Chou, C.-H. Yang, T.-H. Lin, and L.-G. Chen, "1.4μW/channel 16-channel EEG/ECoG processor for smart brain sensor SoC," in *Sym. on VLSI*, 2010.

[12] K. H. Lee and N. Verma, "A low-power processor with configurable embedded machine-learning accelerators for high-order and adaptive analysis of medical-sensor signals," *IEEE J. Solid-State Circuits*, vol. 48, no. 7, pp. 1625–1637, 2013.

[13] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *CVPR*, 2005.

[14] D. G. Lowe, "Object recognition from local scale-invariant features," in *ICCV*, 1999.

[15] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[16] R. E. Schapire and Y. Freund, *Boosting: Foundations and algorithms*. MIT press, 2012.

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[19] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *ISCAS*, 2010.

[20] C. Szegedy and et al., "Going Deeper With Convolutions," in *CVPR*, 2015.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[22] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.

[23] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *ICML*, 2010.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[25] Emer, Joel and Sze, Vivienne and Chen, Yu-Hsin, "Tutorial on Hardware Architectures for Deep Neural Networks," http://eyeriss.mit.edu/tutorial.html, 2016.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.

[27] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *ICLR*, 2015.

[28] C. J. B. Yann LeCun, Corinna Cortes, "THE MNIST DATABASE of handwritten digits," http://yann.lecun.com/exdb/mnist/.

[29] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *ISSCC*, 2014.

[30] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through FFTs," in *ICLR*, 2014.

[31] C. Dubout and F. Fleuret, "Exact acceleration of linear object detectors," in *ECCV*, 2012.

[32] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *ICANN*, 2014.

[33] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *CVPR*, 2016.

[34] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *ISCA*, 2016.

[35] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A Massively Parallel Coprocessor for Convolutional Neural Networks," in *ASAP*, 2009.

[36] V. Sriram, D. Cox, K. H. Tsoi, and W. Luk, "Towards an embedded biologically-inspired machine vision processor," in *FPT*, 2010.

[37] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A Dynamically Configurable Coprocessor for Convolutional Neural Networks," in *ISCA*, 2010.

[38] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," in *CVPRW*, 2014.

[39] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, "A 1.93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications," in *ISSCC*, 2015.

[40] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A Convolutional Network Accelerator," in *GLVLSI*, 2015.

[41] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," in *ICML*, 2015.

[42] Z. Du and et al., "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *ISCA*, 2015.

[43] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for Convolutional Neural Networks," in *ICCD*, 2013.

[44] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *FPGA*, 2015.

[45] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning," in *ASPLOS*, 2014.

[46] Y. Chen and et al., "DaDianNao: A Machine-Learning Supercomputer," in *MICRO*, 2014.

[47] Y.-H. Chen and et al., "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *ISSCC*, 2016.

[48] A. Suleiman and V. Sze, "Energy-efficient HOG-based object detection at 1080HD 60 fps with multi-scale support," in *SiPS*, 2014.

[49] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, 2010.

[50] A. Suleiman, Z. Zhang, and V. Sze, "A 58.6 mW real-time programmable object detector with multi-scale multi-object support using deformable parts model on 1920$\times$ 1080 video at 30fps," in *Sym. on VLSI*, 2016.

[51] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented Approximation of Convolutional Neural Networks," in *ICLR*,

2016.
[52] S. Higginbotham, "Google Takes Unconventional Route with Homegrown Machine Learning Chips," Next Platform, May 2016.
[53] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," in *Sym. on VLSI*, 2016.
[54] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," *IEEE Computer Architecture Letters*, 2016.
[55] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, 2015.
[56] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.
[57] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *ECCV*, 2016.
[58] R. Andri and et al., "YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights," in *ISVLSI*, 2016.
[59] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in *NIPS*, 1990.
[60] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both Weights and Connections for Efficient Neural Network," in *NIPS*, 2015.
[61] T.-J. Yang and et al., "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," *CVPR*, 2017.
[62] "DNN Energy Estimation," http://eyeriss.mit.edu/energy.html.
[63] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: ineffectual-neuron-free deep neural network computing," in *ISCA*, 2016.
[64] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *ISCA*, 2016.
[65] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE J. Solid-State Circuits*, vol. 51, no. 1, 2017.
[66] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," in *ICLR*, 2016.
[67] J. Zhang, Z. Wang, and N. Verma, "A machine-learning classifier implemented in a standard 6T SRAM array," in *Sym. on VLSI*, 2016.
[68] Z. Wang, R. Schapire, and N. Verma, "Error-adaptive classifier boosting (EACB): Exploiting data-driven training for highly fault-tolerant hardware," in *ICASSP*, 2014.
[69] B. Murmann, D. Bankman, E. Chai, D. Miyashita, and L. Yang, "Mixed-signal circuits for embedded machine-learning applications," in *Asilomar*, 2015.
[70] J. Zhang, Z. Wang, and N. Verma, "A matrix-multiplying ADC implementing a machine-learning classifier directly with data conversion," in *ISSCC*, 2015.
[71] E. H. Lee and S. S. Wong, "A 2.5 GHz 7.7 TOPS/W switched-capacitor matrix multiplier with co-designed local memory in 40nm," in *ISSCC*, 2016.
[72] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "RedEye: analog ConvNet image sensor architecture for continuous mobile vision," in *ISCA*, 2016.
[73] J. Choi, S. Park, J. Cho, and E. Yoon, "A 3.4-μW object-adaptive CMOS image sensor with embedded feature extraction algorithm for motion-triggered object-of-interest imaging," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, pp. 289–300, 2014.
[74] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in *ISCA*, 2016.
[75] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.
[76] P. Chi, S. Li, Z. Qi, P. Gu, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *ISCA*, 2016.
[77] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.
[78] A. Wang, S. Sivaramakrishnan, and A. Molnar, "A 180nm CMOS image sensor with on-chip optoelectronic image compression," in *CICC*, 2012.
[79] H. Chen, S. Jayasuriya, J. Yang, J. Stephen, S. Sivaramakrishnan, A. Veeraraghavan, and A. Molnar, "ASP Vision: Optically Computing the First Layer of Convolutional Neural Networks using Angle Sensitive Pixels," in *CVPR*, 2016.
[80] T.-J. Lin, C.-A. Chien, P.-Y. Chang, C.-W. Chen, P.-H. Wang, T.-Y. Shyu, C.-Y. Chou, S.-C. Luo, J.-I. Guo, T.-F. Chen *et al.*, "A 0.48 V 0.57 nJ/pixel video-recording SoC in 65nm CMOS," in *ISSCC*, 2013.
[81] A. Suleiman, Y.-H. Chen, J. Emer, and V. Sze, "Towards Closing the Energy Gap Between HOG and CNN Features for Embedded Vision," in *ISCAS*, 2017.