Towards Closing the Energy Gap Between HOG and CNN Features for Embedded Vision

(Invited Paper)

Amr Suleiman*, Yu-Hsin Chen*, Joel Emer, Vivienne Sze
Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology
{suleiman, yhchen, jsemer, sze}@mit.edu
**These authors contributed equally to this work*

***Abstract*—Computer vision enables a wide range of applications in robotics/drones, self-driving cars, smart Internet of Things, and portable/wearable electronics. For many of these applications, local embedded processing is preferred due to privacy and/or latency concerns. Accordingly, energy-efficient embedded vision hardware delivering real-time and robust performance is crucial. While deep learning is gaining popularity in several computer vision algorithms, a significant energy consumption difference exists compared to traditional hand-crafted approaches. In this paper, we provide an in-depth analysis of the computation, energy and accuracy trade-offs between learned features such as deep Convolutional Neural Networks (CNN) and hand-crafted features such as Histogram of Oriented Gradients (HOG). This analysis is supported by measurements from two chips that implement these algorithms. Our goal is to understand the source of the energy discrepancy between the two approaches and to provide insight about the potential areas where CNNs can be improved and eventually approach the energy-efficiency of HOG while maintaining its outstanding performance accuracy.**

I. Introduction

Computer vision (CV) is a critical technology to numerous smart embedded systems, such as advanced driver assistant systems, autonomous cars/drones, and robotics. It extracts meaningful information from visual data for further decision making. However, many modern CV algorithms require high computational complexity, which makes their deployment on battery-powered devices challenging due to the tight energy constraints. Near-sensor visual data processors should consume under 1nJ/pixel with a logic gate counts of around 1000 kgates and a memory capacity of few hundred kBytes in order to be comparable with video codecs, which are present in most cameras [1]. For many applications, offloading computation to the cloud is also undesirable because of latency, connectivity, and security limitations. Thus, dedicated energy-efficient CV hardware becomes very crucial.

Feature extraction is the first processing step in most CV tasks, such as object classification and detection (Fig. 1). It transforms the raw pixels into a high-dimensional space, where only meaningful and distinctive information is kept. Traditionally, features are designed by experts in the field through a hand-crafted process. For instance, many well-known *hand-crafted features* use image gradients, such as histogram of oriented gradients (HOG) [2] and scale invariant feature transform (SIFT) [3], based on the fact that human eyes are sensitive to edges. In contrast, *learned features* learn

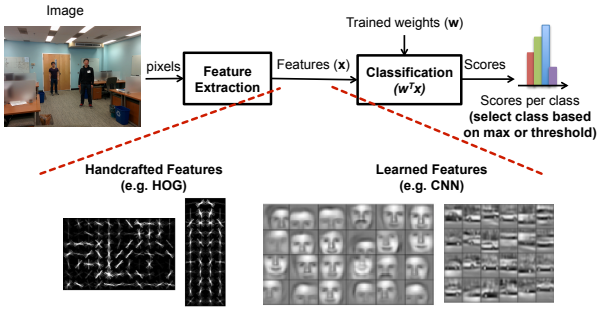

Fig. 1. General processing pipeline for object classification and detection.

a representation with the desired characteristics directly from data using deep convolutional neural networks (CNNs) [4]. Learned features are gaining popularity, as they are outperforming hand-crafted features in many CV tasks [5, 6].

The differences in design between hand-crafted and learned features lead to not only different performance in applications, but also different hardware implementation considerations, which have a strong implication for energy efficiency. In general, hardware implementations for hand-crafted features are widely understood to be more energy-efficient than learned features. However, there is no analysis that explains the energy gap between the two types of features. Also, an open question is whether the energy gap can be closed in the future.

In this paper, we will provide an in-depth analysis on the causes for the energy gap between hand-crafted and learned features. We use results from two actual chip designs: [7] implements the hand-crafted feature using HOG, and [8] implements the learned feature using CNN. Both chips use 65nm CMOS technology and have similar hardware resource utilization in terms of logic gate count and memory capacity. Based on the insights derived from the two implementations, we will discuss techniques to help close the energy gap.

II. Feature Extraction Hardware Implementations

A. Hardware for Hand-crafted Feature: HOG

The chip presented in [7] implements the entire object detection pipeline based on deformable parts models (DPM) [9] for high throughput and low power embedded vision applications. DPM extracts HOG features from the input image, and localizes objects by sweeping the features with support vector

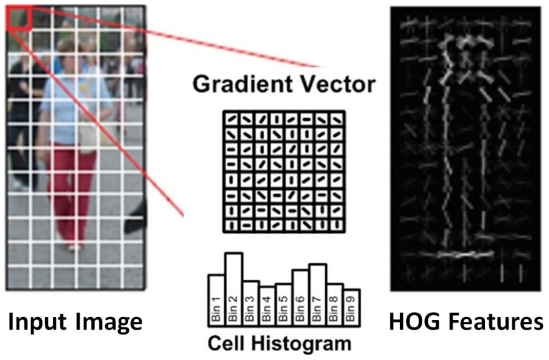

Fig. 2. Feature extraction with histogram of oriented gradients (HOG).

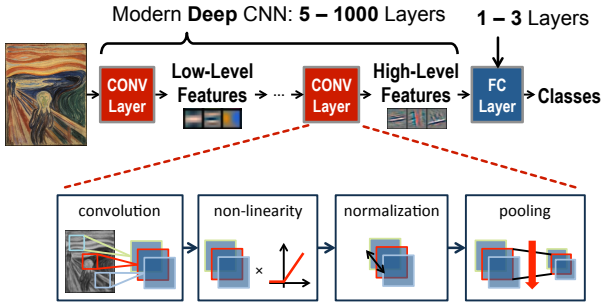

Fig. 3. General processing pipeline of CNN.

machine (SVM) classifiers [10]. Fig. 2 shows the features extraction process using HOG: the image is divided into non-overlapping 8×8 pixel cells. A 9-bin histogram is then generated for each cell using gradients orientations. The histograms are further normalized with respect to the surrounding cells for robustness against illumination and texture variations. Since HOG features are not scale invariant, the features are extracted over an image pyramid to achieve multi-scale detection.

B. Hardware for Learned Feature: CNN

The second chip presented in [8], called Eyeriss, is an energy-efficient accelerator for deep CNNs. Fig. 3 shows a general CNN processing pipeline, consisting mainly of a series of convolutional (CONV) layers. In each CONV layer, a collection of 3D filters are applied to the input images or feature maps to generate output feature maps, which are then used as the input to the next CONV layer. Eyeriss is programmable in terms of the size and shape of filters as well as number of layers. Therefore, it can accommodate different CNN architectures, such as AlexNet [5] and VGG-16 [11].

C. Performance Comparison

Table I shows the hardware specification and measured performance of the two designs for feature extraction. For CNN, Eyeriss is programmed to run two CNN models (five CONV layers of AlexNet and thirteen CONV layers of VGG-16) to demonstrate the hardware performance variations of running different CNN features. Both chip designs use around 1000 kgates and 150 kB memory. While Eyeriss achieves approximately the same computation throughput (i.e., GOPS)

TABLE I
HARDWARE SPECIFICATION AND MEASURED PERFORMANCE OF HAND-CRAFTED FEATURE HOG [7] AND LEARNED FEATURE CNN [8].

	[7]	[8]	
Implemented Feature	HOG	CNN (AlexNet)	CNN (VGG-16)
Technology	65nm	65nm	
Gate Counts (kgates)	893.0	1176.0	
Memory (kB)	159.0	181.5	
Multiplier Bitwidth	5×11 – 22×22	16×16	
Throughput (Mpixels/s)	62.5	1.8	0.04
Throughput (GOPS)	46.0	46.2	21.4
Power (mW)	29.3	278.0	236.0
DRAM access (B/pixel)	1.0	74.7	2128.6
Energy Efficiency (nJ/pixel)	0.5	155.5	6742.9
Energy Efficiency (GOPS/W)	1570.0	166.2	90.7

TABLE II
COMPUTATIONAL COMPLEXITY COMPARISON BETWEEN FEATURES

Feature		**GOP/Mpixel**	**Ratio**
Hand-crafted	HOG	0.7	1.0×
Learned	CNN (AlexNet)	25.8	36.9×
	CNN (VGG-16)	610.3	871.9×

as the HOG design when running AlexNet features, HOG processes 35× more input pixels per second; the gap is even larger between HOG and VGG-16 features. This is due to the differences in computational complexity (i.e., operations per pixel) between different features as shown in Table II. However, the HOG design also consumes around 10× less power than Eyeriss. Thus, the HOG hardware consumes 311× and 13,486× less energy per pixel than Eyeriss running AlexNet and VGG-16 features, respectively. In terms of energy per operation, the HOG hardware is 10× less than Eyeriss. In Section III, we will discuss the cause of this energy gap.

D. Accuracy vs. Energy Efficiency

While there is a large energy gap between the hardware for hand-crafted and learned features, the performance differences in applications, e.g., accuracy, between the features should also be taken into account. To measure accuracy, we use the features for object detection, which localizes and classifies objects in provided images (i.e., the outputs are the class and coordinates of each object in the images). The mean average precision (mAP) metric is used to quantify detection accuracy.

Fig. 4 shows the trade-off between detection accuracy and energy consumption. Note that the vertical axis is logarithmic. All reported detection accuracy numbers are measured on PASCAL VOC 2007 dataset [12], which is a widely used image dataset containing 20 different object classes in 9,963 images. In order to achieve the same detection accuracy of HOG features, it only requires the features extracted from the first three CONV layers of AlexNet instead of five [13], but it comes at the cost of 100× higher energy consumption per pixel to generate the features. Fortunately, mAP can be nearly doubled by using the ouput of all five layers of AlexNet with minimal increase in energy consumption (only 22%). Even higher mAP has been demonstrated by using VGG-16 features [14]. However, the accompanying energy consumption

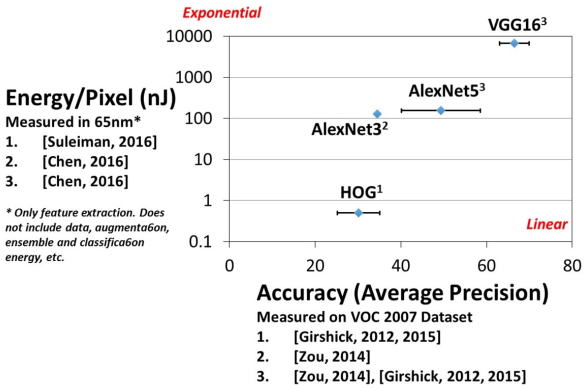

Fig. 4. Energy versus accuracy trade-off comparison for hand-crafted and learned features.

per pixel becomes four orders of magnitude higher than using HOG features. If we look at the minimum hardware energy consumption required for a given accuracy in Fig. 4, we roughly see that a linear increase in detection accuracy requires an exponential increase in energy consumption per pixel.

III. Hardware Implementation Considerations

One of the key factors that differentiates the two types of features in hardware implementation is *programmability*, which defines the flexibility of a feature to be configured to deal with different data statistics or different tasks. While hand-crafted features can achieve a certain degree of invariance to data variations (e.g., images with different exposures), they are mostly designed for very specific tasks with known data statistics, leaving little to be programmed when deployed. In contrast, learned features isolate algorithm design from learning the data statistics, which leaves room for programmability to take advantage of the flexibility. We categorize programmability into two types: Programmability of Hyper-Parameters (PoHP) and Programmability of Parameters (PoP).

A. Programmability of Hyper-Parameters (PoHP)

Hyperparameters refer to the number and/or dimensionality of parameters of a feature such as the number of layers and size of filters in a CNN, or the number of histogram bins in HOG. They are usually determined at design time by using heuristics or through experiments.

Advantages: For learned features, changes in hyperparameters with proper (re-)training can result in significant performance improvements. As a result, hardware that supports PoHP can easily trade-off computational complexity for higher accuracy. For instance, a CNN object detection system with PoHP can choose between lower-complexity, lower-accuracy, e.g., AlexNet, and higher-complexity, higher-accuracy, e.g., VGG-16, according to the use cases. For hand-crafted features, however, the impact of PoHP on performance is usually limited, since it can be hard to capture the changes in parameter dimensionality without redesign. As a result, PoHP are not commonly supported for the hardware implementation of hand-crafted features.

Energy Cost: Although PoHP can greatly benefits the performance of learned features, it comes at the price of lowered energy efficiency, since the hardware implementation must accommodates a wide range of possible configurations. First, it introduces *irregular data movement*. For example, a CNN processor that supports PoHP has to deal with filters of many shapes and sizes, which complicates the control logic, synchronization schemes between parallel blocks, and data tiling optimization. Second, it usually requires *higher dynamic range and resolution for data representation*, which have a negative impact on both computation and data movement. The bitwidth of datapath and the bandwidth requirement of memory all need to be designed for the worst-case scenario, which penalize the average energy efficiency.

B. Programmability of Parameters (PoP)

Parameters are the actual coefficients, such as the filter weights, that are used in the computation. For learned features, parameters are learned through training; for hand-crafted features, they are usually carefully designed according to the data statistics or pre-determined based on the desired properties. For instance, in HOG, the filter mask for the gradient extraction is simply $[-1\ 0\ 1]$.

Advantages: With PoP, learned features can adapt to new data by simply retraining the parameters. For example, AlexNet has been shown to work for both RGB images and depth images [15]. For hand-crafted features, however, there is no PoP since all parameters are fixed at design time.

Energy Cost: PoP also negatively impacts the hardware energy efficiency since the parameters have to be treated as data instead of fixed values during hardware implementation. This not only increases the required memory capacity and data movement, but also complicates the datapath design. In the case of CNN, the amount of parameters is usually too large to fit on-chip, which increases the accesses to energy-consuming off-chip memory, such as DRAM. In contrast, hand-crafted features can be greatly optimized for energy efficiency by hard-wiring the fixed parameters in datapaths or ROM. In the case of HOG, multiplications between the input images and gradient filter can be completely avoided.

IV. Closing the Energy Gap

In Section II, we have shown the energy efficiency results of hardware implementations from two extremes: the HOG hand-crafted feature with no PoP or PoHP, and the CNN learned feature with both PoP and PoHP, and the latter consumes an order of magnitude higher normalized energy per pixel than the former. A simple approach to reduce this energy gap is to remove all programmability in the hardware implementation of CNN, but this is not straightforward.

For example, the CONV layer weights from AlexNet can either be hard-wired in the multipliers, or stored in on-chip SRAM or ROM. This is not feasible if the available hardware resources are constrained to the level of the HOG design [7], i.e., 1000 kgates with 150 kB SRAM. Assuming each input and weight value take 1 byte, only 10k multipliers with fixed

TABLE III
CNN ENERGY AND MEMORY SAVINGS USING DIFFERENT TECHNIQUES. *MEASURED ON ALEXNET **ASSUMING A 16-BIT BASELINE

Method	Energy	Memory Size
Reduced precision [17, 22]	2.56×	2.0×
Sparsity by pruning [25]	3.7×	6.6×*
Data Compression [22]	-	>2.0×**
Energy optimized dataflow [27]	1.4–2.5×	-

weights can be implemented in 1000 kgates, and only 150k weight values can fit in the SRAM. This number of multipliers cannot even fit 1% of AlexNet 2334k weights in the CONV layers, and 15× larger memory is required to store all weights.

In this section, we will discuss some techniques that can be applied to reduce the energy efficiency gap.

Reduced Precision: Reducing data precision is an active research area for DNN [16], which directly reduces the worst-case bitwidth requirement when supporting PoHP. Specifically, 8-bit integer precision has become popular in recent DNN hardware [17]. Non-uniform quantization [18] and bitwidth reduction to 1-bit [19–21] have also been demonstrated. Energy efficiency can also be improved if the hardware can adapt to the need of actual data. Custom datapath designs that adapt to the lower data precision show 2.56× energy savings in [22].

Sparsity: Increasing data sparsity reduces the intrinsic amount of computation and data movement, which improve energy efficiency. For CNN, the number of weights can be greatly reduced without reducing accuracy through pruning [23–25]. Specifically, [25] has shown that the number of CONV layer weights in AlexNet can be reduced from 2334k to 352k, which is only double the memory capacity used in HOG; furthermore it reduces energy by 3.7×. Specialized hardware designs can be used to exploit sparsity for increased speed or reduced energy consumption [8, 18, 26].

Data Compression: Sparsity also suggests opportunities for compression, which saves memory space and data transfer bandwidth. Many lightweight lossless compression schemes, such as run-length coding [8] and Huffman coding [18], are proposed to reduce the amount of off-chip data [8, 18, 22].

Energy Optimized Dataflow: PoHP incurs irregular data movement, preventing memory access optimization during hardware design. Therefore, designing hardware architecture that can adapt to the irregular data movement becomes critical to high energy efficiency, since data movement often consumes more energy than computation. Eyeriss demonstrates a reconfigurable architecture that can optimize data movement for various CNNs with a row stationary dataflow, and achieves 1.4× to 2.5× higher energy efficiency than existing designs [27].

Table III summarizes the energy and memory savings using the discussed techniques. Combining them all can potentially deliver an order of magnitude reduction. Taking into account the fundamental computation gap discussed in Section II, this reduction has the potential of closing the gap between HOG and CNN features.

V. CONCLUSION

The CNN learned features outperform the HOG hand-crafted features in visual object classification and detection tasks. This paper compares two chip designs for CNN and HOG to better understand the energy discrepancy between the two approaches and provide insight about the potential optimizations. Although learned features achieve more than 2× accuracy, it comes at a large 311× to 13,486× overhead in energy consumption. While a fundamental computation overhead exists, another order of magnitude gap is mainly caused by the fact that CNN architecture is programmable. A simple approach of removing all programmability in CNN and hard-wiring all multiplications doesn't work due to significant area cost (i.e., logic gates and on-chip memory). Combing the techniques highlighted in the paper can potentially reduce the energy and memory sizes by an order of magnitude, and help reduce the gap between learned and hand-crafted features.

REFERENCES

[1] C. C. Ju and et al., "A 0.5nJ/pixel 4K H.265/HEVC codec LSI for multi-format smartphone applications," in *ISSCC*, pp. 1–3, Feb 2015.
[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *CVPR*, 2005.
[3] D. G. Lowe, "Object recognition from local scale-invariant features," in *ICCV*, 1999.
[4] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *ISCAS*, 2010.
[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NIPS*, 2012.
[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *CVPR*, 2014.
[7] 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× 1080 video at 30fps," in *Sym. on VLSI*, 2016.
[8] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *ISSCC*, 2016.
[9] R. B. Girshick, P. F. Felzenszwalb, and D. McAllester, "Discriminatively trained deformable part models, release 5," 2012.
[10] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, 2000.
[11] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *ICLR*, 2015.
[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results." http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.
[13] W. Zou, X. Wang, M. Sun, and Y. Lin, "Generic Object Detection with Dense Neural Patterns and Regionlets," in *BMVC*, 2014.
[14] R. Girshick, "Fast R-CNN," in *ICCV*, 2015.
[15] S. Gupta, R. Girshick, P. Arbelaez, and J. Malik, "Learning rich features from RGB-D images for object detection and segmentation," in *ECCV*, 2014.
[16] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented Approximation of Convolutional Neural Networks," in *ICLR*, 2016.
[17] S. Higginbotham, "Google Takes Unconventional Route with Homegrown Machine Learning Chips," May 2016.
[18] 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.
[19] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, 2015.
[20] 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.
[21] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *ECCV*, 2016.
[22] 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.
[23] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in *NIPS*, 1990.
[24] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both Weights and Connections for Efficient Neural Network," in *NIPS*, 2015.
[25] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," *CVPR*, 2017.
[26] 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.
[27] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *ISCA*, 2016.