Brain-inspired Classroom Occupancy Monitoring on a Low-Power Mobile Platform

Francesco Conti*, Antonio Pullini† and Luca Benini*†
*Department of Electrical, Electronic and Information Engineering, University of Bologna, Italy
†Integrated Systems Laboratory, ETH Zurich, Switzerland
f.conti@unibo.it,{pullinia,lbenini}@iis.ee.ethz.ch

***Abstract*— Brain-inspired computer vision (BICV) has evolved rapidly in recent years and it is now competitive with traditional CV approaches. However, most of BICV algorithms have been developed on high power-and-performance platforms (e.g. workstations) or special purpose hardware. We propose two different algorithms for counting people in a classroom, both based on Convolutional Neural Networks (CNNs), a state-of-art deep learning model that is inspired on the structure of the human visual cortex. Furthermore, we provide a standalone parallel C library that implements CNNs and use it to deploy our algorithms on the embedded mobile ARM big.LITTLE-based Odroid-XU platform. Our performance and power measurements show that neuromorphic vision is feasible on off-the-shelf embedded mobile platforms, and we show that it can reach very good energy efficiency for non-time-critical tasks such as people counting.**

I. Introduction

Brain-inspired computer vision (BICV) has attracted much attention lately as a possible means to overcome the limits of current algorithmic solutions by taking inspiration from a proven, energy-efficient vision system: the mammalian brain [4]. *Convolutional neural networks* are designed to imitate the deep, layered structure of the visual cortex and its capability to link low-level features and derive higher level concepts in each new layer. In this way, they can often achieve better efficiency and generality than traditional computer vision techniques, which usually rely on strictly separated phases of extraction and classification of image features.

Potential applications for BICV are usually constrained by cost and power consumption. Fortunately, modern system-on-chips for the embedded and mobile markets are able to provide an amount of performance rivaling that of desktop computers of the previous generation, and even this gap is closing quite rapidly: deploying heavy neuromorphic algorithms on cheap embedded chips is now feasible, at least when energy - as opposed to time - is the main constraint.

In this work, we concentrate on BICV algorithms for monitoring classroom occupancy to drive the HVAC system of our faculty building. Counting people is a task that humans can perform reasonably well (even if with a significant error rate). For a CV system, though, it raises a number of challenges: head orientation, distance from camera, color, expression and hair can differ and the room itself can vary wildly in contrast, luminosity and hue depending on external factors. An algorithm that is used to count people must be robust with respect to all of these sources of noise.

We propose and compare two distinct approaches to occupancy monitoring, both relying on convolutional neural networks (CNNs). The main contributions of this work are: (i) the development of the algorithms (along with the dataset used to train the CNNs); (ii) their embedding and acceleration on Odroid-XU, a low-power embedded board based on a state-of-art mobile SoC platform; (iii) the analysis of their accuracy, performance, and energy footprint.

The rest of the paper is organized as follows: Section II describes related works. In Section III we discuss the mathematical model we used for CNNs and our software implementation. In Section IV we describe the two algorithms we developed for classroom occupancy estimation. Finally, in section V we show accuracy, performance and energy results for both algorithms.

II. Related works

Convolutional neural networks, proposed by Lecun et al. [18], are a deep learning model that is rooted in Hubel and Wiesel's work [16] on the cortex of a cat. Their model of the visual cortex is composed of *simple cells* and *complex cells*, correspondent to convolutional and pooling layers respectively. Other important models inspired on the same findings are the Neocognitron [12] and the HMAX model [25], which differ from CNNs in that the weights they use are not learned through supervised backpropagation, but layer by layer (for the Neocognitron) or by a combination of unsupervised learning and hardwired weights.

Both convolutional neural networks and HMAX have been used to solve a variety of real-world computer vision tasks such as image classification [17][19][21] and scene labeling [9][10]. Convolutional networks are also being used by Google [1] and Microsoft [8] to power various tasks such as visual search algorithms and speech recognition.

Recently, much research has gone into implementing neuromorphic algorithms on custom embedded platforms. For example, Farabet et al. [11] introduce NeuFlow, a FPGA-based dataflow processor designed explicitly to maximize performance with convolutional neural network algorithms. The authors of [2] also propose a multi-FPGA system to efficiently implement neuromorphic algorithms based on the HMAX model. Targeting FPGAs requires essentially designing custom

HW to deploy on expensive chips; by contrast, we focus on mobile chips that are ∼10x less expensive and programmable in SW by a much larger community.

There are other models, such as those based on the more physiologically faithful "spiking" neurons, that are also being studied. Important academic research lines focusing on computation (as opposed to the simulation of the brain) include liquid state machines [20] and reservoir computing [23][22]. In these models, a random recurrent network of spiking neurons is excited by a stimulus and output is read out from the internal state of the network through a learned classifier. Although interesting, these approaches are less mature and are not yet competitive with traditional CV.

There has also been some research around traditional CV techniques for people counting. Andriluka et al. [3] and Breitenstein et al. [7] propose various techniques for pedestrian detection and tracking, while Hou et al. [14] propose a technique for occupation estimation in a crowded outdoor area based on expectation maximization. All these approaches are ad-hoc and hence not easily retargetable, while BICV algorithms can be readily adapted to new tasks.

III. Convolutional network model

Convolutional neural networks [18] (CNNs) are a well known deep learning model inspired to the structure of the human visual cortex. The model we use is mainly inspired to recent advancements in feature extraction from complex scenes [10]. In a CNN, each layer operates on an array of *feature maps*, i.e. 2D images that embed some feature of the input of the network. Each layer takes all or some of the feature maps of the previous layer and produces a set of feature maps that feed the next layer. Two main kinds of layers are present: *convolutional* layers and *pooling* layers.

Convolutional layers implement the localized receptive field of the human cortex: each output feature map is produced by filtering a set of input feature maps with two-dimensional convolution kernels (or *weights*). Pixel $y_n(i,j)$ of the n-th output feature map is given by

$$y_n(i,j) = \tanh\left(b_n + \sum_{m \in \mathrm{CF}_n} \Big(W_{n,m} * x_m\Big)(i,j)\right) \qquad (1)$$

where $*$ is the 2D convolution, b_n is the bias, $W_{n,m}$ is the weight, x_m is the m-th feature map and CF_n is the set of connected feature maps of the n-th output feature map.

Pooling layers take input feature maps and reduce their size by mechanisms such as average-pooling or max-pooling: output pixels are computed as (respectively) the average or the maximum over a pool of input pixels, such as a 2x2 square. Using the same representation as in equation (1), the max-pooling layer performs

$$y_n(i,j) = \max\left\{x_n(i,j); \cdots; x_n(i+P, j+P)\right\} \qquad (2)$$

where $P+1$ is the pooling factor.

This CNN model can be extended by replicating the action of all convolutional and pooling layers on multiple *scales* that share the same weights to impose scale-invariance. To merge different scales into a single feature map, we use an *average-merge* layer that rescales all of the feature maps to a common scales and computes their average.

Weights and biases in CNNs can be trained through supervised backpropagation. For the purpose of this work, we used two tools for network learning: the open source EBLearn [24] tool and a simpler tool (*PyConvNet*) we developed in Python, based on the open source Theano [5] library. Both EBLearn and PyConvNet make use of multi-core and GPU computing to speed up the learning phase, which is based on the stochastic gradient descent algorithm [6].

To deploy the trained convolutional neural networks we developed a portable stand-alone software library written in C99, *CConvNet*, targeted for both x86 and embedded (ARM) targets. To accelerate CNNs, we take advantage of their data parallelism by using the OpenMP programming model; we also use the NEON extensions for vectorization on ARM targets. The CConvNet library is open-source and available from our website: http://www-micrel.deis.unibo.it/~conti/cconvnet.

IV. Occupancy estimation algorithms

A. People counting by head detection

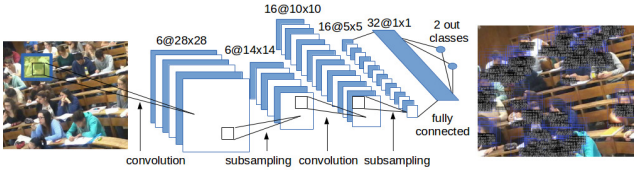


Figure 1: Head detection CNN.

The first approach we followed can be classified as a template matching as it verifies the presence or absence of a face at each possible location in the image by sliding the detection window. The architecture of the detector shown in Figure 1 is based on the LeNet-5 proposed by Le Cun at al. [18] where there are only 2 output neurons and a less deep set of fully connected layers. The overall approach is similar to what Garcia et al. proposed in [13].

The input image shot by the camera is converted to a luminance map and prescaled at 3 different resolutions (1152x864, 960x720, 800x600). The choice of the resolutions is dependent on the position of the camera in the room and was tuned to minimize the number of detections needed to cover the whole area and depth. The sliding window of the detector is shifted by half the detector window width and spans the whole set of scaled images.

The input of the detector is a 32x32 crop of the scaled image, while the outputs of the network are a boolean value indicating the presence of a face and the confidence of the decision, both derived from the values (y_0, y_1) of the output neurons. During training, we used a ground-truth output of $(-1, 1)$ and $(1, -1)$ for the output neurons to indicate faces and backgrounds, respectively. The final classification is the

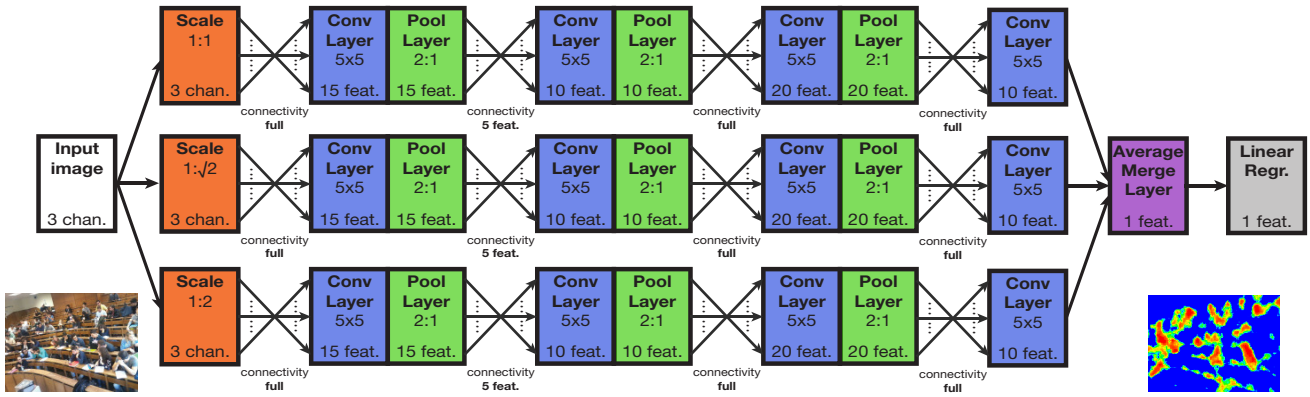


Figure 2: People density estimation CNN.

following: an image is a face if $y_1 - y_0 < 0$; the confidence of the decision is given by $\frac{1}{2}|y_1 - y_0|$.

The network was trained using the EBLearn [24] software. The full dataset is based on the LFW [15] dataset with the addition of 3000 faces cropped from pictures taken by the cameras installed in the classrooms. The additional images are needed to increase the robustness of the network to partially covered and non-frontal faces. To train the network to recognize the background we created our own dataset, consisting in 17000 32x32 crops from 165 images of empty rooms taken with different light conditions; in order to reduce training time we used only a subset of 5000 randomly selected.

In order to improve accuracy, considerable training time was spent in bootstrapping loops to decrease the rate of false positives on the background. The bootstrapping loop consisted of running the detection again on the full images previously used to create the background dataset. The detection was done considering a very low confidence threshold of 0.01 to identify false positives. Each false positive was cropped from the image and added to the background dataset. The bootstrapping step stopped when 400 false positives were detected or all the images in the set had been checked. After extending the background dataset with all false positives, the procedure was concluded by running again the training. If necessary, the whole bootstrapping loop was repeated: in our case, we stopped after 3 iterations when the total number of false positive over all empty images decreased below 400.

The training time for a single bootstrap run was of 2.5 hours on a i7 workstation running at 2.7 Ghz. The total training time including the first training and the three following bootstraping loops was 10 hours; the final accuracy we achieved was 97.6% on 500 randomly selected images of the validation dataset.

Since the detector window slides by half the detector window width at a time, a single face often results in multiple hits. After processing all the detection at the different scales, face candidate boxes are thus mapped back to the input scale and the bounding boxes are grouped according to the proximity of their centers, along with their confidence value. If the centers of two bounding boxes are less than 30 pixels aside, we retain only the smallest one, with its confidence value set to the sum of the confidence of each box. Applying a threshold of 3 on the confidence value of the resulting boxes filters out most of the false positives detected on body parts other than heads.

B. *People counting by density estimation*

The second approach we followed to estimate the number of people in a classroom is based on estimation of *people density*, and is inspired by the full-scene parsing algorithm proposed in [10]. Rather than identifying heads in the image and then counting them, we trained a convolutional neural network to classify each pixel as part of a human or not. The output of the network was then interpreted as the people density in the surroundings of a pixel, and thus can be used to estimate the number of people in the image.

The whole image is parsed by a single big convolutional neural network, which is shown in figure 2. The image is processed in full RGB color at three scales: 1 (1600x1200), $1/\sqrt{2}$ (1131x848) and $1/2$ (800x600); feature maps from the various scales are finally unified by an average-merge layer.

The output of the average-merge layer of the CNN is a 192x142 feature map that can be interpreted as the probability density of human presence in a given region. The value of each pixel ranges from -1.0 to $+1.0$. Figure 2 shows an example density map; "cool" regions indicate a low probability or density of human presence, while "hot" ones indicate a high probability. In the example all pixels not reaching a threshold value of 0.0 have been set to -1.0 (i.e., null people density).

We trained this network on PyConvNet exploiting Theano's ability to offload most of the computation on a GPU. For training, we took advantage of the CNN scale-invariance property by using 1000 smaller 400x300 images taken by the camera built by cropping full-size images in a random position. This shortcut allows us to: 1) train the network at higher speed (about 80 minutes on a Nvidia Tesla GPU), and 2) maximize the randomness of the image background, that could confuse the training procedure. During learning, the output of the network for each training image was compared with a ground-truth output, where the pixels in head regions had been set to $+1.0$ and all other regions to -1.0; we used the well known negative log-likelihood cost metric to drive the gradient descent backpropagation training.

An output linear stage multiplies the 192x142 output map of

the CNN with a weight matrix to obtain the overall estimation of classroom occupancy (i.e., a single number). The weight matrix was obtained by linear regression over a set of 100 images where people had been manually counted. Contrary to the CNN itself, the output weight matrix is heavily dependent on proportions and prospective and specific to a single room.

V. Results

A. Accuracy

To test the accuracy of head detection (HD) and density estimation (DE) we ran them on a large set of images taken from three classrooms in our faculty; the camera took three pictures one minute apart every ten minutes in the 8am-8pm time span (216 images/day). We manually counted the number of people present in two days in two of these classrooms to form a ground-truth dataset: Figure 3 presents the results of the HD and DE algorithms compared with it.

Algorithm	Room 1	Room 2
Head detection	6.46	8.55
Density estimation	8.35	8.44

Table I: People counting RMS error

Table I shows root-mean-square error[1] for both algorithms in both rooms. Error sources differ depending on the algorithm. DE can fail if i) the CNN does not correctly detect head features, or ii) the output stage does not weigh pixels correctly. The first case happens mainly in low-light conditions, where some pixels are wrongly classified as high-density. The second kind of error is due to transient conditions, such as a head occupying a higher than usual area in the camera view. For HD, error rate is highly dependent on the rate of false positives, though in some conditions (particularly low-light) some heads can be missed. The overall better performance of HD can be attributed to the bootstrapping effort to eliminate false positives; the same procedure could not be replicated for DE, as the CNN has a full image as output instead of a single boolean value.

[1] $E_{\text{RMS}} = \sqrt{\frac{1}{N_{\text{images}}} \sum (y_{\text{predicted}} - y_{\text{real}})^2}$

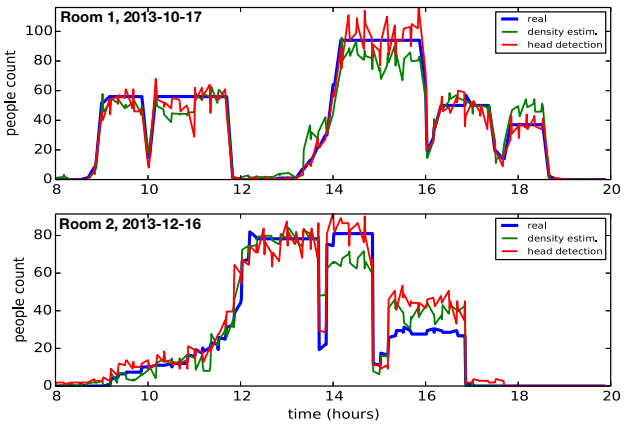


Figure 3: Example of occupancy estimation.

For some applications (for example lighting control), it is critical to distinguish between empty and non-empty rooms. In this case HD performs generally better than DE: in the result set from the first room 12 images are inaccurately identified as non-empty by HD and 22 by DE over a total of 51 empty images. Both algorithms inaccurately identify as empty 8 images over the remaining 165.

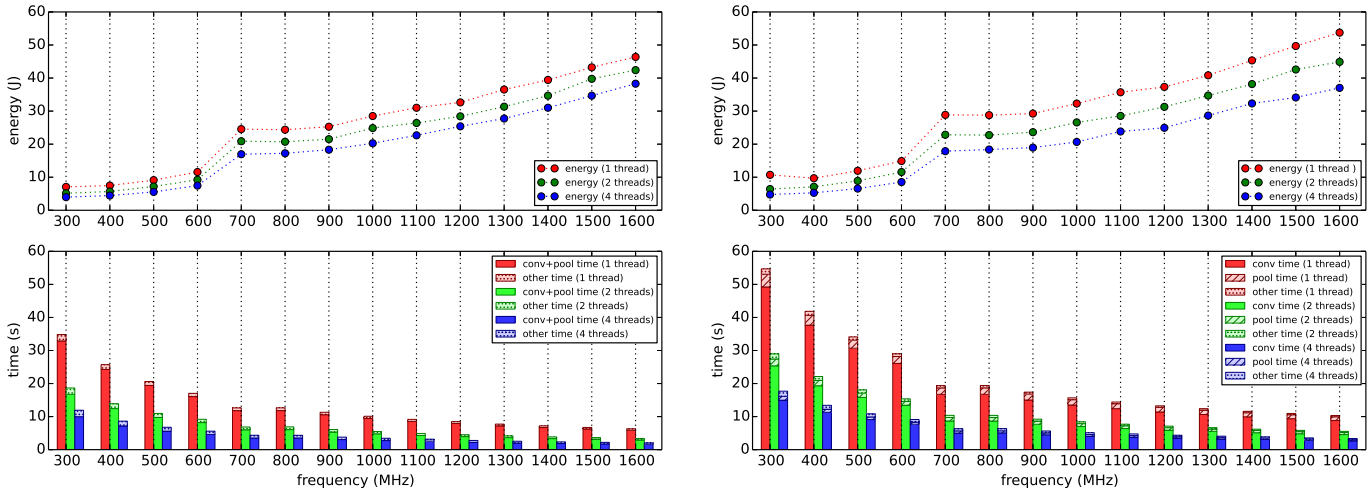
Result fluctuation could be attenuated by correlating results in time domain instead of considering each image apart; this could considerably improve accuracy, though it would also impose new constraints on performance (i.e. minimum frame rate).

B. Performance on Odroid-XU

To evaluate the performance of the algorithms on an embedded platform, we used an Odroid-XU board featuring a Samsung Exynos 5410 SoC. The Exynos 5410 contains a cluster of four low-power ARM A7 cores and one of four high-performance A15 cores, organized in big.LITTLE configuration (i.e. working alternatively). The Odroid-XU is able to run both Linux and Android; we ran our tests on Android to ease future integration with the camera application generating images, which is an Android app.

We implemented our algorithms using the CConvNet library, compiled with GCC 4.6 from the Android NDK, with the following flags for automatic NEON vectorization: `-ftree-vectorize -mfpu=neon -mfloat-abi=softfp -march=armv7-a`. CConvNet can use OpenMP to parallelize the inner loops of Equations 1 and 2; alternatively, coarse-grained parallelism can be extracted by applying multiple CNNs over an image simultaneously. In the first case, when a convolutional layer is executed a pool of threads is created using the `#pragma omp parallel` clause. The threads compute multiple rows of the same output feature map in parallel, without waiting one another (we use a `#pragma omp for nowait` clause). An implicit barrier is placed at the end of the convolution operation. Pooling layers are parallelized by computing each output feature map in a different thread. In the second case, we disable internal parallelization and replicate the CNN data structures to allow for multiple instances of the CNN to work simultaneously; we use OpenMP to parallelize the outer loops that span the input image. From the functional point of view, both algorithms show virtually identical results on x86 and on the Odroid-XU board.

Figure 4 shows execution time and energy consumption for the execution of both algorithms on a single image, while scaling the number of OpenMP threads from 1 to 4 with the `OMP_NUM_THREADS` environment variable. Execution time is divided in time spent in convolutional layers, in pooling layers and in other parts of the program (e.g. image loading and output stages). Head detection (Figure 4a) and density estimation (Figure 4b) show different properties, mainly related to the sizes of the input window and of the CNN used. Sequential HD is ∼50% faster than DE: this effect is due to the fact that HD uses a much simpler CNN and, even if the CNN has to



(a) Head detection CNN

(b) Density estimation CNN

Figure 4: Energy and execution time per image vs frequency.

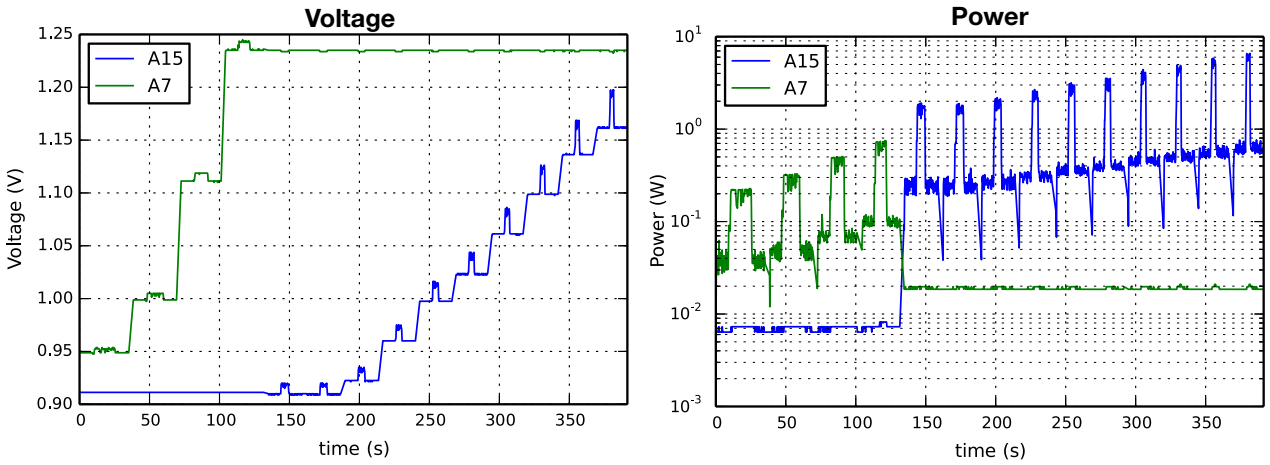


Figure 5: Power and voltage measured while running the density estimation CNN with 4 threads, sweeping maximum frequency from 300 MHz to 1600 MHz in 100 MHz steps.

operate many times on each image, the total computational burden is simply smaller than that of DE. The small size of the input window (32x32) limits the amount of fine-grained data parallelism that can be extracted from HD. In fact, with 4 threads the thread creation overhead in HD would be greater than the potential gain from fine-grained parallelism. HD was therefore parallelized using coarse-grained parallelism, i.e. with each thread applying the CNN to a different window. By contrast, DE's much larger CNN works on whole pictures and can largely exploit fine-grained data parallelism.

Figure 4 also shows the total energy consumption for both the A15 and the A7 clusters, measured through the Odroid-XU onboard sensors. Using the *ondemand* governor of the Linux kernel, the A7 cluster is used until the maximum frequency is set to 600 MHz, while the A15 is used from 700 MHz onward. It is very interesting to note that for our task the performance gain when we pass from sequential to parallel execution also leads to improved energy efficiency, while the gain due to higher clock frequencies always results in a greater energy footprint. Moreover, it is clear that powering up the A15 cluster is worth it only if there is a performance constraint, which is not our case (one picture per minute is the highest rate). At the most efficient point (4 threads, 300 MHz), HD consumes 3.97 J/image, and DE 4.76 J/image. The battery of a smartphone such as a Samsung Galaxy S4 can hold more than 9.8 Wh (35 kJ), which means that (neglecting idle power) up to 8750 images could be processed before needing a recharge ($\sim$40 days at the same rate of 216 images/day used in Section V-A).

Figure 5 shows voltage and power readings for the DE case (HD ones are qualitatively similar). The voltage plot shows voltage scaling applied to different clusters when changing the operating frequency. We can see the much finer voltage control that is done on the A15 domain with respect to the A7 domain and the direct impact it has on power. The voltage spikes we see aligned with the maximum activity are due to the attempt by the external SMPS to compensate for the increase of current consumption and the resulting IR drop at transistor

level. Power consumption has a discontinuity around 700 MHz where the A15 cluster turns on; this discontinuity is caused not only by a higher consumption of the A15 cluster but also by the higher static power of the A7 when idle. The main reason is that the A7 cluster never enters a deep sleep state while the A15 does. The performance speedup of the A15 over the A7 does not match the power increase; this is the cause of the rapid decrease of energy efficiency seen in Figure 4 between 600 and 700 MHz.

VI. Conclusion

We have demonstrated two implementations of neuromorphic algorithms for classroom occupancy estimation on a standard off-the-shelf embedded SoC. The two techniques presented show that BICV methods such as convolutional neural networks can attain very good results in complex tasks[2], and have therefore great potential in a variety of embedded CV applications such as automotive and biomedical.

We also show that it is possible to achieve these results on off-the-shelf hardware with good energy efficiency, spending as few as 3.97 J/image. To further lower power consumption in neuromorphic applications and meet the needs of more constrained applications, we believe a new generation of highly optimized brain-inspired computing architectures will be needed.

Acknowledgements

This work was supported by the EU Project P-SOCRATES (FP7-611016) and the EU ERC-Advanced Grant MULTITHERMAN (FP7-291125).

References

[1] Improving Photo Search: A Step Across the Semantic Gap. http://googleresearch.blogspot.ca/2013/06/improving-photo-search-step-across.html. 1

[2] A. Al Maashri, M. Cotter, N. Chandramoorthy, M. DeBole, C.-l. Yu, V. Narayanan, and C. Chakrabarti. Hardware Acceleration for Neuromorphic Vision Algorithms. *Journal of Signal Processing Systems*, 70(2):163–175, Sept. 2012. 1

[3] M. Andriluka, S. Roth, and B. Schiele. People-tracking-by-detection and people-detection-by-tracking. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, June 2008. 2

[4] G. Anthes. Deep learning comes of age. *Communications of the ACM*, 56(6):13, June 2013. 1

[5] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-farley, and Y. Bengio. Theano : A CPU and GPU Math Compiler in Python. (Scipy):1–7, 2010. 2

[6] L. Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186. Physica-Verlag HD, 2010. 2

[7] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool. Online Multi-Person Tracking-by-Detection from a Single, Uncalibrated Camera. *IEEE transactions on pattern analysis and machine intelligence*, 33(9):1820–1833, Dec. 2010. 2

[8] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, Y. Gong, and A. Acero. Recent advances in deep learning for speech research at Microsoft. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8604–8608, May 2013. 1

[9] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers. page 9, Feb. 2012. 1

[10] C. Farabet, C. Couprie, L. Najman, and Y. Lecun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–29, Aug. 2013. 1, 2, 3

[11] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. NeuFlow: A runtime reconfigurable dataflow processor for vision. *Cvpr 2011 Workshops*, pages 109–116, June 2011. 1

[12] K. Fukushima, S. Miyake, and T. Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):826–834, Sept. 1983. 1

[13] C. Garcia and M. Delakis. Convolutional face finder: a neural architecture for fast and robust face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(11):1408–1423, Nov 2004. 2

[14] Y.-l. Hou and G. K. H. Pang. People Counting and Human Detection in a Challenging Situation. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 41(1):24–33, Jan. 2011. 2

[15] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007. 3

[16] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962. 1

[17] A. Krizhevsky and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. pages 1–9. 1

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1, 2

[19] A. A. Maashri, M. Debole, M. Cotter, N. Chandramoorthy, Y. Xiao, V. Narayanan, and C. Chakrabarti. Accelerating neuromorphic vision algorithms for recognition. *Proceedings of the 49th Annual Design Automation Conference on - DAC '12*, page 579, 2012. 1

[20] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–60, Nov. 2002. 2

[21] J. Mutch and D. Lowe. Multiclass Object Recognition with Sparse, Localized Features. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1 (CVPR'06)*, volume 1, pages 11–18. IEEE, 2006. 1

[22] T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. a. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier. Six networks on a universal neuromorphic computing substrate. *Frontiers in neuroscience*, 7(February):11, Jan. 2013. 2

[23] B. Schrauwen, D. Verstraeten, and J. Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pages 471–482, 2007. 2

[24] P. Serman, K. Kavukcuoglu, and Y. LeCun. EBLearn: Open-Source Energy-Based Learning in C++. *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, pages 693–697, Nov. 2009. 2, 3

[25] T. Serre, L. Wolf, and T. Poggio. Object Recognition with Features Inspired by Visual Cortex. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2:994–1000. 1

[2] People counting is not only complex for CV, but is actually a challenging task also for humans. The tagging of our ground truth dataset had to be repeatedly revised due to human errors.