Smart imagers modeling and optimization framework for embedded AI applications

Luis Cubero, Arnaud Peizerat, Dominique Morche, Gilles Sicard

Smart imagers modeling and optimization framework for embedded AI applications

Luis Cubero, Arnaud Peizerat, Dominique Morche, Gilles Sicard
Univ. Grenoble Alpes
CEA-LETI
Grenoble, France
{luisangel.cuberomontealegre, arnaud.peizerat, dominique.morche, gilles.sicard}@cea.fr

***Abstract*—This work presents a framework for behavioral simulations of smart imagers with hardware and power constraints. The objective is to compare innovative imaging systems that would be composed of a specific image sensor and a dedicated image processing. For that purpose, a versatile imager model is presented and applied to a time-to-first-spike imager associated with two types of neural networks. Image classification is targeted to assess the system performance, namely the classification accuracy and data throughput. Simulation results depict/show the impact of different key-parameters helping in the choice of the final imaging system architecture.**

***Index Terms*—Smart imagers, behavioral simulation, spiking imagers, embedded artificial intelligence**

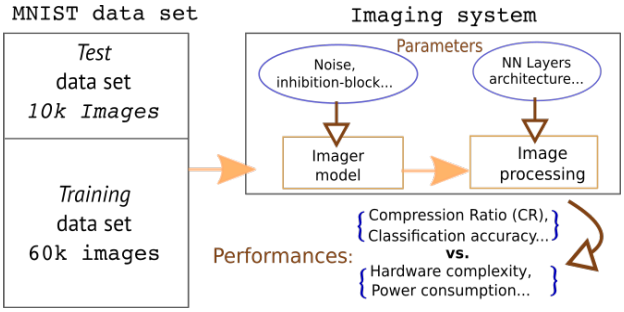


Fig. 1: Simulation framework block diagram

I. Introduction

Artificial intelligence (AI) algorithms, such as convolutional neural networks (CNNs) [1], can be used for image classification within 1000 categories [2]. However, embedded device design requires to take into account hardware and power constraints, besides the performance of the AI algorithm [3] [4]. As a consequence, embedded AI architectures exhibit a trade-off between AI's performance and hardware/power limitations [5]. In the computer vision field, this problem has been addressed with different approaches. For instance, [6] proposed to distribute the signal processing task among the digital and analog domains for face detection and identification. Also, non-conventional and event-based schemes like "time to first spike (TTFS)" imagers [7], and the "time based CMOS dynamic vision and image sensor (ATIS)" [8], have been exploited and adapted to specific application scenarios. For example, [9] proposed an extension of a TTFS imager, with an image compression technique, that reduces the events related data throughput. In addition, [10] reported an ATIS based system for object tracking applications.

Even though new ideas could improve further embedded AI's performance, proof-of-concept circuits are costly to be fabricated and tested. Then, having a simulation tool that enables characterizing performances from different (standard or not) imager types may be critical. Previous efforts have been done for modeling and simulating analog [11] or mixed signal architectures [12], in such a way that lower level models from complex sub-blocks can be used in higher level simulations. Nevertheless, computer vision AI applications carry out (pre) processing steps that require to be tested and verified all together. Pre-processing stages may be linked to the analog/mixed-signal (A/MS) imaging system, and may have an impact on the further all-digital processing stages. This work proposes a framework for behavioral simulations of the whole signal processing chain: from image acquisition to the AI's output. Overall features of this imaging system can be extracted as a function of its parameters (e.g. classification accuracy vs. ADC resolution of the imager). Such a tool would boost research and development of embedded AI on smart imagers. Lower level device features (e.g. delays, noise, mismatch, etc.) can be taken into account in further works. For said purposes, a model which tries to generalize smart imagers in a modular fashion is presented.

This paper is organized as follows : the second (next) section gives our general simulation framework, the third section describes a specific non-conventional imager type that was modeled. Then, section four shows some simulated results before conclusions are drawn.

II. Simulation Framework

The overall simulation framework is presented in fig. 1. The handwritten digits (0-9) data set MNIST [13] was used as an example of classification of images between 10 classes (one for each digit). 8-bit-encoded images from that data set were considered as the "ground truth", and they were used to feed the imager model. The imager model output was used as an input for the AI processing stage. The AI part was implemented with an on-the-shelf framework for machine learning algorithms: TensorFlow [14]. In that illustration, two neural network (NN) types (further explained in [15]) were taken into account: a multi-layer perceptron (MPL) and a CNN

(simple architectures were used for proof of concept). The MPL was the same as the one used in [16]: the 784 inputs (pixel's intensities) were densely connected to 128 neurons with Rectifier Linear Unit (ReLU) activation, following a 10 neuron layer with softmax activation. The output encoded the classified class. The CNN was a simplification of the one explained in [17]: it was composed of a valid convolution layer (stride of 1) with 32 kernels of dimension 3x3, and with ReLU activation. The final layer was the same as for the MPL case.

The code was made in Python: Object Oriented Programming (OOP) was used, since inheritance allows easy representation of small variations of imager types. The simulation algorithm can evaluate imaging system performances (e.g. accuracy for image classification), and it can approximate analog pre-processing complexity for many imager types (thanks to a generalized model explained in section III). That can be mapped to a specific electronic architecture and a respective power consumption, which will be the subject of a further work.

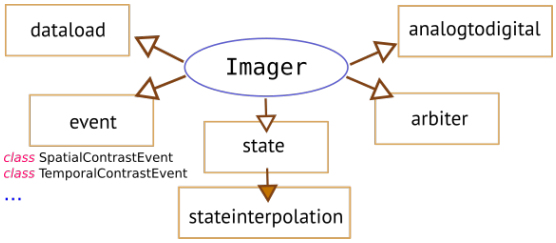


Fig. 2: Python modules organization to represent and decompose the imager's parts.

The code is organized in Python modules that are linked to electronic parts of one "general imager". It can be reduced to a specific and simplified case. The module "imager" contains models of different imager types as OOP classes. It also uses (imports) the other modules (by instantiating objects from those other modules' classes as imager attributes) to simulate electronic components behaviors. This sets a module hierarchy depicted in fig. 2. The module "dataload" loads the dataset and feeds the frames (images) one by one to the model. Modules "event" and "arbiter" are used to represent behaviors of, for example, spiking and/or event based schemes [7]–[10]. The module "state" holds the pixel's output voltages, and other state variables (such as the current time in the simulation) related to a specific imager. The "stateinterpolation" module is used for mimicking physical and transient phenomena associated to photo-current integration. This module can be extended, so more complex/complete interpolations could be simulated or emulated. The module "analogtodigital" can include more electronics besides the ones related to the analog-to-digital conversion, and it will be further explained in section III.

In order to include spiking (event based) imagers in the framework, a general conceptual model, and compatible with the modules in fig. 2, is presented next.

III. Conceptual Model of Spiking Imagers

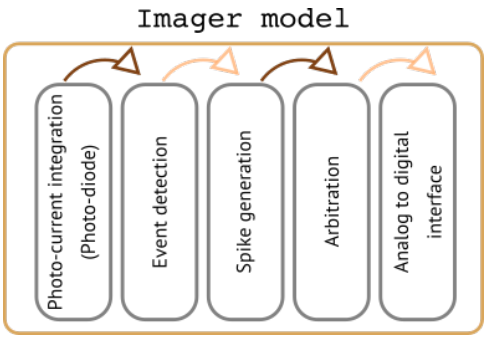


Fig. 3: The five electronic stages of a spiking or general imager

For TTFS imagers, the spike encodes the light intensity by detecting the time the pixel's output voltage reaches a threshold [7]. In the case of the ATIS pixel, the spike encodes a light intensity change [8]. Our imager model is intended to simulate all kind of spike-like signals. Fig. 3 gives the imager signal chain simulated with modules in fig. 2. After an event is detected, a spike is generated. Pixel output information (for example, the address of the spiking pixel) could be given directly as an input to the digital domain for processing, as in the Address Event Representation (AER) scheme, exploited by TTFS and ATIS pixels [8], [9]. In addition, pixels outputs (e.g. a spike, or a voltage) could be pre-processed right before the analog-to-digital conversion. One example is given in [6], where data flow is processed in the analog domain before the digital part. This kind of pre-processing has been represented (included) in a simplistic way as the "analog to digital interface" in fig. 3, and its behavior is contained in the module "analogtodigital" in fig. 2.

A. General Algorithm

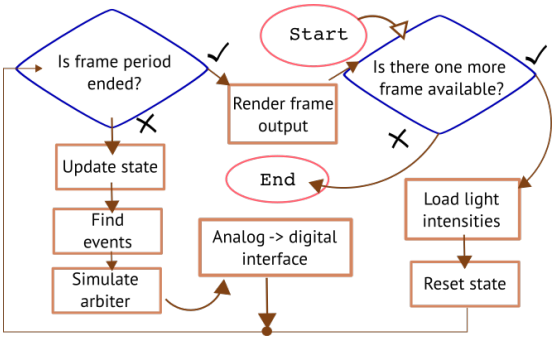


Fig. 4: General algorithm to simulate the imager's behavior.

Thanks to the implementation of the algorithm presented in fig. 4, transient phenomena (e.g. time-dependent parasitic effects) can be included in the modules (to be done in further works). The simulation input is a set of images that could or could not represent a video stream. After all images are loaded, they are fed one by one into the simulated imager

model, so those original frames are interpreted as light intensity (ground truth) values encoded in 8-bits. The reset step is optional, depending on whether the imager is frame based (i.e. contains a global reset) or not. The imager's state starts being updated until an increment variable (for time representation) reaches the frame period. One can note that this frame period is related to the imager which was used to take the original frame set. The state of the imager (i.e. pixel's output voltage) is updated with any interpolation rule within the "stateinterpolation" module. After each update, the time tracking variable is updated as well. The time step of this time tracking variable, in comparison to the frame period of the imager used for making the original data set, is related to the simulation accuracy. A simple hypothesis, already used in the literature [7], is that the photo-current is constant during a frame. Then, the photo-generated current in the pixel will make its output voltage to decrease linearly after each state update. The slope is related with the overall pixel optical efficiency, the electronic signal amplification and the photodiode's capacitance. Those parameters can be measured or approximated.

The simulation is intended for including an arbitrary level of specificity for imager models, as long as they stick to the hierarchy showed in fig. 2. It also aims for modular and easy-to-codve simulations prior to computational efficiency, as long as simulation time stays within an acceptable range. For example, after defining the *OPP class* "GenericImager", which would have as *child class* "TTFS", then specific adaptations of the TTFS architecture (as the one presented in section IV) can be defined by creating *child classes* of *parent* imagers.

Computational consuming matrix operations are vectorized as it helps to decrease the simulation time. Nevertheless, a more detailed optimization for decreasing simulation time will be a subject for further works. Also, the proposed algorithm may not be the most efficient for all imager types and simulation objectives. Indeed, that was a sort of trade-off in order to keep the overall model as general as possible.

IV. Simulation example: TTFS imager with inhibition

The TTFS imagers scheme has been extended in [9] to include an image/video compression architecture. The operation principle is as follows: the whole pixels matrix is divided into equally sized sub-groups of adjacent pixels called "blocks" [9]. In this work, the notation (No. rows, No. columns) is used to characterize the size/shape of a block, which contains (No. rows) * (No. columns) adjacent pixels. One Time Inhibition Control Circuit (TICC) is assigned to each block. After a global reset, each TICC waits for the first spiking pixel. When the first pixel p_f of a block B spikes, its output intensity I_p is read and all output pixels associated to B get that same value. Then, B is inhibited by the TICC during an inhibition time t_{inhib} and p_f is set to off mode until the next global reset. During that period, if another pixel in B spikes, it is put into off mode till the next global reset, and its output value is not assigned to the output image. Once t_{inhib} has passed, other spiking pixels are not inhibited. After that, the imager behavior is as a standard TTFS imager until the next global reset (i.e. if a non set-to-off pixel p_i spikes, its output value is changed in the output image, and only for its corresponding address). Three key parameters are the block's shape, size, and inhibition time. Their impact for image classification is illustrated in the next section.

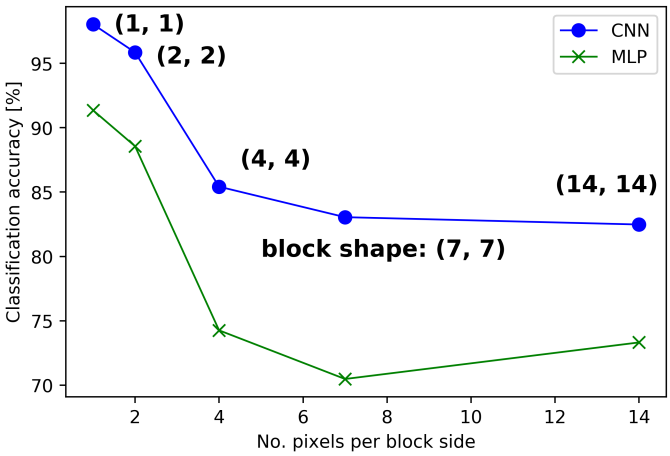
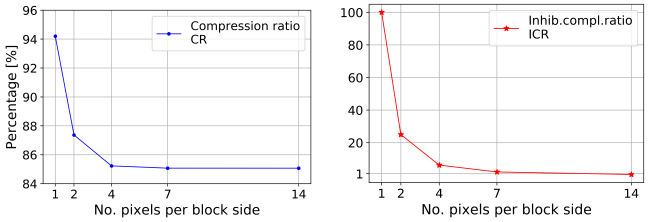


Fig. 5: MNIST classification accuracy vs No. of pixels N per block side for an inhibition block of shape (N, N) for $t_{inhib}/t_{min} = 5$.

The flow depicted in fig. 1 was used to optimize the classification performance in accordance with the block's size and t_{inhib}. Please notice that, for results showed in figures 5-7, t_{min} is an imager parameter expressing the "shortest integration time" [7]. It is assumed to be a constant. Also, in [7] they explain that it is the minimal time in which a pixel can output a spike. The hand-written data set MNIST was used as an example: it is composed of 70 000 8-bit, gray-scaled images of 28 x 28 pixels. It is divided in 10000 images for testing, and 60000 for training. First, gray scaled pixel values were "inverted" (i.e. if a pixel's value was x, its value was updated to 255.0 - x), since handwritten letters were made in white over a black background. That would favor the compression ratio CR = (No. of non-inhibited events) / (max possible No. of events) for each frame, without reflecting the worst case. The imager's simulated output was used as an input for a MLP and a CNN. Fig. 5 shows the classification accuracy vs. block shape. Increasing block size has a higher impact when blocks are relatively small, and it diminishes for bigger blocks at constant t_{inhib}. NN's parameters, such as NN type and layers architecture, allow calculating computational complexity and memory requirements: 259872 parameters / 201480 multiplications for the CNN, and 102544 parameters / 101632 multiplications for the MLP (without taking into account activations). Fig. 6 shows the behavior of two figures of merit corresponding to the CR and the Inhibition Complexity Ratio ICR = (No. of TICC units)/(total No. of pixels). Those figures suggest that increasing the block size decreases

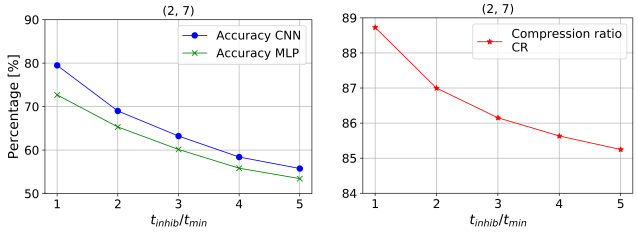
both the number of TICCs and data throughput, which can potentially reduce power consumption as well. That exhibits an expected trade-off with classification accuracy, since smaller block sizes gave better classification performances.



(a) Compression Ratio (CR) (b) Inhib. Complexity Ratio (ICR)

Fig. 6: Figures of merit as a function of (squared) block size for $t_{inhib}/t_{min} = 5$.

With respect to the previous example, the simulation flow was slightly modified. This time, the MNIST *training* data set was not carried-out by the imager model. That represents a case of using a compressing imager to classify images with the NN that has been (pre) trained with a standard (or TTFS) imager's output. Within that context, the effect of changing t_{inhib} is depicted in fig. 7 for a block shape of (2, 7). Obtained results agree with the ones given by [9], since data throughput (given by the CR) decreases with increasing t_{inhib} for constant block size. Moreover, the same trade-off arises between the CR and classification accuracy.



(a) Classification accuracy (b) Compression Ratio (CR)

Fig. 7: Imager performance when varying t_{inhib} for a block of shape (2, 7).

V. Conclusion

A framework for behavioral simulations of smart imagers was presented. It is intended to facilitate high level comparison and characterization for (non) standard architectures, and within the context of AI applications. One general and modular imager model was proposed. As an example, it was reduced to the case of a specific adaptation of a Time-to-first-spike imager, reflecting that classification accuracy on the MNIST dataset kept over 90 % for a block of size (2, 2). The trade-off between data throughput and classification performance has been illustrated. Further works could include specific (transient) models of A/MS electronic blocks. Then, computational complexity and data throughput could be mapped to power consumption or frame rates.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[2] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[3] R. C. Çalik and M. F. Demirci, "Cifar-10 image classification with convolutional neural networks for embedded systems," in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, Oct 2018, pp. 1–2.

[4] W. Benjilali, W. Guicquero, L. Jacques, and G. Sicard, "A low-memory compressive image sensor architecture for embedded object recognition," in *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2018, pp. 881–884.

[5] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8μj/86memory on chip in 28nm cmos," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 222–224.

[6] C. Kim, K. Bong, I. Hong, K. Lee, S. Choi, and H. Yoo, "An ultra-low-power and mixed-mode event-driven face detection soc for always-on mobile applications," in *ESSCIRC 2017 - 43rd IEEE European Solid State Circuits Conference*, Sep. 2017, pp. 255–258.

[7] X. Guo, X. Qi, and J. G. Harris, "A time-to-first-spike cmos image sensor," *IEEE Sensors Journal*, vol. 7, no. 8, pp. 1165–1175, Aug 2007.

[8] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 db 15μs latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb 2008.

[9] C. Dupoiron, A. Verdant, and G. Sicard, "Smart pixel architecture for low power cmos image sensor: Time-to-first spike with inhibition mechanism," in *2017 15th IEEE International New Circuits and Systems Conference (NEWCAS)*, June 2017, pp. 49–52.

[10] F. Gómez-Rodríguez, L. Miró-Amarante, F. Diaz-del-Rio, A. Linares-Barranco, and G. J. Robotics, "Real time multiple objects tracking based on a bio-inspired processing cascade architecture," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 1399–1402.

[11] H. Filiol, I. O'Connor, and D. Morche, "Piecewise-polynomial modeling for analog circuit performance metrics," in *2009 European Conference on Circuit Theory and Design*, Aug 2009, pp. 237–240.

[12] Y. Blanchard, A. Dupret, and A. Peizerat, "Systemc modelization for fast validation of imager architectures," in *Proceedings of the 2011 Conference on Design Architectures for Signal Image Processing (DASIP)*, Nov 2011, pp. 1–5.

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

[14] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[15] S. Khan, H. Rahmani, S. A. A. Shah, and M. Bennamoun, "A guide to convolutional neural networks for computer vision," *Synthesis Lectures on Computer Vision*, vol. 8, no. 1, pp. 1–207, 2018.

[16] F. Chollet. (2017) Train your first neural network: basic classification. Internet draft. TensorFlow. Accessed: February 8th, 2019. [Online]. Available: https://tensorflow.org/tutorials/keras/basic_classification

[17] E. Allibhai. (2018, October) Building a convolutional neural network (cnn) in keras. Internet draft. Towards Data Science. Accessed: February 8th, 2019. [Online]. Available: https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5