Improving Performance Estimation for FPGA-based Accelerators for Convolutional Neural Networks

Martin Ferianc[1,§], Hongxiang Fan[2], Ringo S. W. Chu[3], Jakub Stano[4], and Wayne Luk[2]

[1] Department of Electronic and Electrical Engineering, University College London, London, UK
martin.ferianc.19@ucl.ac.uk
[2] Department of Computing, Imperial College London, London, UK
{h.fan17, w.luk}@imperial.ac.uk
[3] Department of Computer Science, University College London, London, UK
ringo.chu.16@ucl.ac.uk
[4] Department of Information Technology and Electrical Engineering, ETH Zurich, Zurich, Switzerland
jstano@ethz.ch

Abstract. Field-programmable gate array (FPGA) based accelerators are being widely used for acceleration of convolutional neural networks (CNNs) due to their potential in improving the performance and reconfigurability for specific application instances. To determine the optimal configuration of an FPGA-based accelerator, it is necessary to explore the design space and an accurate performance prediction plays an important role during the exploration. This work introduces a novel method for fast and accurate estimation of latency based on a Gaussian process parametrised by an analytic approximation and coupled with runtime data. The experiments conducted on three different CNNs on an FPGA-based accelerator on Intel Arria 10 GX 1150 demonstrated a 30.7% improvement in accuracy with respect to the mean absolute error in comparison to a standard analytic method in leave-one-out cross-validation.

Keywords: Field-Programmable Gate Array · Deep Learning · Convolutional Neural Network · Performance Estimation · Gaussian Process

1 Introduction

Field-programmable gate arrays (FPGAs) are becoming increasingly popular in the deep learning community, particularly in the acceleration of convolutional neural networks (CNNs) [4,11,5]. This acceleration is achieved by parallelising

§Corresponding author.

the extensive concurrency exhibited by CNNs. As such, FPGA is ideal as the platform allows implementation of fine-grain parallelisations. To do an architectural exploration and determine the optimal hardware configuration, it is necessary to estimate the performance with respect to multiple different hardware specifications.

There are several performance estimation frameworks for reconfigurable FPGA-based accelerators [19,2,3], however, estimating the performance without knowing about scheduling is still a very challenging task because of two main reasons. First, the explicit time to execute a certain operation on hardware varies by on/off-chip communication, synchronisation, control signals, I/O interruptions and in particular for the CNN accelerators - the CNN's architecture, which complicate analytic estimation. Second, it is difficult to accurately select the most representative design features for all hardware specifications during the performance estimation.

In this paper, we propose a novel approach for performance estimation for FPGA-based CNN accelerators [11]. This method constitutes a Gaussian process (GP) [18] coupled with a standard analytic method and statistical data. Gaussian process is a stochastic process, such that every finite collection of random variables has a multivariate normal distribution [15]. Experiments were conducted on three different CNNs on Intel Arria GX 1150 FPGA and we compared the method to linear regression (LR), GP with zero mean function, GP with an artificial neural network (ANN) mean function [6], gradient tree boosting (GTB) and ANN in estimating latency. We show that the proposed method achieved the top result among all compared methods.[1]

In Section 2 we demonstrate the standard approach for analytic performance estimation. Afterwards, in Section 3 we introduce the proposed method, followed by Section 4 where we describe the accelerator as well as the dataset on which we benchmarked the method. Then we present the evaluation in Section 5 followed by a conclusion in Section 6.

2 Background

The most accurate method of determining the performance is escalating the CNN onto the hardware. One major drawback of this method is requiring resynthesis and re-implementation for different hardware specifications. Therefore, it is more feasible and practical to perform the design space exploration (DSE) [9] with respect to an estimate of the performance in a software level, rather than running the CNN each time for a different hardware configuration.

Even with a more advanced option of performance estimation, high irregularity within a complex accelerator results in case-by-case estimation. Therefore, this approach is unfeasible in general case, as it is usually constrained to a single hardware configuration. In our work, we are focused on estimating one particular aspect of performance - latency for a CNN reconfigurable accelerator.

[1] A tutorial code is available at https://git.io/Jv31c.

The standard 2D convolution layers, from which the CNN is constructed, occupy over 90% of the overall processing time [17] and their latency T_i on the accelerator needs to be estimated to determine the best hardware configuration through DSE. For 2D convolution, there are several categories of parallelism including filter parallelism (PF) or channel parallelism (PC) in addition to spatial and kernel parallelisms. These are the parameters that usually need to be determined during the DSE.

A performance estimation framework for reconfigurable dataflow platforms was proposed by Yasudo *et al.* [19] that can analytically determine the number of accelerators suitable for the application. Dai *et al.* [2] proposed an estimation method based on a GTB and a high-level synthesis report and they compared it with LR and ANN. However, their method requires a significant amount of data and features from the synthesis report, which might not be available, especially when high-level synthesis is not being used to describe the accelerator. Enzler *et al.* proposed a general heuristic-based method [3] for estimating the performance of accelerator designs, which can be modified for CNN accelerators and is now used as the standard method.

Table 1. Notation used for performance estimation in an FPGA-based accelerator for convolutional neural networks.

Parameter	**Description**
H	Height of input feature map
W	Width of input feature map
H_O	Height of output feature map
W_O	Width of output feature map
K	Kernel size
F	Number of filters
C	Number of channels
PF	Parallelism in filter dimension
PC	Parallelism in channel dimension
M_{CLK} [MHz]	Memory access clock cycle time
L_{CLK} [MHz]	Logic clock cycle time
M_{EFF} [%]	Memory transfer efficiency
S [bits]	Memory transfer size
DW [bits]	Processing data width
M	Number of input features
N	Number of layers in a CNN
P	Number of training samples

The simplest form of a heuristic for estimating latency on a hardware accelerator consists of dividing the overall processing time for a single input T into time steps T_i which correspond to the time to perform one 2D convolution in a feed-forward CNN consisting of N 2D convolutions. The total estimated latency for the CNN in that given configuration is then simply added as $T = \sum_{i=1}^{N} T_i$.

Table 2. Number of operations and a data size for a 2D convolution i.

Sizes	**Number of operations/Data size**
Number of compute operations	$F_i \times C_i \times H_i \times W_i \times K_i \times K_i$
Input size	$H_i \times W_i \times C_i$
Weights size	$F_i \times C_i \times K_i \times K_i$
Output size	$H_{O_i} \times W_{O_i} \times F_i$

The time T_i is being split into three different terms: (1) On-chip memory loading time T_{load_i}, (2) Computation time $T_{compute_i}$ and (3) Off-chip memory storing time T_{store_i}. Assuming the design is pipelined, the runtime T_i is then decided by the slowest path which is chosen by the maximum among T_{load_i}, $T_{compute_i}$ and T_{store_i}. Each of these terms depends on a mixture of parameters that are specified by the 2D convolution: *Input size, Output size, Number of compute operations*, device specific settings: *Memory bandwidth, Clock cycle time* or the hardware architecture: *Parallelism*, which are known prior to making a prediction. The estimated latency per layer is then computed as shown in Equations 1 below

$$T_{load_i} = \frac{\text{Input size}}{\text{Memory bandwidth}} \qquad T_{compute_i} = \frac{\text{Number of compute operations}}{\text{Clock cycle time} \times \text{Parallelism}}$$
$$T_{store_i} = \frac{\text{Output size}}{\text{Memory bandwidth}} \qquad T_i = max\left(T_{load_i}, T_{compute_i}, T_{store_i}\right) \tag{1}$$

The heuristic approach does not depend on any statistical data to perform the estimation and it is simple to implement since it relies only on the features that can be easily read from the respective datasheets. Nevertheless, this general estimation method usually computes the most optimistic estimate and it does not leave room for delays caused by communication, synchronisation or control. One way to refine the estimation is that we can collect runtime data and use this data to improve the estimate. Therefore, in our work, we are proposing to use the standard analytic method as a mean function inside a GP together with the profiling data collected by running the CNN on real hardware to train the GP to model the observed misestimation.

3 Gaussian Process with an Analytic Mean Function

GP is a modelling function built around Bayesian modelling which can embody our prior knowledge/model into our target [15]. A GP is specified by a mean function $m(.)$ and a covariance function (kernel) $k(.,.)$. The mean function represents the supposed average of the estimated data. The kernel computes correlations between inputs and it encapsulates the structure of the hypothesised function. The main benefit of using a GP over other methods such as LR, GTB or ANN is that it can use the developed analytic foundations, such as the standard analytic performance estimation, as prior knowledge in a form of $m(.)$.

The predictive distribution of the GP, $p(\mathbf{y_T}|\mathbf{X}, \mathbf{y}, \mathbf{X_T})$ for the targets $\mathbf{y_T}$ given the corresponding features $\mathbf{X_T}$ and the training data $\mathbf{X}, \mathbf{y}$ is defined as a multivariate Gaussian distribution $\mathcal{N}$ with a predictive mean $\mathbb{E}[\mathbf{y_T}|\mathbf{X}, \mathbf{y}, \mathbf{X_T}]$ and a predictive variance $\mathbb{V}[\mathbf{y_T}|\mathbf{X}, \mathbf{y}, \mathbf{X_T}]$.

The $\mathbf{X} \in \mathcal{R}^{P \times M}$ and $\mathbf{X_T} \in \mathcal{R}^{N \times M}$ are the sets of M features for P samples for training and N samples for testing. The $\mathbf{y} \in \mathcal{R}^{P}$ and $\mathbf{y_T} \in \mathcal{R}^{N}$ are the target objectives corresponding to the number of samples per dataset respectively. The $\mathbb{E}[\mathbf{y_T}|\mathbf{X}, \mathbf{y}, \mathbf{X_T}]$ is defined in Equation 2 below as

$$\underline{m(\mathbf{X_T}) \backsim T_i(\mathbf{X_T})} + k(\mathbf{X_T}, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I})^{-1}(\mathbf{y} - \underline{m(\mathbf{X}) \backsim T_i(\mathbf{X})}) \tag{2}$$

and $\mathbb{V}[\mathbf{y_T}|\mathbf{X}, \mathbf{y}, \mathbf{X_T}]$ is defined in Equation 3 below as

$$k(\mathbf{X_T}, \mathbf{X_T}) - k(\mathbf{X_T}, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I})^{-1}k(\mathbf{X_T}, \mathbf{X})^T \tag{3}$$

where the σ^2 represents the noise amplitude and $\mathbf{I}$ is the identity matrix[2]. In the formulas above, GP possesses a set of hyperparameters associated with both the mean function and the choice of the kernel. The hyperparameter values can be found by maximising the marginal likelihood. The optimal hyperparameters are then chosen by observing the likelihood or by cross-validation.

The GP is usually used with an agnostic mean function centred at zero. However, we propose to use the previously developed latency model T_i, for each 2D convolutional layer i in a CNN, as a mean function $m(.)$ inside the predictive mean to encapsulate the known analytic model of the accelerator into the proposed method. It uses the collected data $\mathbf{X}$, which in this case are the parameters, and the hardware configuration of the accelerator for each convolution, which would normally be used in the standard analytic estimation. By also recording our past measurements from our past implementations $\mathbf{y}$, we can form a training set on which we can learn the nonlinearities that cannot be analytically modelled. The $\mathbf{X_T}$ represents the set of test features corresponding to the 2D convolutions for which we would like to estimate their target performance $\mathbf{y_T}$, in this case, latency.

Therefore, the advantage of this method in comparison to other machine learning (ML) inspired methods is that it avoids completely relying on the data while estimating the performance. Additionally, this method does not need to extract any features from the data because the features for the estimation are already known and they are the ones used in the standard analytic estimation. Hence, this method reuses previously developed knowledge by incorporating the standard method into the model as the mean function of the GP to anchor the estimate within reliable bounds. By anchoring the estimate, the model is also more interpretable in comparison to purely data-reliant methods which depend completely on the learnt features which are usually not human-readable. Additionally, by specifying the mean function and combining it with the collected data, the proposed method can give a prediction outside the observed data sample without collapsing.

[2]For a detailed derivation please refer to [15].

In the next Section, we present the FPGA-based accelerator from which we have collected the data and onto which we have evaluated our proposed method.

4 Accelerator and Dataset

4.1 Accelerator's Architecture

The per-layer latency of an implemented FPGA-based CNN accelerator is characterised according to the standard method into three parts: (1) Loading time for loading the input, (2) Computation time, (3) Storing time for storing the results.

The input has to be loaded into the on-chip memory only once for the first layer, similarly to the output being stored only once from the on-chip memory to the off-chip memory. The output of intermediate layers is buffered in the on-chip memory.

The notation is shown in Table 1 and the size of the weights and input/output for convolution is shown in Table 2. Following the standard method, the per-layer latency T_i for a single input is shown in Equations 4, 5 and 6 as follows

1. Loading time i.e., the time to load the input into the on-chip memory

$$\begin{aligned} T_{weights_i} &= \frac{K_i \times K_i \times F_i \times C_i \times DW}{PF \times M_{CLK} \times S \times M_{EFF}} \\ T_{data_i} &= \frac{H_i \times W_i \times C_i \times DW}{PF \times M_{CLK} \times S \times M_{EFF}} \\ T_{load_i} &= T_{weights_i} + T_{data_i} \end{aligned} \tag{4}$$

2. Computation time i.e., the time to compute $PC \times PF$ parallel channels and filters respectively

$$T_{compute_i} = \frac{F_i \times C_i \times H_i \times W_i \times K_i \times K_i}{PF \times PC \times L_{CLK}} \tag{5}$$

3. Storing time i.e., the time to store the output back to the off-chip memory

$$T_{store_i} = \frac{H_{O_i} \times W_{O_i} \times F_i \times DW}{PF \times M_{CLK} \times S \times M_{EFF}} \tag{6}$$

Therefore, the time required to process a single 2D convolutional layer can be written as in Equation 7 below as

$$T_i = \begin{cases} T_{i=1} & = T_{load_i} + T_{compute_i} \\ T_{i \neq 1 \vee N} & = max(T_{weights_i}, T_{compute_i}) \\ T_{i=N} & = max(T_{weights_i}, T_{compute_i}) + T_{store_i} \end{cases} \tag{7}$$

4.2 Dataset

The evaluation dataset comprises of several different configurations of 2D convolutional layers which are the building blocks of three different CNNs, namely SSD [12] with 24 2D convolutions, Yolo [16] with 75 2D convolutions and ResNet-50 [8] with 57 2D convolutions. SSD and Yolo are characteristic for their irregularities, which results in the output being produced at different times, while the ResNet is known for its residual blocks. Each network was trained in 32-bit floating-point representation and then linearly quantised into 8-bit integer representation [4]. In total giving P training samples $\mathbf{X}$ as 156 and the input feature size M being 15 corresponding to the first 15 parameters in the Table 1. The recorded latency per each convolution represents the targets $\mathbf{y}$.

Each network was executed on the implemented accelerator on Intel Arria GX 1150 FPGA. The analysis of the dataset together with the evaluation parameters can be found in Tables 3 and 4.

Table 3. Dataset for evaluation.

Parameter	Min	Mean	Max
H/W	1	42	418
H_O/W_O	1	37	416
K	1	2	7
C	3	360	2048
F	64	371	2048
Latency [ms]	0.018	0.841	11.727

Table 4. Evaluation parameters.

Parameter	Value
PC	64
PF	64
M_{CLK}	200 MHz
L_{CLK}	200 MHz
M_{EFF}	70%
S	64-bit
DW	8-bit

5 Evaluation

In evaluation, the proposed method is compared with the standard method, including a GP with a zero mean function, a GP with the ANN mean function [6], LR, GTB and ANN. The dataset described in Section 4.2 is being used to evaluate all these methods.

For a more comprehensive evaluation, leave-one-out cross-validation (LOO-CV) with respect to the mean absolute error (MAE) is used to compare the estimators. LOOCV is a particular case of leave-k-out cross-validation where $k = 1$, which means that a model is trained on all samples except one, onto which the performance is then evaluated. In this instance, the performance of the predictor is measured by the absolute error between the prediction and the target value. The error is accumulated for all samples from which the mean is then calculated by dividing the total summed error by the number of samples.

This approach was also used to determine the best hyperparameters for each regressor with respect to the LOOCV MAE. The results, as well as the individual properties and implementation details for the estimators, are summarised

in Table 5. We considered several hyperparameters for the proposed GP-based method such as the learning rate, ranging from 0.1 to 0.000001 on a logarithmic scale and the kernel, ranging from linear, Gaussian to Matérn kernels [15] and their combinations. The best parameters were found by a grid search with respect to the LOOCV MAE.

In case of the GP with the ANN mean function, it was necessary to find hyperparameters for the ANN such as the number of nodes in the hidden layers, between 16, 32 and 64 and the number of hidden layers, ranging from 1 to 3. For the activation function, we considered tanh, ReLU and sigmoid. For GTB and ANN, we needed to determine the most influential parameters such as the learning rate, ranging from 0.01 to 0.0001 on a logarithmic scale or for the GTB, the number of trees or the tree depth that was determined by gradual pruning. For the ANN we needed to decide the number of hidden nodes, between [10, 1], [10, 10, 1] and [10, 10, 10, 1] and for the activation function, we again considered tanh, ReLU and sigmoid. The hyperparameters were similarly found through a grid search with respect to the LOOCV MAE. For the standard method and LR, it was not necessary to determine any hyperparameters.

Overall, the best method proved to be the combination of the standard method and the collected data in the form of the GP with an analytic mean function. In comparison to other approaches, the proposed method achieved approximately a 30.7% improvement in LOOCV with respect to MAE decreasing to 0.312 ms in comparison to the second best-performing methods, which were LR and the standard method with 0.450 ms MAE.

The main advantage of the method lays in its implementation simplicity, as it reuses the analytic approximation that is commonly used for DSE, combined with recorded measurements. The method can be improved by recording more measurements and simple fine-tuning of the hyperparameters related to the kernel k or the analytic mean m.

A potential limitation of this method stems from the kernel computation which scales with the complexity of $O(P^3)$, which means that the inference time can be prolonged if there are many training samples. One possible solution to overcome this problem is using k-Means clustering to determine the k most important points that have to be included in the kernel. Nevertheless, the inference time is much less than the time needed for synthesis and then running the design on hardware.

6 Conclusion and Future Work

In this paper, we proposed an accurate method for estimating the performance of an field-programmable gate array-based accelerator for convolutional neural networks and compared it with the standard method and variations of the Gaussian process, linear regression, gradient tree boosting and an artificial neural network. The evaluation demonstrated that the innovative Gaussian process paired with the domain-specific knowledge and collected data can provide an approximately

Table 5. Evaluation of latency estimation for different methods.

Methods	**LOOCV MAE [ms]**	**Implementation and Optimiser**	**Properties**
Standard method	0.450	None	None
Gaussian process	0.521	GPFlow [13] - Adam [10]	*Mean function:* Zero *Learning rate:* 0.001 *Best kernel:* Matérn 3/2
Our method	**0.312**	GPFlow [13] - Adam [10]	*Mean function:* T_i *Learning rate:* 0.001 *Best kernel:* Matérn 3/2
Gaussian process with Artificial neural network mean function	0.692	GPFlow [13] - Adam [10]	*Mean function:* Artificial neural network 15, 64, 1 nodes and tanh activations *Learning rate:* 0.00001 *Best kernel:* Matérn 3/2
Linear regression	0.450	sklearn [14]	Default
Gradiet tree boosting	0.607	sklearn [14] - AdaBoost [7]	*Learning rate:* 0.1 *Number of trees:* 10 *Maximum depth:* 3
Artificial neural network	1.257	Tensorflow [1] - Adam [10]	*Batch size:* 8 *Learning rate:* 0.1 *Regulariser:* L2, 0.001 *Number of nodes:* 10,10,1 *Activations:* ReLU

30.7% accuracy improvement with respect to the standard method or the linear regression.

In the proposed method, users need to decide what are the relevant software/hardware features M together with an analytic approximation for the modelled performance that will be used as the mean function $m(.)$ in the Gaussian process. Afterwards, they need to supply the profiling data for training $\mathbf{X}, \mathbf{y}$, $\mathbf{X}$ is the feature matrix and $\mathbf{y}$ are the targets, in this case, the per-layer latency. In the end, the user needs to decide what is going to be the best kernel $k(.,.)$ and use it to train the Gaussian process to obtain the best values for the hyperparameters.[1]

In the future, we will validate the method on more configurations on different hardware boards. Furthermore, we will formulate similar analytic approximations for other potential objectives, for example, the resource usage or power consumption and use them as priors for estimating these objectives through our proposed Gaussian process-based method.

Acknowledgments. We thank Yann Herklotz, Alexander Montgomerie-Corcoran and ARC'20 reviewers for insightful suggestions.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., S., G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/
2. Dai, S., Zhou, Y., Zhang, H., Ustun, E., Young, E.F., Zhang, Z.: Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In: Proceedings of the 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). pp. 129–132. IEEE, Boulder, CO, USA (2018)
3. Enzler, R., Jeger, T., Cottet, D., Tröster, G.: High-level area and performance estimation of hardware building blocks on FPGAs. In: Proceedings of the 2000 International Workshop on Field-Programmable Logic and Applications (FPL). pp. 525–534. Springer, Villach, Austria (2000)
4. Fan, H., Liu, S., Ferianc, M., Ng, H.C., Que, Z., Liu, S., Niu, X., Luk, W.: A real-time object detection accelerator with compressed SSDLite on FPGA. In: Proceedings of the 2018 International Conference on Field-Programmable Technology (FPT). pp. 14–21. IEEE, Sakura, Japan (2018)
5. Fan, H., Luo, C., Zeng, C., Ferianc, M., Que, Z., Liu, S., Niu, X., Luk, W.: F-E3D: FPGA-based acceleration of an efficient 3D convolutional neural network for human action recognition. In: Proceedings of the 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP). vol. 2160, pp. 1–8. IEEE, New York, NY, USA (2019)

6. Fortuin, V., Rätsch, G.: Deep mean functions for meta-learning in Gaussian processes. arXiv preprint arXiv:1901.08098 (2019)
7. Friedman, J.H.: Stochastic gradient boosting. In: Computational statistics & data analysis. vol. 38, pp. 367–378. Elsevier (2002)
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). vol. 2016-, pp. 770–778. IEEE, Las Vegas, NV, USA (2016)
9. Holland, B., George, A.D., Lam, H., Smith, M.C.: An analytical model for multilevel performance prediction of multi-FPGA systems. ACM Transactions on Reconfigurable Technology and Systems (TRETS) **4**(3), 27 (2011)
10. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
11. Lian, X., Liu, Z., Song, Z., Dai, J., Zhou, W., Ji, X.: High-performance FPGA-based CNN accelerator with block-floating-point arithmetic. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **27** (2019)
12. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.: SSD: Single shot multibox detector. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 9905, pp. 21–37. Springer (2016)
13. Matthews, D.G., Alexander, G., Van Der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., Hensman, J.: GPflow: A Gaussian process library using TensorFlow. In: Journal of Machine Learning Research. vol. 18, pp. 1299–1304 (2017)
14. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
15. Rasmussen, C.E.: Gaussian processes in machine learning. The MIT Press (2005)
16. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). vol. 3, pp. 779–788. IEEE, Las Vegas, NV, USA (2016)
17. Venieris, S., Kouris, A., Bouganis, C.S.: Toolflows for mapping convolutional neural networks on FPGAs: A survey and future directions. In: ACM Computing Surveys (CSUR). vol. 51, pp. 1–39. ACM (2018)
18. Williams, C.K., Rasmussen, C.E.: Gaussian processes for regression. In: Advances in neural information processing systems. pp. 514–520 (1996)
19. Yasudo, R., Coutinho, J., Varbanescu, A., Luk, W., Amano, H., Becker, T.: Performance estimation for exascale reconfigurable dataflow platforms. In: Proceedings of the 2018 International Conference on Field-Programmable Technology (FPT). pp. 314–317. IEEE, Sakura, Japan (2018)