ML-based Power Estimation of Convolutional Neural Networks on GPGPUs

Christopher A. Metz[1] Mehran Goli[1,2] Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{cmetz, mehran, drechsler}@uni-bremen.de

***Abstract*—The increasing application of *Machine Learning* (ML) techniques on the *Internet of Things* (IoTs) has led to the leverage of ML accelerators like *General Purpose Computing on Graphics Processing Units* (GPGPUs) in such devices. However, selecting the most appropriate accelerator for IoT devices is very challenging as they commonly have tight constraints e.g., low power consumption, latency, and cost of the final product. Hence, the design of such application-specific IoT devices becomes a time-consuming and effort-hungry process, that poses the need for accurate and effective automated assisting methods.**

In this paper, we present a novel approach to estimate the power consumption of CUDA-based *Convolutional Neural Networks* (CNNs) on GPGPUs in the early design phases. The proposed approach takes advantage of a hybrid technique where static analysis is used for features extraction and the *K-Nearest Neighbor* (K-NN) regression analysis is utilized for power estimation model generation. Using K-NN analysis, the power estimation model can even be created with small training datasets. Experimental results demonstrate that the proposed approach is able to predict CNNs power consumption up to a *Absolute Percentage Error* of 0.0003% in comparison to the real hardware.

I. Introduction

The number of IoT devices that leverage *Machine Learning* (ML) algorithms are considerably increased in the last decade, ranging from manufacturing to scientific-, health- and security-related applications [1], [4], [5]. Among the existing ML algorithms, *Convolutional Neural Network* (CNN) is widely used in pattern recognition tasks and image analysis due to its ability to handle large and unstructured data [15]. However, CNNs require high computational resources. For example, the convolutional layers, made up of 4-dimensional convolutions, are responsible for over 90% of the computation and require processing massive amounts of data with potentially trillions of computations per second [14]. Due to these huge computations, designers take advantage of hardware accelerators such as *General Purpose Computing on Graphics Processing Units* (GPGPUs) to gain performance and meet the time-to-market constraints.

One of the major challenges that designers are commonly faced during the design phase of such IoT devices is to choose the right ML accelerator that adheres to the design constraints such as low power consumption, latency, and cost of the final products [2], [3]. For example, assume that designers need to design an IoT device where its CNN application is performed on a GPGPU (as hardware accelerator). In the case that the power consumption and battery lifetime of the IoT device are considered as the design constraints, choosing the most proper GPGPU early in the design phase can significantly avoid costly design loops occurring as fewer prototypes need to be built. Moreover, in the case of Cloud-based IoT devices where data processing of the CNN application performs remotely on Cloud-based accelerators (i.e., GPGPUs), choosing an appropriate GPGPU can significantly reduce the renting cost, resulting in a direct impact on the cost of the final product.

Power estimation techniques have been shown as a promising solution to approach this issue. A robust power estimation approach enables designers to choose the most appropriate GPGPU that meets the constraints, early in the design phase. Existing methods mostly rely on so-called performance counters [6]–[8] to estimate power consumption. As a consequence, their estimation depends on the run-time data, meaning the ML model must be run once on the target GPGPU that the performance counter results can be measured. However, this can limit the usage of such methods in the early design phase as the GPGPU must already be selected. Moreover, this can increase the required analysis time.

In this paper, we focus on the power estimation of CNNs on GPGPUs that is one of the most popular ML algorithms in automated manufacturing. We presented a novel approach, enabling designers to predict the power consumption of a given CNN even with small training datasets in the early design phases. The proposed approach takes advantage of a hybrid technique where static analysis is used for features extraction and the *K-Nearest Neighbor* (K-NN) regression analysis is utilized for power estimation model generation. The static analysis for features extraction is performed on *Parallel Thread Execution* (PTX) code (which is generated at compile time) of CNNs, GPGPUs' architectural information, and CNNs topology. To create the power estimation model, we use K-NN which is a non-parametric clustering approach based on distances between data points in a latent space. K-NN can also perform regressions, the output is the average of the values of k nearest neighbors.

Experimental results illustrate the effectiveness of our approach in estimating the power consumption of CNNs on GPGPUs where up to a *Absolute Percentage Error* of 0.0003% in comparison to the real hardware execution is achieved.

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project VerSys under contract no. 01IW19001, by the Data Science Center of the University of Bremen (DSC@UB), and by the University of Bremen's graduate school *System Design* (SyDe).

978-1-6654-9431-1/22/$31.00 ©2022 IEEE

II. Related Work

The existing solutions to predict the power consumption of CUDA-based applications can be divided into two main categories which are 1) the statistical analysis of the application and devices, e.g. [19], [25], [26], and 2) ML-based methods which use different machine learning algorithms to learn the difficult rules from a dataset and create a predictive model, e.g. [6]–[8], [20], [27]. Based on the literature [17], [24], ML-based methods provide better results in comparison to the statistical analysis and become the predominant technique to predict the power consumption of CUDA-based applications. Hence, in this section, we give an overview of ML-based methods and discuss their features and issues.

Existing ML-based methods use so-called performance counters as features [6]–[8], [19], [20], [27] to perform power consumption estimation. Performance counters can only be collected and measured during run-time. This means, that an application needs to be executed once on a real device to collect the performance counters. Afterward, the predictive model can run the prediction for other devices. Using this kind of methodology can limit the usage in early design phases as a GPGPU must already be selected. Moreover, measuring performance counters requires a special GPGPU and CUDA profiler.

The method in [6] uses tree-based regression to predict the power consumption. It analyzes the GPGPU architecture and measures the power consumption of PTX instructions. However, as the method takes advantage of GPGPUSim, it is limited to a small subset of available PTX instructions.

The method in [21] considers scaling frequencies of GPGPU cores and memory for performance estimation. For power consumption estimation, it takes advantage of the [27] which relies on *Support Vector Machines* (SVM) and performance counters. However, performance counters are only available at run-time and limit this approach to be used in early design steps.

The method in [22] introduces an approach to estimate the performance (run-time) of CPU code before porting it to GPGPU code based on machine learning methods. This makes it possible to decide whether executing on GPGPU gives a performance boost or not. A similar goal is pursued in [23] but the prediction can be already performed on GPGPU. The method uses CPU profile data and machine learning methods to estimate the run-time on GPGPUs. However, both aforementioned methods do not support power consumption prediction. They only consider run-time speed up between CPU and GPU. In contrast, we focus on power consumption estimation on GPGPUs.

PPT-GPU is a scalable GPU performance modeling system [16]. However, it does not support power consumption estimation yet. In [9], a layer-wise estimation approach is illustrated. It focuses on embedded GPUs and only considers platforms of the Nvidia Jetson family. ALOHA [17] presents a statistical platform-aware evaluation method for CNNs execution on heterogeneous Systems. For a given heterogeneous system and CNN, it can provide designers with operations and data transfers and their deployment on computing and communication resources. Moreover, it reports an estimation of latency and energy consumption of the CNN on the platform. However, the method requires an execution model that properly describes the details of the platform and the scheduling of different CNN operators on different platform processing elements, which may not be always available.

Our approach does not rely on any run-time information and uses high-level hardware specifications, PTX code, and CNNs attributes which are available at the early stage of design and development. This can significantly help designers in performing the design space exploration of IoT devices and also speed up this process.

III. ML-based Power Estimation Methodology

In order to estimate the power consumption of CNNs in the early design stage, it is important to focus on the information which is available at this stage. The early available information that even does not rely on CNNs execution on real devices is 1) the GPGPUs' architectural details that are available for the different GPGPUs, 2) the low-level PTX code that can be generated at compile time, and 3) CNN architecture and topology that can be distinguished by its trainable parameters. In contrast to [7], [8], we focus on extracting features from the aforementioned sources for the power estimation and do not rely on the performance counter which is only available at run-time. It means that to collect features for power estimation, the aforementioned methods require at least one execution on a real device. This can limit the usage of such methods in the early design stage as the target GPGPU must already be selected.

A. Methodology Overview

The proposed methodology is illustrated in Fig. 1 which has three main phases: 1) information extraction, 2) training dataset creation, and 3) predictive model generation.

In the first phase, we searched for different CNNs and analyze how they load different components of GPGPUs. We compare the architectural information (e.g. CUDA Cores, Memory, or L2 Cache) which are available for different series of GPGPUs. By this, those GPGPUs' attributes and components which have an impact on performing CNN models are extracted. Next, we compile the CNNs to PTX and analyze the PTX code for each CNN. We extract the instructions which are loaded into the GPGPU and build classes of them. Each class contains the number of instructions in the PTX code for a CNN. We also do a high-level CNN analysis and extract the number of trainable parameters for each CNN.

In the second phase, we build a training dataset where the classified extracted CNN instructions and the GPGPU components (that have an impact on performing CNN models) are considered as inputs and the amount of power consumption for each CNN running on the GPGPU as output. The amount of power consumption for each CNN is measured on three different Nvidia GPGPUs (K80, 1080Ti, and V100S) with the Nvidia-smi tool. Since the K-NN regression algorithm is sensitive to the selected features [13], the right combination of features needs to be detected. Thus, instead of using all extracted data as features, we run several combinations to search for the most relevant features. Having fewer features

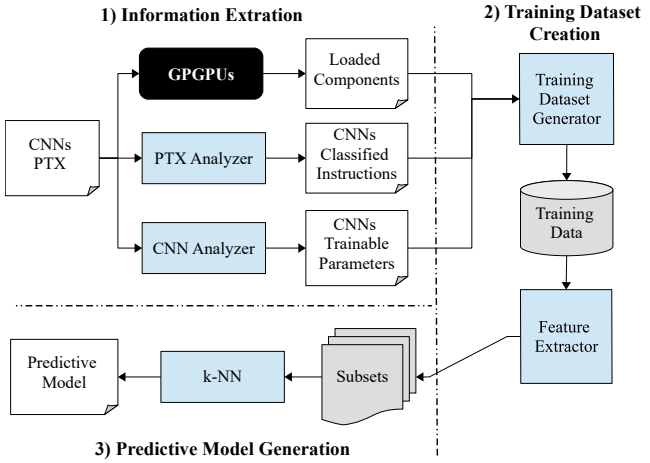

Fig. 1. Overview of the proposed methodology.

```
1 // Generated by LLVM NVPTX Back-End
2 .version 6.0
3 .target sm_70
4 .address_size 64
5 // .globl copy_11
6 .visible .global .align 64 .b8 buffer_for_constant_21
      [1048576];
7 .visible .global .align 64 .b8 buffer_for_constant_34[4]
      = {0, 0, 128, 255};
8 .visible .entry copy_11(
9 .param .u64 copy_11_param_0,
10 .param .u64 copy_11_param_1,
11 .param .u64 copy_11_param_2)
12 .reqntid 72, 1, 1 {
13   .reg .b32  %r<38>;
14   .reg .b64  %rd<55>;
15   ld.param.u64  %rd1, [copy_11_param_0];
16   ld.param.u64  %rd2, [copy_11_param_1];
17   cvta.to.global.u64  %rd3, %rd2;
18   ...}
```

Fig. 2. A part of the PTX file of a CNN model.

leads to simpler models that require shorter training time, reduce the chance of overfitting, and are easier to interpret.

Next in the third phase, we apply the K-NN regression algorithm to the generated training dataset for power estimation model generation. Once the predictive model is trained it can be used to estimate the power consumption for a given CNN on different GPGPU architectures.

B. GPGPUs Architecture Analysis

The power consumption of GPGPUs is affected by many factors. Nvidia lists all the GPGPU's architectural details in [11], [12]. We extract the architectural details for each GPGPU in our experimental setup. The values for the components are transformed into comparable measurement units to make sure the ML method gets the feature for different GPGPUs in the same unit. Moreover, we execute several CNNs on different GPGPUs and monitor the component utilization. This gives us more precise details of which component is affected by CNNs execution and is involved in the power consumption. Furthermore, we also consider GPGPUs architectural attributes which are maximum temperature, transistor size, or maximum power supply. This gives more differentiation between the various GPGPUs.

C. PTX Instructions Analysis

Nvidia provides designers with the CUDA Library to develop GPGPU applications and to write proper code for GPGPUs programming. Nvidia GPUs run so-called kernels. Each kernel is a set of PTX instructions which is generated by compiling the CUDA code with nvcc compiler. The PTX code is a stable low-level programming model and *Instruction Set Architecture* (ISA) for general purpose parallel programming. Fig. 2 shows a part of the PTX code of a given CNN. The PTX code is given to the GPGPU driver and interpreted at run-time [10]. The PTX code contains detailed information of all memory accesses (read or write) as well as computational instructions which will be executed on the GPGPU.

For a given CNN, a static analysis is performed on its corresponding PTX file to count the number of appearances of the instruction that has an impact on the power consumption of the CNN when it runs on a GPGPU. Moreover, we read out the number of threads that are started by the execution of the PTX code. The number for each instruction is multiplied by the number of threads to consider the effect of threads on CNN's power consumption when it runs on the GPGPU. The result of this analysis is stored in the *CNNs Classified Instructions*, including a set of classes (each instruction is associated with a single class) and the number of calls in the PTX code. Hence, not all existing PTX instructions are used for power estimation model generation, instead, we only consider those instructions that appear in the PTX code of CNN benchmarks and have a direct impact on power consumption. As illustrated in Fig. 1 phase 1, the static analysis is performed by *PTX Analyzer* module for several PTX files from different CNN algorithms. The results of this analysis are used to create the training dataset in the next step.

D. CNN Topology Analysis

Every CNN can be characterized based on its architecture and topology. The main feature for this characterization is the CNN's trainable parameters. Trainable parameters are those which changed during the training and are also often refer as weighted connections between each neuron. The number of trainable parameters varies over the different CNNs and gives an additional feature to separate the complexity of different CNNs. The number of trainable parameters of a CNN shows its complexity, meaning it contains more computational operations. Hence, we distinguish a given CNN from others by considering the number of its trainable parameters. This enables us to add more difference into the training dataset for each CNN and to make it distinctive.

As illustrated in Fig. 1–phase 1, these values are extracted by *CNN Analyzer* module for all CNN benchmarks. The results of this analysis are stored in the *CNNs Trainable Parameters* and used to create the training dataset in the next step.

E. Creating Training Dataset and Predictive Model

In order to build the prediction model, it is important to have a robust training dataset. We take advantage of the extracted information from the first phase of the proposed

TABLE I
EXAMPLE OF TRAINING DATASET STRUCTURE USED TO CREATE THE PREDICTIVE MODEL

Observation	CNNs Classified Instructions				GPGPU Components			CNNs Trainable Parameters	Power Consumption (Output)
	Data movement and conversion	Floating-Point	...	Class N	CUDA Cores	...	SM		
CNN_1 on $GPGPU_1$	8	3	...	...	5120	...	80	25549352	$Power_1$
CNN_1 on $GPGPU_2$	8	3	...	...	4352	...	68	25549352	$Power_2$
CNN_1 on $GPGPU_3$	8	3	...	...	3584	...	28	25549352	$Power_3$
CNN_n on $GPGPU_m$	...	...	...	...	...	...	...	...	$Power_{n*m}$

methodology to build the training dataset D based on the following definition.

$$D = \{d_i | d_i = \{y_i, (p_i, c_i, t_i)\}; 1 \leq i \leq n\} \quad (1)$$

Where the parameters p, c, t are considered as inputs (the predictors) of the training dataset and denote the classified extracted CNN instructions, the GPGPU components (that have an impact on performing CNN models), and the CNN trainable parameters, respectively. The parameter y indicates the measured power consumption for each CNN running on the GPGPU and is considered as the output (the response) of the training dataset. Each pair of predictors and the corresponding response is considered as one observation that depicts with parameter d.

In order to give an overview of the training dataset D, Table I demonstrates a part of its structure. The *CNNs Classified Instructions* column lists a part of instruction classes and for each class the number of extracted instructions. Column *GPGPU Components* shows a part of the relevant architectural components. Finally, the *CNNs Trainable Parameters* column depicts the last input (predictor) of the training dataset. The last column shows the power consumption measured by executing the PTX code of each CNN on a real GPGPU. Thus, each row of the table indicates an observation d in the training dataset D where the first three columns are the predictors (or features) while the total power consumption (column *Output*) is the response. The training dataset is split into 70% for the training phase and 30% for the validation phase.

1) Feature Selection: Features that exert little impact on the estimation model should be eliminated to reduce the dimensionality of the training dataset. The reduction in the number of features leads to simpler models that require shorter training time, decrease the chance of overfitting, and are easier to interpret. The K-NN algorithm (that is used in this work for predictive model creation) is sensitive to redundant or irrelevant features [13]. In order to find the best features combination and eliminate irrelevant features, we build different subsets of the main training dataset D where $T_s \subset D$. We start with combinations out of nine different predictors and define the following three types of subsets based on them by considering:

- nine predictors out of all possible ones

$$T_{s_1} = \{d_i | d_i = \{y_i, (c_i, p_i, t_i)\}; 1 \leq i \leq n\} \quad (2)$$

- two subsets where at least one predictor from c must be included:

$$T_{s_2} = \{d_i | d_i = \{y_i, (c_i, p_i)\}; 1 \leq i \leq n\} \quad or$$
$$T_{s_2} = \{d_i | d_i = \{y_i, (c_i, t_i)\}; 1 \leq i \leq n\} \quad (3)$$

- a combination of predictors from p and t where

$$T_{s_3} = \{d_i | d_i = \{y_i, (t_i, p_i)\}; 1 \leq i \leq n\} \quad (4)$$

Since the *GPGPUs' components* predictor has a lot of redundancy with little changes in the training dataset, the statistical automatic feature selection techniques based on variance cannot be used for feature selection due to confusion. The main reason is that the *GPGPUs' components* predictor marks as a low impact predictor by the statistical feature selection. Hence, we select the features manually to solve this issue. The experimental results in Section IV confirm the importance of the *GPGPUs' components* predictor in creating the best predictive model.

2) K-Nearest Neighbors: In order to predict the power consumption of a given CNN, K-NN regression is applied to different subsets T_s of the training dataset D in the definition (1). K-NN is a nonparametric clustering algorithm. It can be used to perform regression prediction by calculating the average value of the k nearest neighbors' values where Y denote the new predicted output and y_i is the output of the ith nearest neighbor.

$$Y = \frac{1}{k} \sum_{i=1}^{k} y_i \quad (5)$$

In order to find the k nearest neighbors, a distance metric is applied to all elements in the training dataset and the new element whose value is to predict. As a consequence, the run-time and complexity is linear scaling with the number of elements in the training dataset. K-NN can be used with different metrics like Euclidean, squared Euclidean City-block, and Chebychev [13]. We use the Euclidean Distance. The Euclidean Distance d between two vectors q and p is defined as follows:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \quad (6)$$

3) Finding the right K: One of the main challenges of applying the K-NN algorithm to a training dataset for prediction is to find the right k. While a large value for k can smoothen the prediction and be assistant with noisy data, a small value of k can corrupt the estimation model. In order to overcome this issue, we perform several experiments of running K-NN for each feature combination, with k in the range starting from one to 20. Since there is no considerable improvement achieved by k values larger than ten, we set the maximum value of k to 20 (see full experiments in Section IV). The K-NN estimates the power consumption by calculating the average power consumption of the k nearest data point in the training dataset.

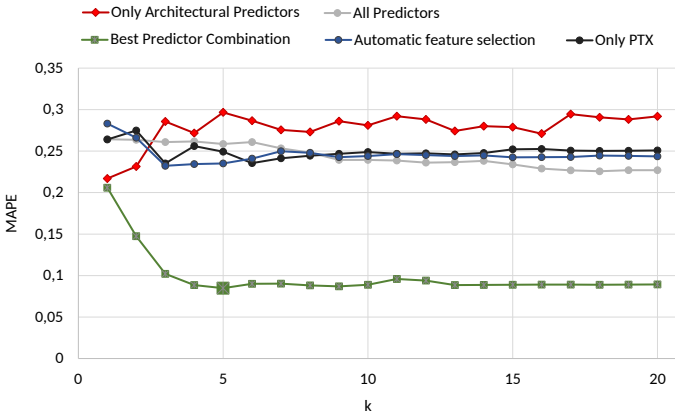

Fig. 3. MAPE value for different k ranging from one to 20.

TABLE II
EXPERIMENTAL RESULTS FOR DIFFERENT INPUT FEATURE COMBINATIONS

Training dataset subsets	k	RMSE	MAPE	R2
Automatic feature selection	3	25.784	0.2322	0.3417
All predictors	18	33.877	0.2270	-0.1934
Only architectural predictors	1	33.2414	0.2169	-0.1613
Only PTX predictors	7	29.8188	0.2412	0.0654
Best predictors combination	5	13.657	0.08849	0.8156

After running the different combinations, we search for the best results in the log file and build the final model based on the best feature combination. Therefore, the obtained predictive model can be used to estimate the power consumption of CUDA-based CNNs on GPGPUs.

IV. EXPERIMENTAL RESULTS

Since the quality of the predictive model is very related to the robust training dataset, five different subsets of the training dataset were created based on the selection of different features (predictors) combination explained in Section III-E1. In order to evaluate the quality of the generated predictive model for each subset of the training dataset, three different standard metrics were applied which are 1) *Root Mean Squared Error* (RMSE), 2) MAPE, and 3) R^2. The lower value for RMSE indicates the better estimation model. The values of MAPE are between zero to one, where the value one stands for an error of 100%. R^2 is a value between zero and one, where one stands for a high correlation between model and data while a negative value indicates no correlation. We also considered the impact of different k values on the predictive model generation when applying the K-NN regression algorithm to each subset of the training dataset. Fig. 3 illustrates the MAPE for different k values for $1 \leq k \leq 20$. As there is no considerable improvement by k values larger than 10, we only performed the experiment by considering the value of k up to 20.

Table II demonstrates the experimental results of our analysis for each subset of the training dataset for the best value of k. Column *Training dataset subsets* lists the features combination that were used to create the best predictive model which are 1) *Automatic feature selection* 2) *All predictors*, 3) *Only GPGPUs' architectural predictors*, 4) *Only PTX predictors*, and 5) *Best predictor combination*. Please note that, for the case of *Automatic feature selection* subset, different automatic feature selection methods were used where the best result is shown in the table that belongs to the F-statistic automatic feature selection method.

TABLE III
DESCRIPTIONS OF THE BEST PREDICTORS (FEATURES) COMBINATION

Features	Brief description
CUDA Cores	Number of CUDA core the GPGPU has
RAM	Amount of GPU memory
Base Frequency	The base frequency of the GPGPU cores
Storage Speed	Bandwidth speed for Storage access
GFLOPS	Number of Floating Point Operations per Second
Memory Clock	Frequency of GPGPU memory
L2 Cache size	Size of L2 Cache of GPGPU in KB
ret	Number of PTX instructions
trainable params	Number of learnable and changeable parameters

As illustrated in Table II, the generated predictive model based on the *Automatic feature selection* subset (using the F-statistic method) has a MAPE of 23.22% which is even worse than the case of *All predictors* where no feature reduction is applied. For the case of *All predictors*, a MAPE of 22.7% was achieved. This also shows that leveraging the automatic feature selection methods (e.g. F-statistic) does not improve the quality of results in this case. By generating a predictive model based on *Only GPGPUs' architectural predictors*, we could improve the quality of estimation to a MAPE of 21.60%. However, for both experiments, the R^2 has a negative score which indicates that the regression model does not fit the data. In the case of generating the predictive model based on *Only PTX predictors*, we could obtain a better result in terms of RMSE and R^2. However, the MAPE of 24.12% indicates the highest error percentage over all Experiments. Based on our experiments, the best power consumption predictive model is obtained by combining the features described in Table III. The power consumption predictive model has an R^2 of 81.56% which indicates a high correlation between the chosen input features and the power consumption. In this case, a MAPE of 8.8% is obtained.

In order to validate the quality of the generated predictive model, we applied the proposed approach on various new CNNs which have not been used in the training phase. Fig. 4 illustrates the predicted (in blue) and the original value of power consumption (in orange) for 12 different CNNs on the Nvidia GTX 1080Ti. As shown in this figure, for some CNNs such as *Vgg19* and *Vgg16*, the prediction is nearly identical to the original value.

In comparison to [3] with an average MAPE of 8.4% for the power consumption predictive model, the proposed approach using K-NN regression achieves almost the same results with only 0.4% accuracy reduction on average. However, in the best case, the proposed approach has better accuracy and a lower absolute percentage error. The best power consumption estimation result belongs to the *Vgg19* CNN with 0.0003% *Absolute Percentage Error* while the estimation result using [3] reports an *Absolute Percentage Error* of 0.73%. Another important point that must be taken into account is that the proposed approach based on K-NN regression only needs nine instead

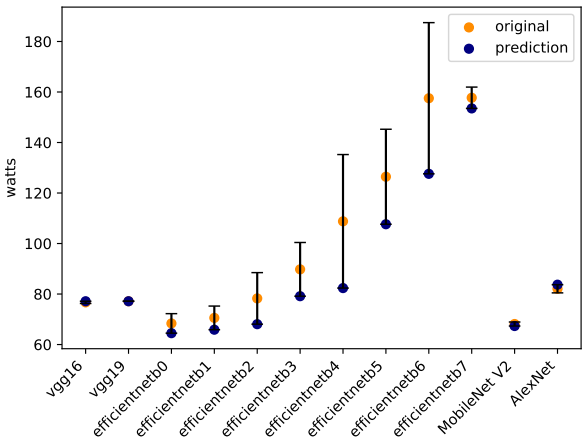

Fig. 4. Scatter plot of the predicted power consumption for different CNNs on the Nvidia GTX 1080 Ti with 8GB Memory.

of 29 features used by [3] to achieve these results. It means that the interpretation of the generated power consumption predictive model using the proposed approach is much easier than [3]. Due to this low number of features, understanding the importance of features is easier for designers which can further help them for design space exploration. Moreover, the experimental results demonstrate that the proposed approach based on the K-NN regression provides designers with an easy-to-interpret and fast power consumption estimation solution, obtaining promising results even with small training data.

V. Conclusion and Future Work

In this paper, we presented a novel power consumption estimation approach for CUDA-based CNNs on GPGPUs based on the nonparametric K-NN regression method. We illustrated how the power consumption of a given CNN on a GPGPU can be estimated by analyzing its PTX code, CNNs' topology, and GPGPUs' architectural Information. We also showed using the K-NN regression, promising results on even small training datasets can be achieved. One of the main usefulness of the proposed approach is for the design space exploration of IoT devices where CNN algorithms need to be implemented as hardware accelerators. In this case, the proposed approach can provide designers with early power consumption (one of the crucial design constraints) estimation of a given CNN model on different GPGPUs at the compilation time. Experimental results on various CNNs demonstrated the advantage of our approach in power consumption estimation.

As future work, we plan to extend our model to any kind of CUDA-based application. Moreover, we are planning to extend our approach in non-functional aspects to a heterogeneous power estimation model that is able to predict both desktop and embedded GPGPUs' power consumption.

References

[1] L. Cui, S. Yang, F. Chen. et al. "A survey on application of machine learning for Internet of Things", In Int. J. Mach. Learn. & Cyber. 9, pp. 1399–1417, 2018

[2] C. A. Metz, M. Goli and R. Drechsler, "Pick the right edge device: torwards power and performance estimation of CUDA-based cnns on GPGPUs", In CoRR abs/2102.02645, 2021

[3] C. A. Metz, M. Goli and R. Drechsler, "Work-in-Progress: Early power estimation of CUDA-based CNNs on GPGPUs", In CODES+ISSS, pp. 29–30, 2021

[4] M. Goli and R. Drechsler, "PREASC: Automatic Portion Resilience Evaluation for Approximating SystemC-based Designs Using Regression Analysis Techniques", In ACM Trans. Design Autom. Electr. Syst. 25(5), pp. 40:1–40:28 , 2020

[5] M. Goli and R. Drechsler, "Resilience Evaluation for Approximating SystemC Designs Using Machine Learning Techniques", In RSP, pp. 97–103 , 2018

[6] J. Chen, B. Li, Y. Zhang, L. Peng and J.-k. Peir, "Statistical GPU power analysis using tree-based methods", In IGCC, pp. 1–6, 2011

[7] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás "Modeling and decoupling the GPU power consumption for cross-domain DVFS", In TPDS, pp. 2494–2506, 2019

[8] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent GPU architectures", In SPDP, pp. 673–686, 2013

[9] M. Lechner and A. Jantsch, "Blackthorn: latency estimation framework for CNNs on embedded Nvidia platforms," In IEEE Access, pp. 110074–110084, 2021

[10] Nvidia, "CUDA TOOLKIT Documentation," {https://docs.nvidia.com/cuda/parallel-thread-execution/index.html}

[11] Nvidia, Nvidia Tesla V100 GPU architecture - The world's most advanced data center GPU, August 2017 available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[12] Nvidia, Nvidia Turing GPU architecture, available: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[13] S. B. Imandoust, and M. Bolandraftar, "Application of k-nearest neighbor (knn) approach for predicting economic events: Theoretical background," In IJERA, pp. 605-610, 2013

[14] M. Fingeroff, Machine learning at the edge: using HLS to optimize power and performance, The Mentor - A Siemens Business, available: https://go.mentor.com/5_3ik

[15] Y. Djenouri, G. Srivastava and J. C. -W. Lin, "Fast and accurate xonvolution neural network for setecting manufacturing data," In IEEE Trans Industr Inform, pp. 2947–2955, 2021

[16] Y. Arafa, A. A. Badawy, G. Chennupati, N. Santhi and S. Eidenbenz, "PPT-GPU: scalable GPU performance modeling," In IEEE Comput Archit Lett, pp. 55–58, 2019

[17] P. Busia, S. Minakova, T. Stefanov, L. Raffo and P. Meloni, "ALOHA: A unified platform-aware evaluation method for CNNs execution on heterogeneous systems at the edge," In IEEE Access, vol. 9, pp. 133289–133308, 2021

[18] Tensorflow, Tensorflow GPU, available: https://www.tensorflow.org/install/gpu

[19] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," In ICGCET, pp. 115–122, 2010

[20] G. Wu, J.L. Greathouse, A. Lyashecksy, N. Jayasena and D. Chiou, "GPGPU performance and power estimation using machine learning," In HPCA, pp. 564–576, 2015

[21] Q. Wang and X. Chu, GPGPU Performance Estimation with core and memory frequency scaling, In TPDS, pp. 2865–2881, 2020

[22] N. Ardalani, C. Lestourgeon, K. Sankaralingam and X. Zhu, Crossarchitecture performance prediction (xapp) using cpu code to predict gpu performance, In MICRO, pp. 725–737, 2015

[23] I. Baldini, S.J. Fink and E. Altman, Predicting gpu performance from cpu runs using machine learning, In SBAC-PAD, pp. 254–261, 2014

[24] L. Braun, S. Nikas, C. Song, V. Heuveline, and H. Fröning. 2021. A simple model for portable and fast prediction of execution time and power consumption of GPU kernels. ACM Trans. Archit. In Code Optim., pp. 1–25, 2020

[25] S. Hong and H. Kim. 2009. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In (ISCA '09), pp. 152–163, 2009

[26] T. C. Carroll and P. W. H. Wong, "An improved abstract GPU model with data transfer," In ICPPW, pp. 113–120, 2017

[27] Q. Wang and X. Chu. 2017. GPGPU power estimation with core and memory frequency scaling. In SIGMETRICS Perform. Eval. Rev., pp. 73–78, 2017