INFER: **IN**ter**F**erence-aware **E**stimation of **R**untime for Concurrent CNN Execution on DPUs

Abstract—**Convolutional Neural Networks (CNNs) are increasingly becoming popular in embedded applications. Hardware designers have proposed numerous accelerators to speed up the execution of CNNs on embedded platforms. Deep Learning Processor Unit (DPU) is one such generic CNN accelerator for Xilinx platforms, available in different configurable sizes and can execute any given CNN. Neural network researchers are also rapidly bringing out newer CNN algorithms with improved performance (typically higher prediction accuracy) with a trade-off in size or energy consumption for embedded applications. In a period of rapid growth in CNN algorithms and availability of multiple configurations of CNN accelerators (like DPU), the design space is fast expanding. To enable quick evaluation of these choices, we propose INFER (INterFerence-aware Estimation of Runtime), a framework to estimate the execution time of any CNN on a given size of DPU without actual implementation. Current FPGA platforms are capable of implementing multiple DPUs whereas many applications consist of multiple sub-tasks with each requiring separate and/or different CNNs. In such scenarios of concurrent use of multiple DPUs on an FPGA, INFER is also capable of estimating the additional time taken for execution due to the sharing of memory bandwidth. Our evaluation on various mixes of 16 standard CNNs and eight configurations of DPU shows that INFER has an average prediction error of 6.6%, which can be useful for design space exploration as well as scheduling in multi-DPU platforms. We demonstrate the applicability of INFER for a real traffic monitoring system which requires the dynamic switching of CNNs at run-time.**

I. Introduction

Owing to their high accuracy in many classification and detection tasks, Convolutional Neural Networks (CNNs) [1] are progressively becoming attractive for embedded systems spanning across autonomous vehicles [2], traffic monitoring [3], assistive devices [4], and many more. These systems execute multiple CNNs concurrently to realize different kinds of classification and detection tasks [2]–[5]. As shown in Fig. 1, sustained effort to improve CNNs has resulted in the availability of a large choice of CNNs for any given task [6]. These CNNs vary in terms of the accuracy that they are able to achieve as well as their compute/memory requirements (Tables II and III). Fig. 1 further shows that CNNs can be implemented on a variety of platforms like CPU, GPU, FPGA, etc. with a trade-off in terms of performance, power, and energy consumption. FPGAs can support high performance-per-watt [7] implementations and are widely used in embedded systems [8].

Many prior works proposed accelerating CNNs or reducing their power consumption using FPGA [8]–[16]. Deep Learning Processor Unit (DPU) [17] from Xilinx, originally developed by DeePhi Tech. and Tsinghua University [9], [11], [18], [19],

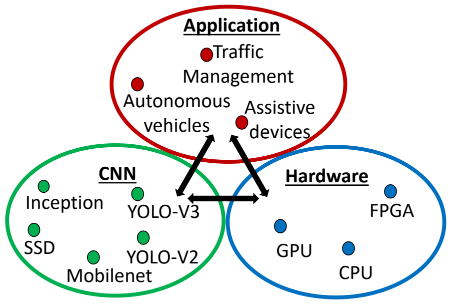

Fig. 1. Growing complexity of applications, CNNs, and hardware

is a generic CNN processor (or accelerator) that supports any CNN. DPU is configurable for various sizes that vary in compute capacity and the amount of FPGA resources required to implement them. Further, the number of DPU instances is also configurable and multiple DPUs could be active concurrently on an FPGA, sharing the memory bandwidth.

Due to their ability to execute a variety of CNNs, DPUs allow dynamic switching of CNN based on the application requirements (at run-time) without any overhead of FPGA reconfiguration. Such dynamic switching is needed for many inference systems like traffic monitoring system [3], MAVI [4], ADAS [2], and f-CNNx [5]. One of the key factors that determine such switching decisions is the execution time of a CNN on the given DPU. With numerous choices of DPU sizes and many newer CNNs evolving, it becomes essential to predict the execution time for making efficient switching decisions at the run-time. To this end, we propose *INFER (INterFerence-aware Estimation of Runtime)*, which is a framework to predict the execution time of a CNN on a given DPU configuration using *easy-to-obtain characteristics* of CNN and DPU.

INFER uses basic CNN characteristics like compute and memory requirements, along with DPU characteristics like number of available processing units and local memory size for predicting the runtime. The estimated value of runtime is further refined to account for memory interference (contention) due to concurrent execution of CNNs on multiple DPUs. INFER has an average error of 6.6% across 16 different standard state-of-the-art CNNs (Table II). INFER is useful both at the design-time for design space exploration to choose the number/size of DPUs and at the run-time for scheduling tasks on DPUs. Being able to successfully estimate the runtime of CNNs using very simple features of CNN and DPU, and interference modeling in a real-life setting are the key highlights of this work. Specifically, this paper makes the following key contributions:

1) Motivating the need for prediction of the runtime of CNNs on generic CNN processors like DPU
2) A framework, INFER, to predict the runtime for a given CNN and DPU size, augmented with interference estimation to account for memory bandwidth sharing
3) Deployment of the proposed framework on various mixes of standard CNNs and different DPU sizes, validated using actual measurements on FPGA board
4) Demonstrating a real traffic monitoring application that uses INFER's predictions to switch CNNs at run-time

II. Related Work

There are several works on designing CNN accelerators for FPGAs [8], [12], [13], [15], [20], [21]. FpgaConvNet [12] and DNNWeaver [13] frameworks can map a variety of CNNs on FPGA. DNNWeaver [13] framework also supports a variety of FPGAs (Intel and Xilinx). However, both these works [12], [13] are useful only at the design-time as a new bitstream needs to be generated for every CNN. FINN [20] framework focuses only on binary neural networks [22]. Haddoc2 [21] is another accelerator where all the CNN weights are stored in FPGA memory itself. This strongly restricts the CNNs that can execute on an FPGA. Unlike all the above works, Xilinx DPU [17] is a generic accelerator that can execute any CNN (which can be changed at run-time using software compilation only) and provides many options to configure the IP at design-time. The presence of such configurability and ability to switch CNNs at the run-time presents newer opportunities for achieving efficiency and forms the motivation for our work.

Prediction models are used across different compute hardwares like CPU [23], GPU [24], and FPGA [25], [26], [27]. Zheng et al. [23] use performance counters to model performance and power of workloads for CPU. O'neal et al. [24] proposed a design-time framework to predict the performance of multiple workloads on GPUs using program counters. Works on estimation for FPGA focus on improving the accuracy of HLS reports as current HLS tools [28] have high error in their prediction. Pyramid [26] and XPPE [27] use machine learning to estimate resource usage and timing of a design for different FPGA types. HLScope+ [25] estimates the number of cycles considering memory interference when multiple PEs are connected. All these works primarily focus on either predicting the performance of only a single module or use internal design details for prediction. In contrast, our prediction framework addresses the concurrent execution of multiple IP blocks and uses only the publicly available information for building the model. Moreover, existing works perform prediction at design-time for a known workload, whereas our framework is useful at both design-time and run-time and adapts well to newer workloads (CNNs).

ProxylessNAS [29] is orthogonal to our work as it considers device-level hardware options like CPU/GPU/Mobile while we consider options (DPU size) within a device (FPGA). Ferianc et al. [30] uses Gaussian process based modeling for layer-by-layer estimation of runtime and uses the off-chip memory bandwidth as a feature. They use fixed hardware architecture and only 3 CNNs for their evaluation. Their reported results have a much higher mean average error than ours. Further, both ProxylessNAS [29] and Ferianc et al. [30] consider a single processor system where the effect of interference due to multiple CNNs running concurrently is not applicable. Qiu et al. [19] introduce an analytical model using DPU's internal architecture to predict the runtime of a CNN on a DPU. In contrast, INFER uses machine learning with only the information that is available publicly and performs significantly better. Moreover, their model considers a single DPU, while we also account for memory bandwidth sharing due to multiple DPUs executing concurrently.

III. Background on CNN and DPU

A. CNN: Convolutional Neural Network

A CNN consists of many cascaded layers of different types like convolution layers, fully connected layers, etc. Each layer can be characterized by various parameters like input size, output size, kernel dimensions, number of input and output channels, etc. which can be combined to form two important attributes of a layer – (i) *Computation load*: Number of MAC (multiply and accumulate) operations in a layer, and (ii) *Memory/data requirement*: total data (input, kernel weights, and output) accessed by the DPU from the main memory.

B. DPU: Deep Learning Processor Unit

A DPU [17] is a generic CNN processor (accelerator) for Xilinx platforms which can be programmed to execute any CNN. It performs layer by layer processing of CNN, which is invoked by a host CPU (Fig. 3). DPU is available in various sizes like B4096, B3136, or B512, where the suffix number represents the processing capacity in terms of number of concurrent MAC operations. A DPU with higher processing capacity requires more FPGA resources. A DPU accesses data from main memory (DRAM) through an AXI bus [31].

A compiler for DPU [18] converts a given CNN description file into a DPU instruction code having details about the number of layers, type of each layer, parameter size, kernel size, scheduling of load/store operations, etc. DPU fetches the required input data and weights from the main memory into a local BRAM and updates the result into an output buffer (BRAM). Once the output buffer (result) is ready, it is written into an appropriate location in the memory [9] and an interrupt informs the host CPU about the completion.

IV. Proposed Approach for Runtime Estimation

In this section, we describe the INFER framework that uses a regression model to predict the runtime of any CNN on a DPU. As shown in Fig. 2, our framework is divided into two modules. The first module predicts the runtime for a single DPU. The second module estimates the memory interference to predict the increase in runtime when different CNNs are executing concurrently on multiple DPUs.

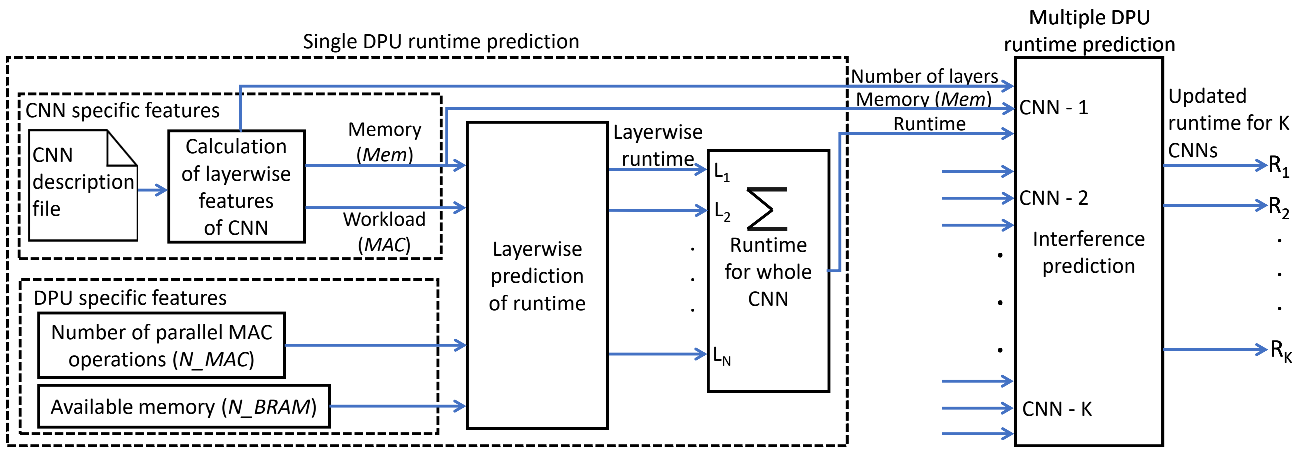
Single DPU runtime prediction
CNN specific features
CNN description file
Calculation of layerwise features of CNN
Memory (Mem)
Workload (MAC)
DPU specific features
Number of parallel MAC operations (N_MAC)
Available memory (N_BRAM)
Layerwise prediction of runtime
Layerwise runtime
L1
L2
LN
Runtime for whole CNN
Number of layers
Memory (Mem)
Runtime
Multiple DPU runtime prediction
CNN - 1
CNN - 2
CNN - K
Interference prediction
Updated runtime for K CNNs
R1
R2
RK

Fig. 2. Block diagram of runtime estimation framework

A. Runtime Estimator for Single DPU

We predict the runtime of individual layers of a CNN, which are then combined (added) to get the predicted runtime for the whole CNN. For any prediction task, it is important to identify the relevant features to build the model. Since we are predicting the runtime of CNNs on different DPU sizes, the features were identified in two categories: Hardware (DPU) related features and CNN specific features.

With i representing the layer number of the given CNN and j representing the DPU size to be used, we use the following notations to define the features:

- MAC_i = Number of MAC operations in the layer
- Mem_i = Amount of data (in MB) required by the layer
- N_MAC_j = Number of parallel MAC operations supported by the DPU
- N_BRAM_j = Available BRAM (in MB) in the DPU

CNN specific features: MAC_i contributes to the time taken for computation of MAC operations in each layer. Mem_i contributes to the time taken for data transfer between local memory of DPU (BRAM) and main memory (DRAM). DPU contains two separate AXI buses for accessing main memory such that reading of weights and input data can happen concurrently. Further, each AXI bus contains concurrent read and write paths [31], which enables the writing of output data to overlap with reading. Therefore, we consider the maximum of the data size of weights, input data, and output data as the memory requirement (Mem_i) of a CNN layer. We considered various other features like taking the size of weights, input, and output data separately or taking the sum of the three parameters instead of the maximum. We observed that the average error in prediction of the runtime is higher with the use of these features (7.4% error in both cases) compared to using the maximum of the three parameters as a feature (6.6% error). Even intuitively, maximum seems appropriate due to concurrent access that is possible due to multiple buses.

Further, some CNNs like mobilenet_v2 [32] and ssd_mobilenetv2 (see Table II for details) have both depthwise and pointwise convolution operation [33]. The profile obtained by executing these CNNs on DPU indicates that DPU merges the pointwise convolution with depthwise convolution causing changes in the compute and data access behavior of these layers. Therefore, we add a binary flag $Merge_i$ to capture such merge behavior. Use of $Merge_i$ as a feature reduces the error in runtime prediction from 9.9% to 3.2% for mobilenet_v2 and from 16.3% to 15.5% for ssd_mobilenetv2. In summary, $\langle MAC_i, Mem_i, Merge_i\rangle$ are CNN specific features used in our prediction model.

CNNs can have layers with the same number of MAC operations and data requirements but with significantly different layer architecture (i.e., different number of input and output channels, filter sizes, and feature map dimensions). We experimentally observed that the difference in runtime for two such layers with different layer architectures is small ($\sim$0.15 ms) except for a very few outliers. Since we are interested in predicting the total execution time of a CNN rather than layer by layer execution time, the overall error still remains within the acceptable range. Thus, we choose a small number of simple features to predict runtime rather than a detailed set of features of a CNN as well as DPU/FPGA architecture.

Hardware related features: As discussed in section III-B, different DPUs differ in number of BRAMs and the supported parallel MAC operations. To generalize the prediction model over different DPU configurations, we use $\langle N\_MAC_j, N\_BRAM_j\rangle$ as hardware specific features.

The runtime of layer i of CNN on DPU configuration j can now be written as:

$$Time(i,j) = fn\left(\left(\frac{MAC_i}{N\_MAC_j}\right), \left(\frac{Mem_i}{N\_BRAM_j}\right), Merge_i\right)$$

The runtime of any layer has two components – computation time and data access time. The computation time is dependent on $\frac{MAC_i}{N\_MAC_j}$. The data access time from external memory is dependent on the amount of data to be transferred, on-chip memory size, and memory bandwidth. The memory bandwidth is fixed for a given FPGA board and there is no interference when executing a single CNN on a DPU. Since we first calculate the execution time of a single CNN only, $\frac{Mem_i}{N\_BRAM_j}$ can be considered as a suitable feature representing the data access time.

TABLE I
ESTIMATION ERROR FOR DIFFERENT TECHNIQUES ON SINGLE DPU

Prediction Technique	Mean	Max.	Median
Random forest	6.6 %	23.7 %	4.8 %
Decision tree	7.4 %	30.0 %	5.9 %
Linear regression (first order)	7.7 %	32.4 %	5.2 %
Polynomial model (second order)	8.0 %	23.1 %	6.7 %

Our prediction model uses these features to support various CNN types (classification or detection) and DPU configurations. These features are simple and easy-to-obtain, which makes the predictor usable in practice. The CNN specific features can be obtained from CNN description file and the DPU related features are available from the hardware specifications of DPU without any need for proprietary information. We use ZCU102 board [34] to measure the runtime of different CNN layers for various DPU sizes (details in Section V-A) and perform training and runtime prediction using four standard regression techniques (Table I). Our results indicate random forest regressor to provide the lowest mean and median error (and low maximum error). The random forest is a non-linear predictor, formed of multiple uncorrelated decisions trees, and provides better prediction than any individual tree [35]. Prior works [24], [26] have also found random forest to provide the best prediction. Therefore, we use random forest for runtime prediction of CNNs on a single DPU.

B. Runtime Estimator for Multiple DPUs

Multiple DPUs can be implemented together on an FPGA to enable concurrent CNN execution (each CNN can however use only one DPU). Each DPU has independent compute resources and local BRAMs, and the main memory is also of much larger size than needed by CNNs. Therefore, concurrent CNN execution would not see performance degradation due to compute or memory size requirements. However, as shown in Fig. 3, DPUs would experience interference due to sharing of main memory bandwidth. Fig. 4 shows the increase in runtime of various CNNs, measured on ZCU102 board, when executing concurrently with other CNNs on separate DPUs (CNN description in Table II). Despite a sophisticated memory controller being available (Fig. 3) for efficiently scheduling concurrent requests to DRAM [36], we observe up to 52% increase in CNN runtime due to interference; which motivates the need to account for it during runtime estimation.

Since memory bandwidth sharing is the cause for runtime deterioration, the bandwidth requirement of a CNN as well as the bandwidth requirement of other CNNs executing concurrently would determine the amount of interference. The interference behavior is not a linear function of the bandwidth of different CNNs because it depends on how much the high bandwidth access requirements of different CNNs overlap in time. Thus we explored a few regression models to estimate the increase in runtime due to such interference. A third-order polynomial shows ∼1% higher accuracy than a second-order model for training cases but overfits and causes lower accuracy on unseen cases. Also, second-order predictor is a simpler model for use. Therefore, we use a **second order** polynomial

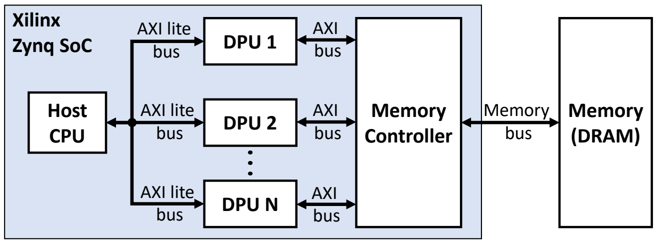

Fig. 3. System level view of DPU and memory interface

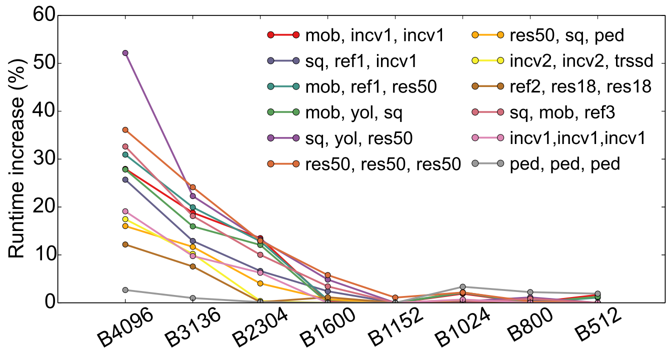

Fig. 4. Runtime increase due to memory contention for various CNNs

regression model for interference prediction. With $Data$ being the total data requirement (in MB) of the CNN and $Time$ being its total execution time (in ms), the average bandwidth (BW) of each concurrently executing CNN is defined as:

$$BW = 1000 \times \frac{Data}{Time} \quad (1)$$

Further, we also observed that CNNs with smaller data requirement per layer (e.g., *sq, mob, incv3*) suffer from larger interference. For such CNNs, we understand that the overhead of fetching instruction code for every layer becomes a significant fraction causing them to experience higher interference. To account for this behavior, we define a new attribute $MBpl = \frac{Data}{N_{layer}}$ (MegaBytes per layer), where N_{layer} is the total number of layers in the CNN. We consider $\frac{1}{MBpl}$ of each concurrent CNN as a feature into our model. Although different layers in a CNN can have varying data requirements, $MBpl$ is sufficient since we measure only the average effect of interference.

To compute the bandwidth of each CNN, the single DPU runtime measured on ZCU102 board is used as $Time$ in Eq. 1. Using the measured increase in runtime along with corresponding BW and $MBpl$ features, we train a second order polynomial regression model to estimate the percentage increase in runtime. We compare the behavior of the second order polynomial model to other regression models like decision tree, random forest, and third order polynomial. We observe the second order polynomial model to provide the lowest error. The estimated increase in runtime due to interference is multiplied with the runtime predicted for the single DPU to obtain the final runtime.

$Time$ used in Eq. 1 to calculate the bandwidth of a CNN is the runtime of the CNN for a given DPU size. Since $Time$ already comprehends the effect of DPU size within it, we believe that DPU size might be redundant as a feature for prediction. We experimentally observed that including DPU size as an additional feature reduces the maximum error by about 1%, but increases the mean error by about 0.5%. We chose to improve the mean error and hence excluded DPU size from the feature set, but including it could be also an acceptable design decision.

V. Experiments and Results

A. Experimental Setup and Measurement Methodology

We evaluate our proposed framework using 16 standard CNNs with different characteristics as shown in Table II. Eight CNNs (*TRAIN* type) are used for training purposes and the remaining eight CNNs (*TEST* type) along with *TRAIN* set are used to validate the trained predictor. Each CNN consists of a large number of layers, thereby forming a rich training dataset (1042 data points) for single DPU prediction. These CNNs also form a large number of combinations used to create the dataset for multiple DPU prediction (837 training points). We consider eight standard sizes of DPU [17] named B4096, B3136, B2304, B1600, B1152, B1024, B800, B512 (see Sec. III-B for description). We use B4096 (largest), B512 (smallest), and B2304 (mid-sized) DPUs for training purposes whereas validation is performed on all DPU sizes.

We create multiple hardware designs, each with three instances of a particular DPU size,[1] and implement them on Xilinx Zynq UltraScale+ (ZCU102) board [34]. Due to the generic nature of DPUs, we can execute any CNN from Table II on any size of DPU. To measure the standalone execution time for a CNN, we execute it as a single thread on the ZCU102 board. We repeat this experiment for each combination of CNN and DPU. We enable the *profile* mode of DPU which helps to record the execution time for individual layers as well as the complete CNN. To account for variability during the measurements, we consider the average value of 50 readings as the measured execution time.

For analyzing the increase in runtime due to memory interference for concurrent DPUs, we create all possible combinations of three CNNs from each of the TRAIN and TEST category. The CNNs in each combination are executed as three concurrent threads with each thread repeating until every thread has been executed for atleast 50 times. Since different CNNs have different execution times, the faster running CNNs get executed more often. For each CNN, we compute the average of measured runtimes and compare it with its standalone runtime to obtain the increase in runtime of the particular CNN for each combination. Please note that while the standalone execution time study requires recording the runtime for each layer, the interference study considers the execution time of the CNN as a whole.

The number of layers, number of MAC operations, and the data requirement (shown in Table II) are static information for a particular CNN (and its layers) and are extracted from the CNN description file for the corresponding CNN.

[1]Xilinx tools currently allow a maximum of 3 DPUs only of the same configuration and not different configurations. We believe such restriction may be removed soon.

TABLE II
Characteristics of various CNNs used in our experiments

CNN Name	#MAC Operations ($x10^6$)	Data Required (MB)	# Layers	Type
squeezenet (*sq*)	775.50	4.88	26	
mobilenet_v2 (*mob*)	601.55	9.86	36	
inception_v1 (*incv1*)	3165.34	14.62	59	T
ssd_pedestrian (*ped*)	5891.81	17.50	35	R
resnet50 (*res50*)	7715.95	51.98	55	A
refinedet_1 (*ref1*)	25196.19	41.79	48	I
vgg16 (*vgg*)	30940.53	156.78	16	N
yolo_v3 (*yol*)	60569.32	156.95	83	
resnet18 (*res18*)	3653.84	17.08	23	
inception_v2 (*incv2*)	4037.70	22.13	79	
refinedet_3 (*ref3*)	5084.69	20.04	48	T
ssd_adas (*adassd*)	6284.15	19.80	35	E
ssd_mobilenetv2 (*mossd*)	6537.10	43.74	50	S
ssd_traffic (*trssd*)	11670.46	21.12	35	T
refinedet_2 (*ref2*)	10096.35	25.43	48	
inception_v3 (*incv3*)	11426.43	49.53	103	

B. Results and Discussion

We present quantitative results of single and multiple DPU runtime prediction (with different number of CNNs).

1) Single DPU runtime prediction: Fig. 5 presents the distribution of error in runtime prediction for different layers across all CNNs and DPU sizes. High percentage errors are concentrated towards layers with small runtime (0.01-1.0 ms) as minor prediction errors become larger percentage when base value is small. The figure also shows the distribution of absolute error in predicting the runtime for various CNN layers. We observe the absolute error to be mostly below 2 ms.

Fig. 6 shows the runtime for different CNNs by combining runtime of their layers for B4096 DPU. We observe that the execution time for different CNNs vary significantly, ranging from $\sim$1.5 ms to $\sim$65 ms for B4096 DPU, and goes up to $\sim$405 ms for B512 DPU. Qiu et al. [19] uses internal design details and propose a model to estimate the theoretical runtime of *vgg16* network on a particular DPU. Their estimation has an error of $\sim$30% compared to 9% in our model.

Fig. 7 shows the prediction error for all CNNs for different DPU sizes. The overall average error across all DPU sizes is 6.6% and the maximum error is 23.7%. We also observe the average as well as median error to be significantly lower than the maximum error due to a small number of outliers which significantly increase the maximum error.

INFER is also useful to predict the runtime for custom CNNs created by modifying any standard CNN by changing the number of filters or adding a new layer. Apart from the 16 standard CNNs mentioned in Table II, we created six custom CNNs by modifying these standard CNNs. We modified yolo_v3 CNN for 3 new image sizes (608x608, 412x412 and 320X320) as well as different counts of filters (for 2-3 different layers). We also tried other variants of yolo like tiny_yolo_v3 and yolo_v2 and combination of other CNNs like ssd_vgg16. We observe a 6% average error in prediction of runtime for

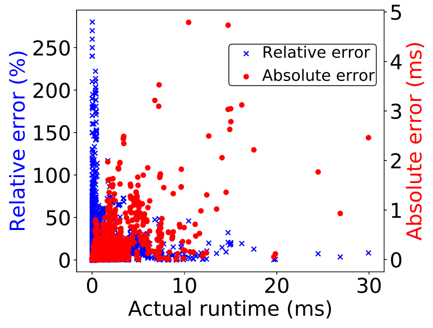

Fig. 5. Error distribution for different layers of various CNNs (for single DPU prediction)

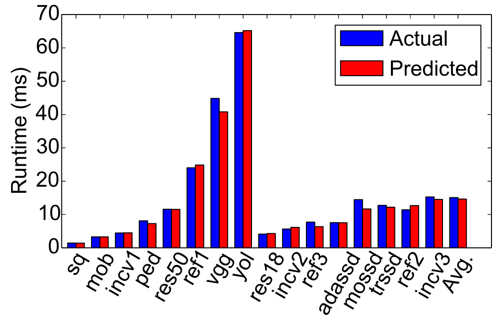

Fig. 6. Actual versus predicted runtime for various CNNs for B4096 (for single DPU prediction)

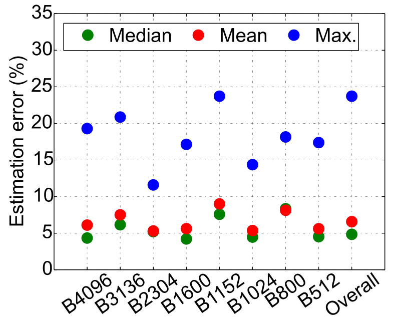

Fig. 7. Prediction error for different DPU sizes (for single DPU prediction)

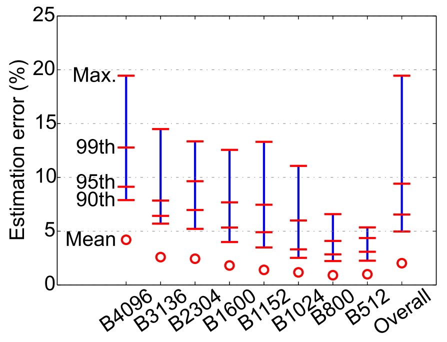

Fig. 8. Interference estimation error for different DPU sizes

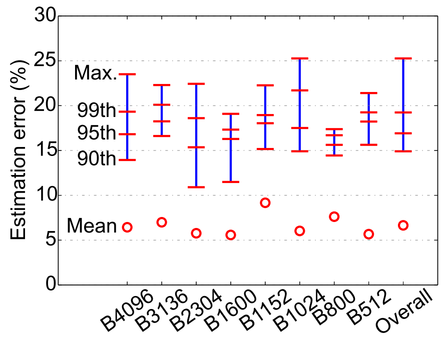

Fig. 9. Final estimation error (single DPU along with interference model) for different DPU sizes

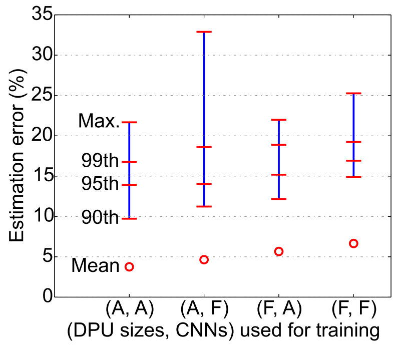

Fig. 10. Final estimation error for various training sets. A: All, F: Few (only *TRAIN* set)

these custom CNNs, which establishes that INFER can be used for a larger range of CNNs.

2) Estimating the effect of memory interference: Fig. 8 presents the error in estimating the effect of interference (increase in the CNN's execution time due to co-execution on multiple DPUs). We observe that the maximum value of mean estimation error is 4.2% (for B4096), and it decreases with decreasing size of DPU. The reduction in error is because of the reduction in the overall range of interference for smaller DPU sizes (Fig. 4), making it more predictable. We also observe the 90th, 95th, 99th percentiles and maximum error to be considerably apart, indicating that only a few outliers degrade the prediction performance. Therefore, it is important to consider these percentile errors during the evaluation of estimation models. Overall, across DPU sizes, the 99th percentile error is 9.4% and the mean error is only 2%.

We study the effect of including *MBpl* (MegaBytes per layer of a CNN) as a feature in the estimation model. Our results indicate that use of *MBpl* reduces the maximum error from 39.5% to 19.4%, 99th percentile error from 16.0% to 9.4%, and the mean error from 2.8% to 2%. This clearly justifies the use of *MBpl* for interference estimation for CNNs executing concurrently on DPUs.

3) Multiple DPU runtime estimation considering interference: Now, we study the error in estimating the final runtime of three CNNs executing concurrently on three different DPUs. The execution time predicted for a given CNN by our single DPU predictor is multiplied by the estimated interference due to other CNNs to predict the final execution time in the presence of memory interference. The predicted execution time is compared with the actual time measured on the ZCU102 board to obtain the prediction error. Fig. 9 shows the error for different combinations of CNNs for different DPU sizes. We observe an overall mean error of **6.6%** and maximum error of **25.3%**. The overall 90th and 99th percentile errors are within 15% and 20%, respectively.

INFER executes on ARM CPU core available on Xilinx Zynq chips (Host CPU from Fig. 3) and takes ~2.4 ms on ZCU102 board for the CNN with largest number of layers (*incv3*). The measured time is negligible in comparison to the much larger period (few seconds to hours) at which switching of CNNs might be required in an application. Moreover, the prediction happens on the CPU core and does not affect the execution of CNNs on DPUs. Therefore, our prediction model is suited to be used at both design-time as well as run-time.

4) Sensitivity to training set size: The presented results have considered the prediction model to be trained using only the *TRAIN* set of CNNs (Table II) and DPU sizes (B4096, B2304, and B512). We show the effect of including more DPU sizes and/or CNNs in the training set. Fig. 10 shows the estimation error across all CNNs and DPU sizes for four cases when all CNNs/DPU sizes are considered for training versus only a few (the *TRAIN* set) are used for training. We observe that the prediction accuracy improves by including more DPU sizes or CNNs into the training set. The mean error and the 90th percentile error reduces by about 5% and 3%, respectively, when including all CNNs and DPU sizes into the training set. However, the maximum error shows an improvement of only 3%. For systems where the set of CNNs and DPU sizes to be used are known to be limited, we could improve the prediction accuracy by training with all CNNs and DPU sizes. However, due to rapid evolution of CNNs and DPUs being developed for variety of sizes, a prediction model trained on limited set is a reasonable choice.

5) Applicability to other CNN counts: Now, we study the usability of the proposed predictor for a smaller (two) and larger (four) number of CNNs. For the former case, we execute various combinations of two CNNs on two separate DPUs on ZCU102 board and measure the increase in runtime, averaged over atleast 50 measurements for each CNN. For the purpose of prediction, we set the bandwidth and *MBpl* of the third CNN as 0. Similarly, we generate various combinations of four CNNs and execute first two on two separate DPUs and the other two alternatively on the third DPU. We measure the increase in runtime for each of these CNNs. We set the bandwidth and *MBpl* values of the third CNN as the average of the two CNNs executing on the third DPU. CNNs executing on the same DPU do not face any interference from each other.

We use the prediction model developed so far, without any additional training, to predict the execution time for two and four CNN scenarios (1040 and 6636 test points, respectively). We obtain a mean error of 7.1% and 6.9%, and a maximum error of 25.8% and 24.7% for two and four CNN scenarios, respectively. These error rates are of similar order as presented earlier for three CNNs (Fig. 9), indicating that the proposed predictor generalizes well for different count of CNNs.

Having explained our proposed runtime estimation framework (INFER), the next section presents the example of a real system where one can use INFER to make switching decisions at run-time.

VI. Application: Traffic Monitoring System

We demonstrate the utility of INFER in a traffic monitoring application [3], which benefits from dynamic switching of CNNs at runtime, using INFER's execution time predictions.

A. The Traffic Monitoring Application

Our application involves traffic monitoring system for a typical 4-way intersection, with two cameras to monitor each approach. The two cameras are placed in opposite directions to capture the full view of traffic when it moves (green light) and when it stops (red light). A total of 8 cameras are placed at different positions on the road, as shown in Fig. 11.

We run two detection tasks, one on the feed from each camera – (i) *detecting speed violation* when it is a green signal and (ii) *detecting signal violation* when it is a red signal. The CNN task (object detection) is same in both cases which detects the vehicles and their position in the frame. For speed violation detection, vehicle is detected at time t1 and t2, and speed is calculated as $Speed = \frac{Distance}{(t2-t1)}$. If speed exceeds the permitted limit, we consider it as a speed violation. On the other hand, if vehicle is detected on or beyond the

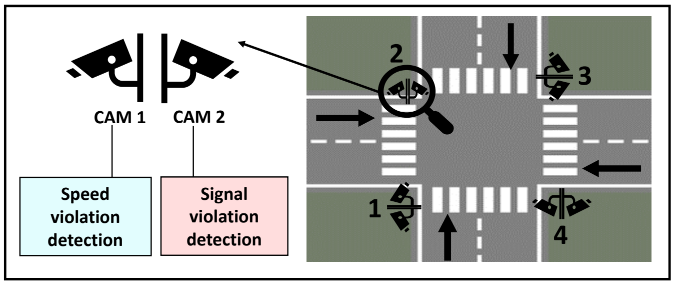

Fig. 11. Traffic monitoring system

zebra crossing at red light, it is considered a signal violation. The CNN task runs on DPU whereas this post processing to identify signal violation or speed violation runs on the CPU.

We experimentally observe that based on number of vehicles on the road, different CNNs show different vehicle detection accuracies (Table III). The number of vehicles (traffic density) can vary at different times of the day – like low density during early morning time, high density during peak office hours, and moderate density otherwise. Detecting this density at run-time is a computationally simple task that can run on CPU, without using CNN, based on background subtraction and optical flow. Based on measured density, CNNs can be dynamically switched using INFER's prediction of runtime.

Choice of CNN – (i) *High FPS CNN during low traffic density*: Speed violations typically occur at low traffic density. To detect a vehicle at both times t2 and t1, processing at high FPS (frames per second) is needed, else a vehicle will escape the frame before the CNN detects it. (ii) *High accuracy CNN during high traffic density*: Signal violation occurs in high traffic density when the signal is red. In that chaotic scene, detecting the violating vehicle (typically the small motorbikes in non-laned traffic in developing countries) will need a very accurate CNN. It is acceptable if this CNN is slower, as at high density, the vehicles move slowly and can be caught even at low FPS.

Considering this target application, 8 CNNs need to run continuously on our DPUs. Each CNN receives an image from 1 camera (total 8 cameras placed). Out of 8 images, the signal violation detection task runs on 4 images (from 4 cameras, namely CAM2 in Fig. 11 looking at outgoing traffic) and speed violation detection task runs on other 4 images (from other 4 cameras, namely CAM1 in Fig. 11 looking at incoming traffic).

B. Experimental Setup to Demonstrate INFER's Use

We train different CNNs using the non-laned, road traffic dataset of developing regions [3] and measure the inference accuracy of each CNN as mean average precision (MAP), as shown in Table III. We also show other metrics (precision and recall) for each CNN. *Precision* is the fraction of correctly identified objects among total predictions made by the CNN while *recall* is the fraction of correctly identified objects of interest among its total count. MAP is the mean value of precision when varying recall between from 0 to 1.

The table shows three categories for the traffic density that are high, moderate and low. Taking 600 images from the dataset, we manually classify them as high (vehicles more

TABLE III
RUNTIME AND ACCURACY TRADE-OFFS FOR DIFFERENT CNNS

CNN	**Different traffic densities**									**Runtime for different DPU sizes (ms)**	
	High			**Moderate**			**Low**			**Small**	**Large**
	MAP	**Precision**	**Recall**	**MAP**	**Precision**	**Recall**	**MAP**	**Precision**	**Recall**		
yolo_v2	77.83	0.78	0.73	88.62	0.85	0.89	91.50	0.76	0.93	231	31
yolo_v3-320	76.46	0.91	0.76	86.62	0.95	0.84	87.16	0.90	0.77	247	38
yolo_v3-416	93.06	0.88	0.94	94.15	0.91	0.96	95.33	0.83	0.96	428	61
yolo_v3-608	94.22	0.91	0.94	94.46	0.94	0.96	95.03	0.91	0.96	873	123

than 15), moderate (between 6 and 15), or low (less than or equal to 6) density by looking at the number of vehicles in each image. There are 200 images in each category.

The table also shows the execution time of different CNNs measured for two different sizes of DPU on Xilinx Zynq UltraScale+ ZCU102 board [34]. We take 2 DPUs for this example that are DPU 4096 (largest) and DPU 512 (smallest). We have listed 4 networks in the table. We also trained VGG-16_SSD on the traffic dataset. Since it was less accurate than other networks, we did not include it in the table.

C. Observations Highlighting INFER's Importance

1) INFER's use when only 8 CNNs run: As seen from Table III, different CNNs have different accuracy in different traffic densities. At low densities, all CNNs are accurate. But at high densities, especially in developing countries where vehicles do not follow lanes making the scene chaotic with motorbikes, cars, buses, and trucks all standing together, larger CNNs outperform smaller CNNs. For high traffic density, more accurate CNN is required. Both yolo_v3-416 and yolo_v3-608 have similar accuracy (MAP) for high traffic density. Since yolo_v3-608 has higher precision, it is chosen. Precision is important in rule violation detection, as false positives can lead to fining of drivers who are following the rules. For low traffic density, CNN with high FPS is required. The choice is between yolo_v2 and yolo_v3-320 (lower runtime compared to other CNNs). Both have similar accuracy and FPS but yolo_v3-320 has higher precision. Hence, yolo_v3-320 is chosen. The runtime predicted by INFER for different DPU sizes, given any CNN, is crucial to make dynamic choices of CNN as described above, based on the current traffic density. Table III only shows the runtime when a single CNN is running on a single DPU. But INFER can also predict the time when multiple CNNs run together (taking interference into consideration).

2) INFER's use when additional critical tasks run: There can be some other critical tasks that gets scheduled occasionally, like routing an ambulance or a fire engine, detecting and tracking a stolen vehicle. Since priority is given to the critical task, remaining time will be available to run the 8 CNNs. The network with highest accuracy feasible within the remaining time budget should be dynamically selected. Again INFER's runtime predictions are crucial for these dynamic decisions.

3) How much overhead does INFER introduce?: If we use INFER, the time taken to evaluate CNN choice would be 9.6 ms (for 4 choices of CNNs in Table III). This time is very small compared to time taken for running each of the 4 CNNs, which would take more than 1.8 sec (in case of small DPU) or 250 ms (in case of large DPU). Thus in comparison to the CNN runtime, INFER's prediction time is very small.

4) Design overhead in absence of INFER: CNNs are evolving fast and DPUs have numerous choices. Since INFER can predict execution time without compiling CNNs or generating bitfiles, it is useful at design-time to choose proper size FPGA and identify suitable CNNs and DPUs as per cost-accuracy trade-off analysis. Doing the same using measurements would take considerable time and effort. For example, if we consider 3 FPGA boards, 8 CNNs and 8 DPU configurations, total of 192 bitfiles are generated to evaluate the choices. However, INFER can do such evaluation without generating bitfiles. After design-time decisions, execution times could be stored in a table to be used at run-time. At run-time, newer CNNs evolved after system deployment could be evaluated by executing on DPU and added to table, but such evaluation would disturb the ongoing execution. Use of INFER can avoid such disturbance.

VII. CONCLUSION AND FUTURE WORK

We presented the motivation for predicting the execution time of Convolutional Neural Networks (CNNs) on generic CNN accelerators like DPU. Subsequently, we developed a framework to predict the runtime of a CNN on a given size of DPU. For systems that use multiple DPUs to concurrently execute CNNs, we developed an interference estimation model that is used to refine the predicted runtime to account for interference due to shared memory bandwidth. We evaluated our prediction framework (INFER) using various mixes of 16 standard CNNs and 8 different sizes of DPU. Using only 8 CNNs and 3 DPU sizes for training, we obtain an average prediction error of 6.6% across the entire set of CNNs and DPU sizes. Specifically for the *vgg16* network, INFER has an estimation error of 9% compared to $\sim$30% in prior works.

INFER uses basic CNN and DPU characteristics which are easy to obtain and are publicly available. We show that INFER generalizes to different number of concurrent CNNs and can be used at both design-time and at run-time. We also show the use-case of INFER for a real traffic monitoring system that require switching the CNNs dynamically. In the future, we would like to incorporate energy estimation into INFER to enable energy-aware decision making under performance constraints. We also plan to integrate INFER in a scheduler so that given the resources one can choose an appropriate CNN that gives the best performance while still being able to meet the time deadlines.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[2] J. Peng, L. Tian, X. Jia, H. Guo, Y. Xu, D. Xie, H. Luo, Y. Shan, and Y. Wang, "Multi-task ADAS system on FPGA," in *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, March 2019, pp. 171–174.

[3] M. S. Chauhan, A. Singh, M. Khemka, A. Prateek, and R. Sen, "Embedded CNN based vehicle classification and counting in non-laned road traffic," in *International Conference on Information and Communication Technologies and Development (ICTD)*, 2019. [Online]. Available: https://doi.org/10.1145/3287098.3287118

[4] R. Kedia, A. Sobti, M. Rungta, S. Chandoliya, A. Soni, A. K. Meena, C. M. Lobo, R. Verma, M. Balakrishnan, and C. Arora, "MAVI: Mobility assistant for visually impaired with optional use of local and cloud resources," in *International Conference on VLSI Design and International Conference on Embedded Systems (VLSID)*, Jan 2019, pp. 227–232.

[5] S. I. Venieris and C. Bouganis, "f-CNNx: A toolflow for mapping multiple convolutional neural networks on FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2018, pp. 381–3817.

[6] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[7] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017, pp. 5–14. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021740

[8] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst. (TRETS)*, vol. 12, no. 1, Mar. 2019. [Online]. Available: https://doi.org/10.1145/3289185

[9] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 37, no. 1, pp. 35–47, Jan 2018.

[10] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A high-performance CNN processor based on FPGA for mobilenets," in *International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2019, pp. 136–143.

[11] K. Guo, S. Han, S. Yao, Y. Wang, Y. Xie, and H. Yang, "Software-hardware codesign for efficient neural network acceleration," *IEEE Micro*, vol. 37, no. 2, pp. 18–25, Mar 2017.

[12] S. I. Venieris and C. Bouganis, "Latency-driven design for FPGA-based convolutional neural networks," in *International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2017, pp. 1–8.

[13] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[14] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Design Automation Conference (DAC)*, June 2017, pp. 1–6.

[15] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2016, p. 16–25. [Online]. Available: https://doi.org/10.1145/2847263.2847276

[16] Q. Sun, T. Chen, J. Miao, and B. Yu, "Power-driven DNN dataflow optimization on FPGA," in *International Conference on Computer-Aided Design (ICCAD)*, Nov 2019, pp. 1–7.

[17] Xilinx, "DPU for CNN v3.0," 2019. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_0/pg338-dpu.pdf

[18] Y. Xing, S. Liang, L. Sui, X. Jia, J. Qiu, X. Liu, Y. Wang, Y. Shan, and Y. Wang, "DNNVM: End-to-end compiler leveraging heterogeneous optimizations on FPGA-based CNN accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, pp. 1–1, 2019.

[19] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, and et al., "Going deeper with embedded FPGA platform for convolutional neural network," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2016, p. 26–35. [Online]. Available: https://doi.org/10.1145/2847263.2847265

[20] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017, p. 65–74. [Online]. Available: https://doi.org/10.1145/3020078.3021744

[21] K. Abdelouahab, C. Bourrasset, M. Pelcat, F. Berry, J.-C. Quinton, and J. Serot, "A holistic approach for optimizing DSP block utilization of a CNN implementation on FPGA," in *International Conference on Distributed Smart Camera (ICDSC)*, 2016, p. 69–75. [Online]. Available: https://doi.org/10.1145/2967413.2967430

[22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Neural Information Processing Systems (NeurIPS)*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 4107–4115. [Online]. Available: http://papers.nips.cc/paper/6573-binarized-neural-networks.pdf

[23] X. Zheng, L. K. John, and A. Gerstlauer, "Accurate phase-level cross-platform power and performance estimation," in *Design Automation Conference (DAC)*, 2016. [Online]. Available: https://doi.org/10.1145/2897937.2897977

[24] K. O'neal, P. Brisk, A. Abousamra, Z. Waters, and E. Shriver, "GPU performance estimation using software rasterization and machine learning," *ACM Trans. Embed. Comput. Syst. (TECS)*, vol. 16, no. 5s, Sep. 2017. [Online]. Available: https://doi.org/10.1145/3126557

[25] Y.-k. Choi, P. Zhang, P. Li, and J. Cong, "HLScope+: Fast and accurate performance estimation for FPGA HLS," in *International Conference on Computer-Aided Design (ICCAD)*, 2017, p. 691–698.

[26] H. Mohammadi Makrani, F. Farahmand, H. Sayadi, S. Bondi, S. M. P D, H. Homayoun, and S. Rafatirad, "Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design," in *International Conference on Field Programmable Logic and Applications (FPL)*, Sep 2019, pp. 397–403.

[27] H. M. Makrani, H. Sayadi, T. Mohsenin, S. rafatirad, A. Sasan, and H. Homayoun, "XPPE: Cross-platform performance estimation of hardware accelerators using machine learning," in *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2019, p. 727–732. [Online]. Available: https://doi.org/10.1145/3287624.3288756

[28] "Vivado design suite user guide (ug902)," 2020. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug902-vivado-high-level-synthesis.pdf

[29] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://arxiv.org/pdf/1812.00332.pdf

[30] M. Ferianc, H. Fan, R. S. Chu, J. Stano, and W. Luk, "Improving performance estimation for fpga-based accelerators for convolutional neural networks," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2020, pp. 3–13.

[31] ARM, "AMBA$^{®}$ AXI and ACE protocol specification," 2013. [Online]. Available: https://static.docs.arm.com/ihi0022/e/IHI0022E_amba_axi_and_ace_protocol_spec.pdf

[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *IEEE conference on computer vision and pattern recognition (CVPR)*, 2018, pp. 4510–4520.

[33] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[34] Xilinx, "Zynq UltraScale+ MPSoC ZCU102 evaluation kit," 2019. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

[35] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[36] Xilinx, "Zynq UltraScale+ Device," Aug. 2019. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf