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

Shikha Goel, Rajesh Kedia, M. Balakrishnan and Rijurekha Sen
Indian Institute of Technology Delhi, New Delhi, India
E-mail: {shikha.goel, kedia, mbala, riju}@cse.iitd.ac.in

***Abstract*—Deep Learning Processor Unit (DPU) from XILINX is among the numerous accelerators that have been proposed to speed up the execution of Convolutional Neural Networks (CNNs) on embedded platforms. DPUs are 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. To enable quick evaluation of choices among evolving CNN algorithms and accelerator configurations, we propose INFER (INterFerence-aware Estimation of Runtime). INFER is a framework to estimate the execution time of any CNN on a given size of DPU without actual implementation. Further, 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.**

***Index Terms*—Deep neural networks, Accelerators, Runtime estimation, Memory bandwidth sharing.**

I. Introduction

Embedded systems, spanning across autonomous vehicles [1], traffic monitoring [2], assistive devices [3], and many more, execute multiple Convolutional Neural Networks (CNNs) concurrently to realize different kinds of classification and detection tasks. Sustained efforts to improve CNNs has resulted in the availability of a large choice of CNNs for any given task [4]. These CNNs vary in terms of the accuracy that they are able to achieve as well as their compute/memory requirements (Table II). Deep Learning Processor Unit (DPU) [5] from Xilinx, originally developed by DeePhi Tech. and Tsinghua University [6], [7], is a generic FPGA based CNN accelerator that supports any CNN. DPU is configurable for various sizes that vary in compute capacity and the amount of FPGA resources used. Further, multiple DPUs could be implemented on the same FPGA and be active concurrently. With numerous choices of DPU sizes and newer CNNs evolving, it becomes essential to predict execution time for making efficient switching decisions at the run-time.

ZCU102 board used in this work was funded by MeitY, Govt. of India under "SMDP-C2SD" project. Accepted for publication in ICFPT 2020. Copyright IEEE, refer to original version on publisher's website (IEEE Explore).

We propose *INFER (INterFerence-aware Estimation of Runtime)*, a framework to predict the execution time of a CNN on a given configuration of DPU. Basic CNN characteristics like compute and memory requirements, along with DPU characteristics like number of available processing units and local memory size are used for prediction in INFER. The estimated value is further refined to account for delays in memory access due to the contention arising from 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). It is useful both at design-time for design space exploration [8] to choose the number/size of DPUs as well as at run-time for scheduling tasks on DPUs. Execution time estimation using very simple features of CNN and DPU, and interference modeling in a real-life setting are the key highlights of this work. Specifically, we claim the following key contributions of this paper:

1) Motivating the need for prediction of the runtime of CNNs on generic CNN processors like DPU
2) INFER, a framework 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 [9]

II. Related Work

There are several works on designing CNN accelerators for FPGAs [4], [10]–[12]. DNNWeaver [10] framework can map a variety of CNNs on FPGA and supports a variety of FPGAs (Intel and Xilinx). However, this is useful only at the design-time as a new bitstream needs to be generated for every CNN. FINN [11] framework focuses only on binary neural networks. Haddoc2 [12] 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 [5] 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.

ProxylessNAS [13] 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. [14] 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 [13] and Ferianc et al. [14] consider a single processor system where the effect of interference due to multiple CNNs running concurrently is not applicable. Qiu et al. [7] 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. Proposed Approach for Runtime Estimation

A CNN consists of many cascaded layers of different types like convolution layers, fully connected layers, etc. A DPU [5] is a generic CNN processor (accelerator) for Xilinx platforms. It performs layer by layer processing of CNN, which is invoked by a host CPU. 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 ARM AXI bus.

A. Runtime Estimator for Single DPU

We predict the execution time of individual layers of a CNN, which are then combined (added) to get the predicted execution time 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. We use subscript i to represent the layer number of the given CNN and subscript j to represent the DPU size.

CNN specific features: MAC_i represents the total number of MAC (multiply and accumulate) operations in a layer and contributes to the time taken for computation. Mem_i represents the amount of data transferred between local memory of DPU (BRAM) and main memory (DRAM) and contributes to the time taken for data transfer. DPU contains two separate AXI buses for accessing main memory to facilitate concurrent reading of weights and input data. Further, each AXI bus contains separate read and write paths, 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 options 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 other two options (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 and ssd_mobilenetv2 (see Table II for details) have both depthwise and pointwise convolution operation [15]. 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 the merge behavior of a CNN layer. 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 (~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.

DPU hardware related features: Different DPUs differ in number of BRAMs (N_BRAM_j) and the supported parallel MAC operations (N_MAC_j). 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 memory type and configuration associated with a 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.

Our prediction model uses these features to support various CNN types (classification or detection) and DPU configurations. 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 obtaining proprietary information. We 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

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 %

error). The random forest is a non-linear predictor, formed of multiple uncorrelated decision trees, and provides better prediction than any individual tree [16]. Makrani et al. [17] 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 though each CNN can 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, DPUs would experience interference due to sharing of main memory bandwidth. Fig. 1 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 for efficiently scheduling concurrent requests to DRAM [18], 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 main cause for increase in runtime, 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 $\sim$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 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:

$$BW = 1000 \times \frac{Data}{Time} \tag{1}$$

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.

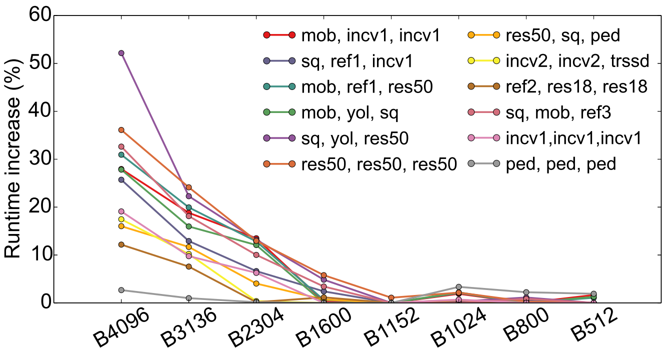

Fig. 1. Runtime increase due to memory contention for various CNNs

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 that the second order polynomial model has the least error. The estimated percentage 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 captures 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 minimize the mean error and hence excluded DPU size from the feature set.

IV. 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 [5] named B4096, B3136, B2304, B1600, B1152, B1024, B800, B512. We use B4096 (largest), B2304 (mid-sized) and B512 (smallest) DPUs for training purposes whereas validation is performed on all DPU sizes.

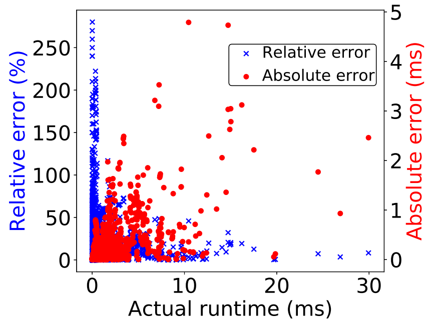

Fig. 2. Error distribution for different layers of various CNNs (for single DPU prediction)

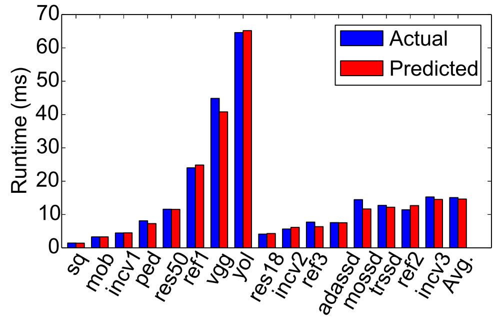

Fig. 3. Actual versus predicted runtime for various CNNs for B4096 (for single DPU prediction)

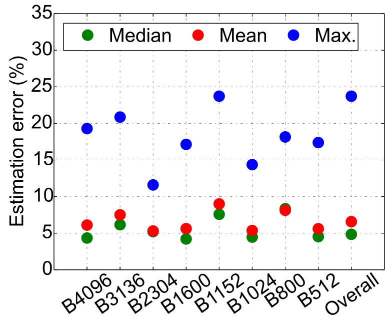

Fig. 4. Prediction error for different DPU sizes (for single DPU prediction)

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	

We create multiple hardware designs, each with three instances of a particular DPU size,[1] and implement them on Xilinx Zynq UltraScale+ (ZCU102) board [9]. 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.

[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.

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 specific CNN description file.

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. 2 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 small error values translate to large percentage errors when the base value is small. The figure also shows the distribution of absolute error in predicting the runtime for various CNN layers. The absolute error is mostly below 2 ms.

Fig. 3 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 ∼1.5 ms to ∼65 ms for B4096 DPU, and goes up to ∼405 ms for B512 DPU. Qiu et al. [7] uses internal design details and proposed a model to estimate the runtime of *vgg16* network on a particular DPU. Their estimation has an error of ∼30% compared to 9% in our model. The average leave-one-out cross-validation error (corresponding to Fig. 3) is 9.97% while the average out-of-sample error is 7.82%.

Fig. 4 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.

2) Estimating the effect of memory interference: Fig. 5 presents the error in estimating the effect of interference i.e., 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

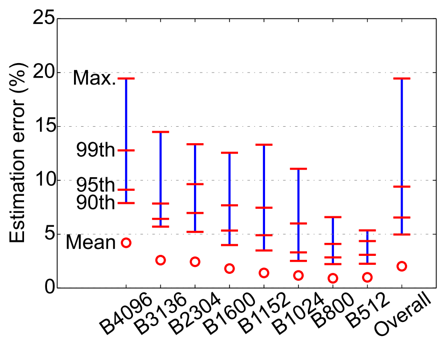

Fig. 5. Interference estimation error for different DPU sizes

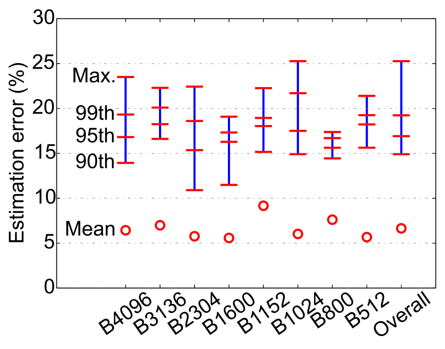

Fig. 6. Final estimation error (single DPU along with interference model) for different DPU sizes

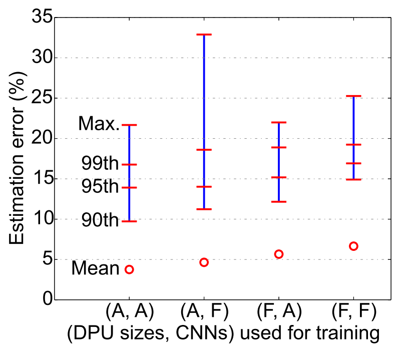

Fig. 7. Final estimation error for various training sets. A: All, F: Few (only *TRAIN* set)

the reduction in the overall range of interference for smaller DPU sizes (Fig. 1), 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 further observe that the use of *MBpl* (MegaBytes per layer of a CNN) as a feature for prediction 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 increased 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. 6 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. The average and maximum out-of-sample-error for INFER are 7.4% and 25.3% respectively. We observe high percentage errors primarily for networks which has a very small runtime (*sq*) or the training set did not include comparable networks (*mossd*). For CNNs with small runtime, any small absolute error in prediction results in high percentage error.

INFER executes on ARM CPU core available on Xilinx Zynq chips 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. 7 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. 6), indicating that the proposed predictor generalizes well for different count of CNNs.

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	78	0.78	0.73	89	0.85	0.89	91	0.76	0.93	231	31
yolo_v3-320	76	0.91	0.76	87	0.95	0.84	87	0.90	0.77	247	38
yolo_v3-416	93	0.88	0.94	94	0.91	0.96	95	0.83	0.96	428	61
yolo_v3-608	94	0.91	0.94	94	0.94	0.96	95	0.91	0.96	873	123

C. Illustrative Application: Traffic Monitoring System

We illustrate the use of execution time estimation of CNNs on DPUs using the example of a traffic monitoring system [2]. Table III shows the inference accuracy (measured as mean average precision) for different CNNs trained using the road traffic dataset [2], as well as their execution time for two different sizes of DPU. We experimentally observe that based on the number of vehicles on the road, different CNNs show different vehicle detection accuracy values. Further, the execution time of these networks varies significantly from 231 ms to 873 ms, providing trade-off opportunity. Vehicle detection systems can be used for multiple tasks with different accuracy requirements e.g., signal violation and speed violation.

Higher traffic density has fewer violations in comparison to lower traffic density and thus can do with a lower FPS (frames per second). At a lower FPS, both yolo_v3-416 and yolo_v3-608 have similar accuracy (MAP) but since yolo_v3-608 has higher precision, it can be chosen if we have adequate time available. At a higher FPS, the choice is between yolo_v2 and yolo_v3-320 and both have similar accuracy and runtime, but yolo_v3-320 is preferred based on precision. The runtime predicted by INFER for different DPU sizes, given any CNN, is crucial to make such dynamic choices of CNN 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).

V. CONCLUSION AND FUTURE WORK

CNNs are evolving fast and DPUs themselves have a variety. Since INFER can predict execution time without compiling CNNs or generating bitfiles, it is useful at design-time for choosing a suitable size FPGA along with suitable CNNs and DPUs as per cost-accuracy trade-off analysis [8]. After design-time decisions, execution times could be stored in a table for use at run-time. INFER allows seamless addition of new CNNs as it can also predict their run-time. We presented an approach for predicting the execution time of CNNs on generic CNN accelerators like DPU. Subsequently, we presented results from our framework INFER that predicts 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 INFER, our prediction framework, 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 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. In our future work, 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 one can choose an "optimal" CNN that is able to meet the deadlines.

REFERENCES

[1] J. Peng *et al.*, "Multi-task ADAS system on FPGA," in *AICAS*, 2019.
[2] M. S. Chauhan *et al.*, "Embedded CNN based vehicle classification and counting in non-laned road traffic," in *ICTD*, 2019.
[3] R. Kedia *et al.*, "MAVI: Mobility assistant for visually impaired with optional use of local and cloud resources," in *VLSID*, 2019.
[4] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network inference accelerators," *ACM TRETS*, 2019.
[5] "DPU for CNN v3.0," 2019. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_0/pg338-dpu.pdf
[6] K. Guo *et al.*, "Angel-eye: A complete design flow for mapping cnn onto embedded FPGA," *IEEE TCAD*, 2018.
[7] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *FPGA*, 2016.
[8] R. Kedia, S. Goel, M. Balakrishnan, K. Paul, and R. Sen, "Design space exploration of FPGA based system with multiple DNN accelerators," *IEEE ESL*, in press.
[9] Xilinx, "Zynq UltraScale+ MPSoC ZCU102 evaluation kit." [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html
[10] H. Sharma *et al.*, "From high-level deep neural models to FPGAs," in *MICRO*, 2016.
[11] Y. Umuroglu *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," in *FPGA*, 2017.
[12] K. Abdelouahab *et al.*, "A holistic approach for optimizing DSP block utilization of a CNN implementation on FPGA," in *ICDSC*, 2016.
[13] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *ICLR*, 2019.
[14] M. Ferianc, H. Fan, R. S. Chu, J. Stano, and W. Luk, "Improving performance estimation for fpga-based accelerators for convolutional neural networks," in *ARC*, 2020.
[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018.
[16] L. Breiman, "Random forests," *Mach. Learn.*, 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324
[17] H. Mohammadi Makrani *et al.*, "Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design," in *FPL*, 2019.
[18] Xilinx, "Zynq UltraScale+ Device." [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrascale-trm.pdf