FPGA-based Dynamic Deep Learning Acceleration for Real-time Video Analytics

Yufan Lu[1], Cong Gao[1], Rappy Saha[2], Sangeet Saha[1], Klaus D McDonald-Maier[1], and Xiaojun Zhai[1]

[1] University of Essex, Colchester, United Kingdom
{yl19888,cg21670,sangeet.saha,kdm,xzhai}@essex.ac.uk
[2] rappysaha10@gmail.com

Abstract. Deep neural networks (DNNs) are a key technique in modern artificial intelligence that has provided state-of-the-art accuracy on many applications, and they have received significant interest. The requirements for ubiquity of smart devices and autonomous robot systems is placing heavy demands on DNNs-inference hardware, with high requirement for energy and computing efficiencies, along with the rapid development of AI techniques. The high energy efficiency, computing capabilities, and reconfigurability of FPGAs make these a promising platform for hardware acceleration of such computing tasks. This paper primarily address this challenge and propose a new flexible hardware accelerator framework to enable adaptive support for various DL algorithms on an FPGA-based edge computing platform. This framework allows run-time reconfiguration to increase power and computing efficiency of both DNN model/software and hardware, to meet the requirements of dedicated application specifications and operating environments. The achieved results shows that with the proposed framework is capable to reduce energy consumption and processing time up to 53.8% and 36.5% respectively by switching to a smaller model. In addition, the time and energy consumption are further elaborated with a benchmark test set, which shows that how input data in each frame and size of a model can affect the performance of the system.

Keywords: Deep Neural Networks, Hardware Accelerator, Edge Computing, real-time video analytics, FPGAs

1 Introduction

Due to the recent advancement of digital technologies, deep neural networks (DNNs) have emerged as a key technique in modern artificial intelligence (AI), that provides state-of-the-art accuracy for many applications [1]. Generally speaking, DNN models inference technique can be adopted wide range of devices, i.e. CPU, ASIC, FPGA, GPU, Microcontroller. While for CPU and GPU implementation, these techniques are available for a wide range of DNN models. For ASIC and FPGA implementation, although they are commonly applied in embedded

fields [2], the techniques and tools supported in DNNs applications are relatively new. Different hardware platforms have different advantages, depending on the applications and user requirements. For our paper, we focus on FPGA based heterogeneous platform. FPGA based DNN model implementation can deliver better performance and energy efficiency compare to CPU and GPU [3]. We conclude this statement from the previous researches, proposed different kinds of FPGA based accelerators [3–6] to improve overall performance.

Although choosing hardware is important, it is also important to understand the application, implementation technique and tool's availability. Usually, we focus to solve a problem by applying single model. But it may not seem to be the best solution. [7] showed that for image classification purpose switching between different models based on the input increases overall performance. They demonstrated their idea in a GPU-based environment. [8] focused on available resource monitoring in run-time to switch between the pre-defined models in a CPU-GPU environment. To be able to switch between different models, it must be necessary to generate multiple models based on the necessity. Here comes the idea of neural architectural search (NAS) [9]. Multiple model generation, model switching to improve performance, model switching to improve resource usability, these three aspects drive our motivation to propose a new framework for the FPGA based heterogeneous environment where we will be able to switch between the models in order to improve performance and resource usability.

In this paper, we present an improved flexible hardware accelerator framework, that can provide a significant level of adaptability support for various DL algorithms on an FPGA-based edge computing platforms. The platform can dynamically configure hardware and software processing pipelines to achieve better cost, power, and processing efficiency for the dedicated application requirements at run-time. To demonstrate the effectiveness of the proposed solution, we implement our framework for a DNNs based real-time video processing pipeline on a Xilinx ZCU104 platform, where we carry out a set of comprehensive experiment tests to evaluate the performance of the proposed scheme.

The achieved results shows that with the proposed framework is capable to reduce energy consumption and processing time up to 53.8% and 36.5% respectively by switching a smaller model. With a further discussion on time and power cost of the framework, a algorithm could be developed to define a strategy to control the switching behavior to achieve the optimised system performance. In summary, we make the following novel contributions in this paper:

- We propose an improved flexible DNN hardware accelerator framework that can dynamically configure the hardware and software processing pipelines to achieve improved power consumption and latency performance metrics.
- We have carried out comprehensive evaluations of DNN model sizes and inference performance, when using Xilinx DPUs in video analytic applications.
- We build a complete DNNs based real-time video processing pipeline and evaluate the effectiveness of the proposed framework.
- We designed a benchmark tool set to further analysis time and energy consumption of the framework.

The paper is organized as follows: Section 2 introduces the overall system. Section 3 briefly explains the DNN model optimisation strategy. Section 4 details the hardware and software pipeline setups. Section 5 details the experiments carried out under different scenarios and evaluates, compares, and discusses the results, Section 6 details the time and energy consumption of the proposed framework, and Section 7 draws conclusions and future plans.

2 Overview of the proposed system

The proposed scheme can support one to n implementations and offers a great level of flexibility and run-time efficiency for run-time video analytics applications. The proposed system consists of three main software components: 1) Neural network architecture search algorithms, that can generate different sub-networks with given constraints, 2) Neural network model compilation that can convert sub-networks into FPGA focused executing formats, and 3) Run-time management that can support dynamic execution of sub-networks on heterogeneous devices. A high-level overview diagram is presented in Fig 1.

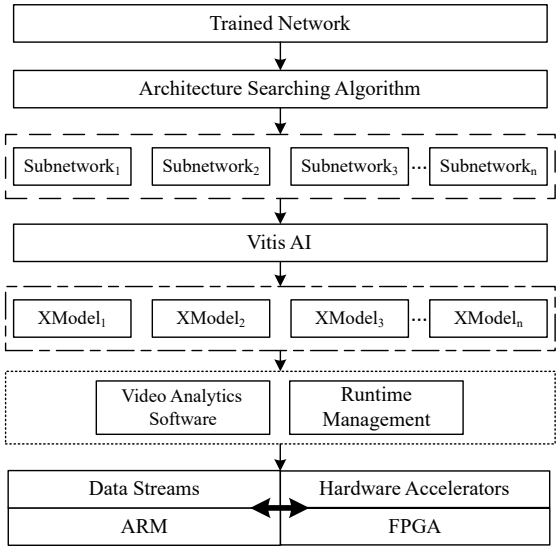


Fig. 1. The system architecture of the proposed scheme.

2.1 Neural network architecture search

Neural network architecture search (NAS) is a popular technologies to reduce the size of the neural networks. Generally, the NAS algorithm does not consider the

target hardware architecture and run-time conditions directly because it lacks accurate cost information to feedback to the NAS algorithm. For example, one to one NAS optimisation is able to generate a network architecture Net_1 that meets the design requirements of accuracy $a_1 > A$, power consumption $p_1 < P$, and latency $l_1 < L$. However, during run-time inference, the input data might be challenging. The accuracy does not meet the designed parameters, e.g. $a_1 < A$.

Consequently, the power consumption and latency could increase accordingly due to longer processing time required. Recent work proposed by [10] introduced an interesting NAS method, OFA that can produce a variety of network architectures based on the constraints of latency and accuracy. In this paper, we integrate OFA into a joint optimisation tool-chain, that can take advantage of this approach to produce a one to n inference model to meet various needs at run-time. The details of this optimisation approach are introduced in Section 3.

2.2 Neural network model compilation

Network models developed in the mainstream frameworks needs to be mapped into a high-efficient instruction set and data flow for the targeted hardware platform. In this work, we use Vitis-AI to generate complied network model, where 32-bit floating-point weights and activations are converted 8-bit fixed-point [11]. Ultimately, the AI model is mapped into a high-efficient instruction set and data flow along with sophisticated optimisations, such as layer fusion, instruction scheduling, and reuses on-chip memory as much as possible by Vitis-AI.

2.3 Software and hardware run-time management

The run-time management of this system is implemented using Vitis AI Runtime (VART), which enables the applications to use the unified high-level runtime API. VART offers asynchronous submission and collection of jobs to the accelerator and supports multi-threading, and multi-process execution [12].

3 DNN model optimisation

3.1 Brief introduction of OFA

OFA [10] consists mainly of 5 blocks, and in each block (i.e. convolution layers unit), depth, width and weight kernel size can be varied as per the following example: depth $D = \{2, 3, 4\}$, width $W = \{3, 4, 6\}$, kernel size $K = \{3, 5, 7\}$, where D, W and K represents the number of convolution layers and channels, size of filters in a single block respectively. It is assumed that each variable is independent to each other, so, number of subnetworks will be $((3 \times 3)^2 + (3 \times 3)^3 + (3 \times 3)^4)^5 \approx 2 \times 10^9$. In OFA, any model like Resnet [13] and Mobilenet [14] can be utilised and trained progressively, while maintaining the variability in depth, width, or kernel size. To identify a subnetwork from this vast number of subnetworks, they used latency and accuracy as a constraint in the random search and evolutionary search algorithms.

3.2 Model generation and optimisation

We use the OFA trained network as a super network and its searching algorithms to generate multiple subnetworks according to our requirements. The latency is firstly used as an input parameter in the search algorithm. Fig 2 describes the model generation technique, where the model is optimised in terms of latency and accuracy. In the OFA framework, random search is first used to determine a set of subnetworks (Subnet N) that are close to the defined latency. Evolutionary search is then used to identify the subnetworks (Subnet K) with the highest accuracy among the previously selected subnetworks.

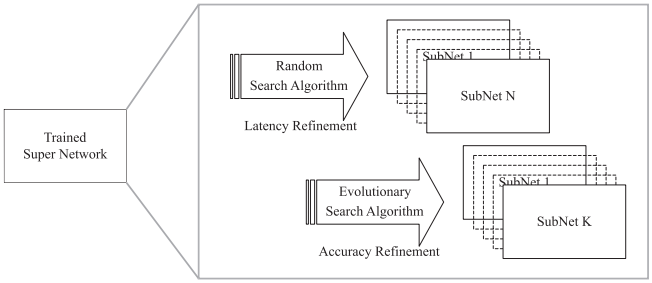


Fig. 2. The model generation and optimisation technique.

4 System hardware/software co-design

4.1 Hardware architecture

A real-time DNN based video analytic system typically includes four parts: 1) *video decoding*, 2) *preprocessing* (e.g. resize and normalisation), 3) *DNN inference* and 4) *post-processing*. Because both DNN, inference and other processes, require significant computational resources, the acceleration design should consider DNN inference and other processing tasks. Therefore, in addition, to deploying the DNN inference, hardware accelerators for video decoding and preprocessing should also be deployed. However, as the requirements of post-processing algorithms vary in different DNN models, the post-processing tasks are implemented in software.

The overall system architecture is shown in Fig. 3. There are mainly three types of accelerators: 1) *Xilinx H.264/H.265 video Codec unit (VCU)* [15], which is a hardware IP used for video coding and decoding tasks; 2) *Preprocessing module*, which is a high level synthesis (HLS) implemented hardware module and dedicated for resizing and normalisation tasks; and 3) *Deep learning processing units (DPUs)*, used for deep learning inference tasks, which can be reconfigured in different scenarios at run-time.

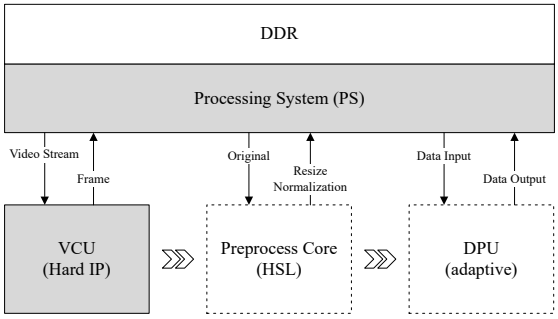


Fig. 3. The hardware architecture of the proposed platform.

To process video streams in real-time, the input video stream will be firstly decompressed by the VCU, so that the video stream is converted into separate frames. Secondly, the preprocessing core will carry out resize and pixel value normalisation on each frame. Both the VCU and preprocessing modules in this system can process up to 3840×2160 pixels at 60 frames per second, which has been set to 1080p video streams in our experimental scenarios. Therefore, the system's bottleneck should not be the VCU or preprocessing modules. Instead, the system performance will most likely be limited by the DPUs and other processes. Hence our ultimate goal is to reconfigure them at run-time to achieve the optimal performance for the entire system.

4.2 Software implementation

For the software part, we developed video analytic applications using Vitis Video Analytics SDK (VVAS) [12], a GStreamer-based plugin development framework. The Gstreamer runs video processing pipelines in multi-threads. Hence, we can precisely control the DNN inference processes by introducing several customised plugins for multiple video analytic applications.

Fig. 4 shows two types of pipelines representing different video analytic applications. 'Pipeline (a)' represents a typical one-stage video analytic application (e.g. object detection and segmentation), where only one DNN model is used to conduct an inference once per frame. 'Pipeline (b)' represents a two-stage video analytic application (e.g. tracking, Re-Identification, and car plates detection), where two DNN models are executing simultaneously, and the second one may execute multiple times, due to detection results of the first one.

4.3 Communication between processes

We design a pair of communication interfaces (e.g., read and write communications) in DNN inference plugins, to report DNN inference information and control its run-time DNN model process. While the pipeline is running, the execution time of the DNN model, processing time of pipeline and the power consumption of the entire system will be simultaneously sent to a separate system

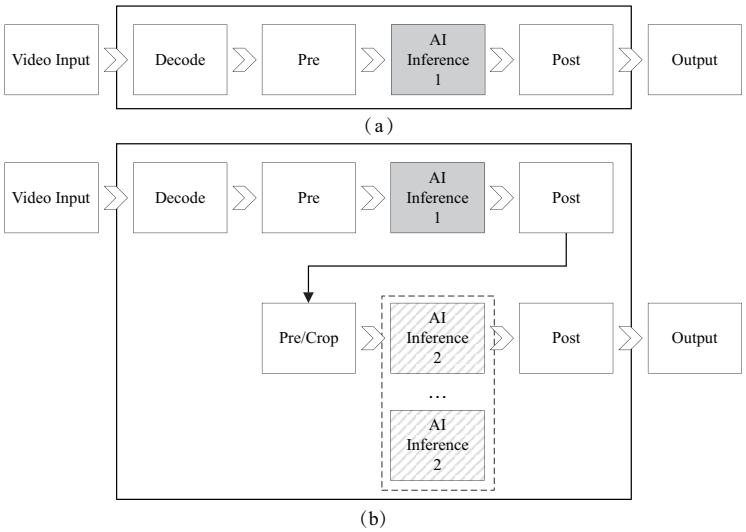


Fig. 4. Video processing pipelines in the proposed system. (a) represents an application with one stage AI inference task. (b) represents an application with two-stage AI inference task.

management thread. According to the real-time performance metrics from the system, the processing pipeline can be reconfigured accordingly. In our previous work, [16], it is concluded that power and computing efficiency of the proposed system can be further improved by hardware reconfiguration, when workloads of the system are increased.

The design of the communication framework is shown in Fig. 5. There are three software layers, including 1) Python management interfaces for user control, 2) Gstreamer applications to run AI inference and 3) system info (e.g., hardware temperature and power consumption). In the work, the system info is recorded in Proc file system (a virtual file system in Linux). During run-time, Gstreamer applications continuously read the virtual via file IO interfaces. Each time when a virtual file is read, a function will be triggered to read sensor data. The history data points are stored in applications. If necessary, the data can also be passed to the Python management program via the name pipe interface. The named pipe interface is the main method to transfer data between applications and management programs. Applications can send real-time status to or get commands from the management program. The "post" plugin in Gstreamer applications is a key module to collect system info (e.g., power) and running status (e.g., FPS). It is also responsible for transferring inference results. If the output result contains massive data (e.g., semantic segmentation), it will use file IO to store output results.

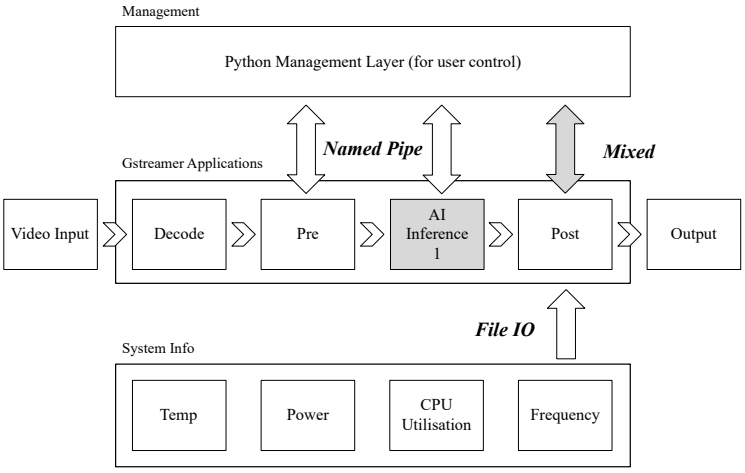


Fig. 5. Design of the communication framework.

5 Experiments

In this section, experimental setup and results are comprehensively reported. A typical real-time video processing pipeline is implemented on a Xilinx ZCU104 (XCZU7EV) for both car and pedestrian detection and classification via analytic applications.

5.1 Overall system setup

We implement the proposed video analytics framework on a Xilinx ZCU104 (XCZU7EV-2FFVC1156 MPSoC) development platform, where each frame follows the architecture of the pipeline described below:

1) *The input*: the video streams are loaded to the pipeline; 2)*Pre-processing unit*: carries out resizing and normalisation functions to allow the processed video data streams to meet the input requirements of DNN models; 3) *Object detection DNN inference*: deployment of object-detect DNN model on DPUs (e.g. YOLOv3 for cars and Refinet for pedestrians detection respectively [17, 18]); 4) *Image cropping*: cropping detected objects in each video frame and send them to a second AI inference module for more precise classification tasks; 5) *Object classification*: deployment of Resnet-50 based backbone networks, and the network models are generated with different sizes based on the OFA algorithm.

Configuration The proposed design is implemented using Xilinx Vivado 2021.1, VVAS 1.0 framework and PetaLinux 2021.1 on a Xilinx ZCU104 evaluation platform, video streams ($1920 \times 1080@30FPS$) are used for testing. We deploy and integrate two DPUs (i.e. B3136) in the video processing pipeline (i.e. Fig 3). The detailed hardware utilization are reported in Table 1 and Table 2.

Table 1. Hardware resource utilisation

	LUT	LUTRAM	FF	BRAM	DSP
Used	144,913	13,599	247,407	261	1,217
Available	230,400	101,760	460,800	312	1,728
Utilisation (%)	62.9	13.4	53.7	83.5	70.4

Table 2. Sub-module resource utilisation

Sub module	LUT	Register	BRAM	DSP
DPU	47667	85778	210	436
VPU	105	24	0	0
Pre-processing unit	13147	17390	12	40

5.2 DNN model management

We use Xilinx Vitis-AI 1.4.1 tool-chain to convert Pytorch DNN models into xmodel files. By scaling DNN models based on the OFA searching strategy, we obtain three different sizes of OFA-resnet-50 models: OFA700, OFA1000 and OFA2000. The number after each OFA model represents million floating-point operations per second (MFLOPs), which is used as the threshold of the sub-net searching in the OFA algorithm. Because 700 is the lowest value from all selected sub-networks and when value above 2000 the model size will increase with less accuracy improved, and we select 700 and 2000 to represent a large model and a small model separately and select 1000 as a medium one. Table 3 summarises a list of models used in our experiments, which includes a number of sub-networks generated by using the OFA network. The communication interfaces introduced in Section 4.C are implemented to update the DNN models at frame level dynamically.

Table 3. Parameters of the used DNN models

Model	Parameter size (MB)	Workload (MOPS)	Accuracy (Top1/Top5 ImageNet-1k)
OFA700	10.75	1340.61	74.9%/92.4%
OFA1000	18.02	1905.48	77.0%/92.8%
OFA2000	32.88	3805.47	79.7%/94.7%
ResNet-50	26.22	7360.32	83.2%/96.5%

In our experiment, we have verified the effectiveness of this management scheme using a DNN powered car/pedestrian re-identification application, where a two-stage DNN pipeline is implemented. The first stage of the DNN pipeline uses Yolov3/Refinet for car/pedestrian detection, respectively, and the second stage of the pipeline uses Resnet-50 as a backbone network for car/pedestrian classification.

5.3 Results and analysis

As shown in Fig 6, we have tested our framework for two different video analytic applications: car and pedestrian Re-Identification. The proposed framework shows the capability to handle the videos and identify objects correctly through tests in both scenarios.



(a) (b)

Fig. 6. Testing scenarios. (a) Car re-identification; (b) Pedestrian re-identification

In our experiment, we use the same video processing pipeline to handle different video input streams, and continuously monitoring system's performance metrics, such as frame rate (FPS), energy consumption and DPU latency, via the proposed customised communication interference plugin. In the proposed work, we measure real-time on-board power from ZCU104 registers when using different sizes of DNN models. The total energy consumption can be calculated using the following equation:

$$E = \sum_{i=1}^{n} P_i / f \tag{1}$$

where E denotes the total energy consumption in Joules (J), P denotes power consumption in Watt (W) at time i, f denotes sampling frequency in Hz. From Fig 7, the total energy consumption is reduced by 18.9%, 38.4% and 53.8% in the car scenarios respectively, and similarly reduced by 25.4%, 41.1% and 61.6% respectively in the pedestrian scenarios.

In general, smaller size DNN models achieve better DPU inference latency due to less operations required, as the result of this, the entire video pipeline will finish quicker than deploying a larger model. Therefore, we can implement a dynamic model-switch strategy to select a suitable DNN model based on the real-time performance metrics to improve performance of the entire system. By comparing the original model, the proposed system can reduce the latency of DPUs for the whole video pipeline by 12.9%, 23.9%, and 36.5%, 14.0%, 25.9%, and 38.6% in car and pedestrian scenarios respectively.

The detailed DPU inference latencies for each OFA network model are summarised in Fig 8.

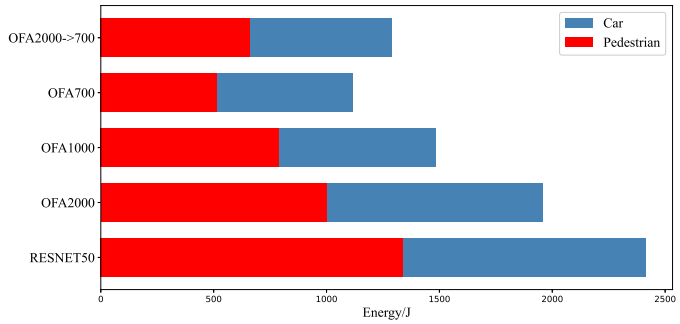


Fig. 7. Total energy consumption for running different models

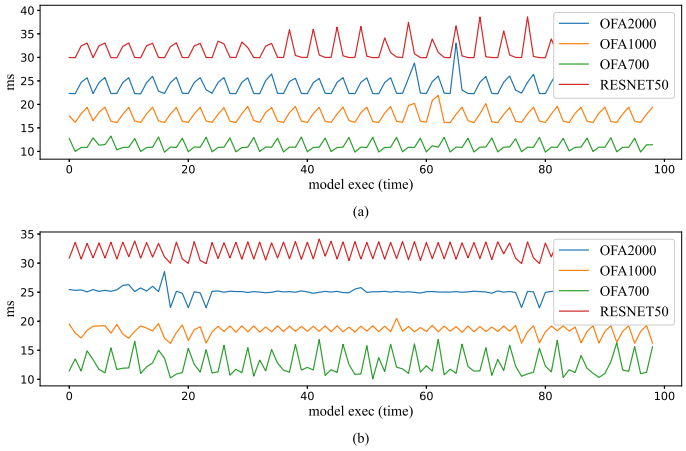


Fig. 8. Latency in different scenarios. (a) represents Latency in car scenarios. (b) represents Latency in pedestrian scenarios

Fig 9. (a) and (c) show FPS results of OFA700, OFA1000, OFA2000 and Resnet-50 on the car and pedestrian scenarios respectively. By comparing the FPS of Resnet-50, the overall FPS is increased by 26.3%, 65.6% and 113.0% in car scenarios by using OFA700, OFA1000, and OFA2000 models respectively; Similarly, it is also increased 27.1%, 65.7% and 132.1% FPS in pedestrian scenarios respectively.

In certain running environments (e.g. simple scenarios but with varying amounts of objects), we can dynamically switch the DNN models by monitoring the run-time performance metrics. Thus it could further increase overall power and computing efficiency with an acceptable loss of classification accuracy (Table 3). For example, as shown in Fig 9 (b) and (d), by switching the DNN model to a smaller one at run-time, the average frame rates are increased from 17.04 FPS to 29.4 FPS in the car scenarios and from 16.9 FPS to 30.8 FPS in the pedestrian scenarios respectively. Meanwhile the energy consumption is also dropped by 34.2% and 34.0% respectively for the two scenarios (see Fig 7).

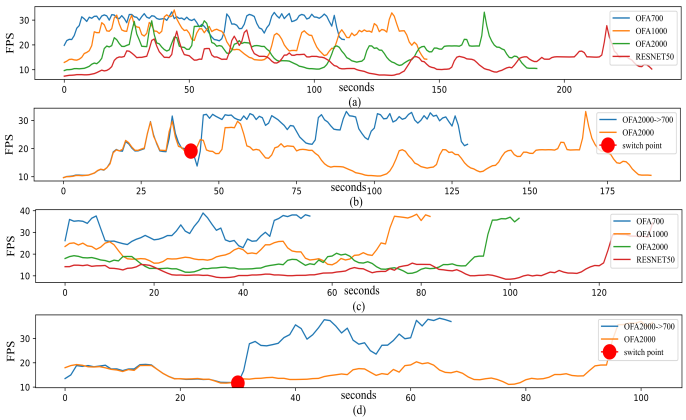


Fig. 9. FPS of the proposed system in different scenarios. (a) Car scenarios without model switch. (b) Car scenarios with model switch. (c) Pedestrian scenarios without model switch. (d) Pedestrian scenarios with model switch

6 Time and power consumption

In this sections, we will discuss and elaborate consumption (e.g. Time and energy) for the proposed framework.

6.1 Impact factors

Data volume: In section 5, each detected object will be cropped and sent to the second stage of the pipeline (e.g. Resnet-50 or OFA-Resnet-50 networks) for the classification work. The workloads will be significantly varied in the second stage of the pipeline with the number of objects were detected in the first stage. **Size of models:** Additional time consumption is majorly needed for rewriting the bitstream and updating software drivers, and the time for rewriting bitstream is varied by the sizes of the reconfiguration stream. On the other hand, the extra power consumption maybe caused by rewriting the RAMs. Scale/size of a model may influence time consumption of a single object classification job as well.

6.2 Benchmark

In order to analysis these factors, we create 7 different scene videos contain different number of cars, and test them for 2 different modes. The first mode is normal display mode, in which it just displays videos using VVAS. In the second mode, there are some model switch behaviors added with same time interval. We record both time and energy consumption and attempt to analyse how these factors influence the system performance.

Time consumption: We can calculate a single switch time cost by comparing the time consumption between two modes, and through multi-set of data, we find that time cost of model switching behavior is irrelevant to the sizes of the models. It is varied between 30 ms to 300 ms. It may because the bandwidth of DDR is large enough for small data exchanges in the benchmark. From Fig.10, it is easily to conclude that the cost in a normal process will increase if either the sizes of the models or the target number increases, and there is a linear relationship between data volume and time consumption, which is a factor of 12 ms per object for using OFA700 model. Models with larger size (e.g. OFA1000 and OFA2000) will spend 64.4 % and 112.5 % more time respectively compared to OFA700 model.

Energy consumption: As shown in Fig.11, the energy consumption will be increased by 24.6 %, 111.8 %, 289.0 %, 636.2 % and 1345.4 % for detecting 2, 4, 8, 16, 32 objects respectively when comparing to only 1 object is detected. With the same object number, the energy consumption will be increased by 3 % and 9 % for OFA1000 and OFA2000 respectively when comparing to OFA700.

7 Conclusion

In this work, we proposed designing a new flexible hardware accelerator framework to enable adaptive support for various DNN algorithms on a FPGA-based edge computing platform. The achieved results shows that with the proposed framework is capable to reduce energy consumption and processing time up to 53.8% and 36.5% respectively by switching to a smaller model. While using a dynamical model-switch strategy, the frame rates are increased immediately

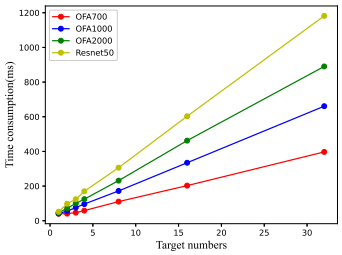


Fig. 10. Time consumption with different target numbers

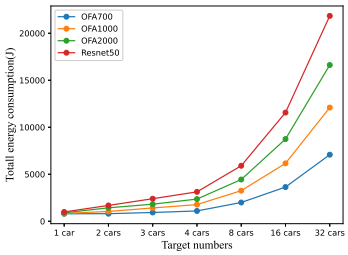


Fig. 11. Energy consumption with different target numbers

at the switching point. The average frame rates are increased from 17.04 FPS to 29.4 FPS in the car scenarios and from 16.9 FPS to 30.8 FPS in the pedestrian scenarios, respectively. Two major impact factors, data volumes and size of model, have been further discussed with a designed benchmark and the statistical data shows that both factors have the positive impact on the energy and time consumption of the framework. By further combining a dynamic-reconfiguration strategy in hardware modules, the proposed system could offer an unprecedented opportunity to create new adaptable architectures and algorithm models using the hybrid-computing units and resources. It is anticipated that it will greatly impact increasing energy efficiency, performance, and flexibility.

Acknowledgment

This work is supported by the UK Engineering and Physical Sciences Research Council through grants EP/R02572X/1, EP/P017487/1, EP/V034111/1, EP/X0 15955/1 and EP/V000462/1.

References

1. A. Shawahna, S. M. Sait, and A. El-Maleh, "Fpga-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.
2. Y. Lu, X. Zhai, S. Saha, S. Ehsan, and K. D. McDonald-Maier, "A self-adaptive seu mitigation scheme for embedded systems in extreme radiation environments," *IEEE Systems Journal*, 2022.
3. K. Lübeck and O. Bringmann, "A heterogeneous and reconfigurable embedded architecture for energy-efficient execution of convolutional neural networks," in *ARCS*, 2019.
4. 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." New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3020078.3021744

5. J. Haris, P. Gibson, J. Cano, N. B. Agostini, and D. R. Kaeli, "SECDA: efficient hardware/software co-design of fpga-based DNN accelerators for edge inference," *CoRR*, vol. abs/2110.00478, 2021. [Online]. Available: https://arxiv.org/abs/2110.00478
6. X. Zhang, H. Ye, J. Wang, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnexplorer: A framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3400302.3415609
7. B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," *SIGPLAN Not.*, vol. 53, no. 6, p. 31–43, jun 2018. [Online]. Available: https://doi.org/10.1145/3299710.3211336
8. W. Lou, L. Xun, A. Sabet, J. Bi, J. Hare, and G. V. Merrett, "Dynamic-ofa: Runtime dnn architecture switching for performance scaling on heterogeneous embedded platforms," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 3104–3112.
9. T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html
10. H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://arxiv.org/pdf/1908.09791.pdf
11. V. Kathail, "Xilinx vitis unified software platform," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 173–174.
12. "Xilinx vitis-ai 1.4 release," https://github.com/Xilinx/Vitis-AI, 2021.
13. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2016, pp. 770–778. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.90
14. 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," 2017.
15. Xilinx, "H.264/H.265 Video Codec Unit v1.2," Tech. Rep., 2021. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/vcu/v1_2/pg252-vcu.pdf
16. Y. Lu, X. Zhai, S. Saha, S. Ehsan, and K. D. McDonald-Maier, "Fpga based adaptive hardware acceleration for multiple deep learning tasks," *2021 IEEE 14th International Symposium on Embedded Multicore*, 2021.
17. J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," apr 2018. [Online]. Available: https://arxiv.org/abs/1804.02767v1
18. K. Fu, Q. Zhao, and I. Y.-H. Gu, "Refinet: A deep segmentation assisted refinement network for salient object detection," *IEEE Transactions on Multimedia*, vol. 21, no. 2, pp. 457–469, 2018.