Realisation of Early-Exit Dynamic Neural Networks on Reconfigurable Hardware

Anastasios Dimitriou[1], Lei Xun[2], Jonathon Hare[3], Geoff V. Merrett[4]
School of Electronics and Computer Science, University of Southampton, UK
Email: {[1]ad1r20, [2]l.xun}@soton.ac.uk, {[3]jsh2, [4]gvm}@ecs.soton.ac.uk

***Abstract*—Early-Exiting is a strategy that's becoming popular in Deep Neural Networks (DNNs), as it can lead to faster execution and a reduction in the computational intensity of inference. To achieve this, intermediate classifiers abstract information from the input samples to strategically stop forward propagation and generate an output at an earlier stage. Confidence criteria are used to identify easier-to-recognise samples over the ones that need further filtering. However, such dynamic DNNs have only been realised in conventional computing systems (CPU+GPU) using libraries designed for static networks. In this paper, we first explore the feasibility and benefits of realising early-exit dynamic DNNs on FPGAs, a platform already proven to be highly effective for neural network applications. We consider two approaches for implementing and executing the intermediate classifiers: *pipeline*, which uses existing hardware, and *parallel*, which uses additional dedicated modules. We model their energy needs and execution time and explore their performance using the *BranchyNet* early exit approach on LeNet-5, AlexNet, VGG19 and ResNet32, and a Xilinx ZCU106 Evaluation Board. We found that the dynamic approaches are at least 24% faster than a static network executed on an FPGA, consuming a minimum of 1.32x lower energy. We further observe that FPGAs can enhance the performance of early-exit dynamic DNNs by minimising the complexities introduced by the decision intermediate classifiers through parallel execution. Finally, we compare the two approaches and identify which is best for different network types and confidence levels.**

***Index Terms*—dynamic neural networks, early-exiting, FPGA, low resource, hardware architecture for machine learning.**

I. Introduction

The success of modern Deep Neural Networks (DNNs) is partly due to their increased depth and complexity, containing a large number of sequentially connected layers. This leads to higher effectiveness and accuracy in solving many real-life machine learning problems, including computer vision [1] and voice recognition [2]. However, this comes at the cost of a higher computational burden, leading to increased latency and energy demands, rendering their deployment on resource-restricted devices very challenging or even impossible.

Dynamic inference is an emerging approach that utilises information from input samples to selectively execute only important subsets of the DNN, e.g. layers [3], channels [4] or sub-networks [5]. More specifically, early exiting [6], [7], [8] is a method that adds intermediate classifiers to the network, terminating inference once the network is confident enough to recognize the input. To achieve this, they compare their exit with predetermined confidence thresholds that can also take into account elements like target device capabilities and workload. While these classifiers introduce additional computational load, careful design and training can lead to significant enhancement in the overall performance of the network [9],[10].

This article is an extended version of our previous short paper [11], which represented the first implementation of a Dynamic DNN on an FPGA. We contend that there is a gap between theoretical findings and real-world applications [12]. The majority of early exit networks are designed using libraries optimized for static models and evaluated on conventional Central and Graphics Processing Unit (CPU-GPU) systems, resulting in inconsistency between anticipated and actual effectiveness. Performance on CPUs still cannot meet the real-time processing requirements in embedded devices [13], and GPU power consumption is too high [14]. Field Programmable Gate Arrays (FPGAs) have proven to be very effective in accelerating DNNs [15]. They consist of a flexible collection of logic elements and IP blocks that can be configured for a specific application and can accommodate massively parallel operations. They have been shown to offer enhanced capabilities to efficiently accommodate matrix multiplications, the majority of computations in DNNs while achieving very low energy consumption [16]. However, the benefits of dynamic DNNs have not been explored on reconfigurable hardware. The novel contributions of this paper are:

- The first exploration into the benefits of dynamic DNNs on FPGAs, experimentally evaluated on a ZCU106 FPGA with four different models and compared against a desktop CPU, GPU and embedded Jetson Xavier. Results show that the dynamic DNN accelerates the average inference time by 1.5-3x on the FPGA while also executing faster than the desktop CPU and with lower energy consumption than the embedded Jetson.
- We present and explore ***pipeline*** and ***parallel*** design approaches for the intermediate classifiers, showing how the latter eliminates both the halting of the backbone network and the requirement to store outputs of intermediate layers, reducing the worst-case inference latency to be equal to the average latency of a static network.
- We characterise the ***pipeline*** and ***parallel*** design approaches over latency, energy consumption and memory needs and explore the effect of different early-exit confidence thresholds, allowing the exploration of the most appropriate approach for a given scenario.

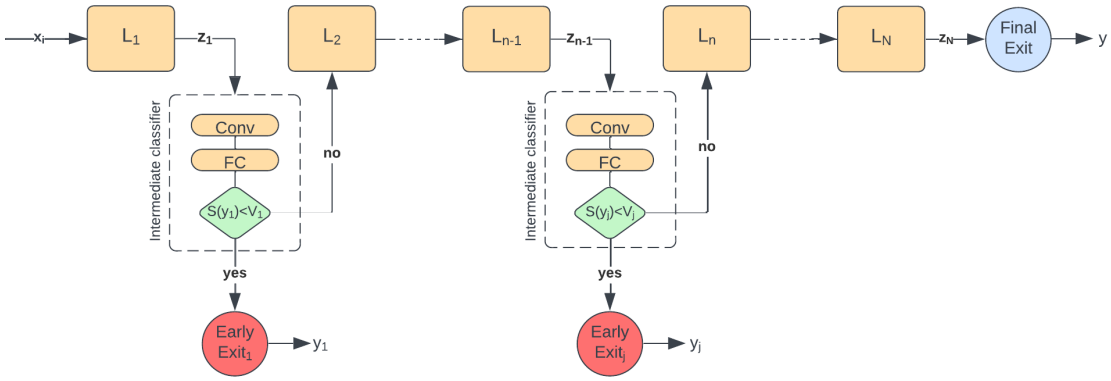

Fig. 1: Structure of a typical *N* layer early exit dynamic DNN with *j* early exits.

II. Preliminaries

A. Early Exiting

Early Exiting is a structure-focused dynamic approach that allows inference to stop in an intermediate stage when the network has sufficient confidence in classifying the input. If x_i is the *ith* input sample, the forward propagation of a N-layer deep network can be written:

$$y = L_{n_i} \circ L_{n_{i-1}} \circ \cdots \circ L_1(x_i), 1 \leq n_i \leq N \tag{1}$$

Where $L_{n_i}$is the *i-th* output of the *N* network layers, n_i is the layer at which the inference stops and y is the network's output.

A common technique that enables early exiting is placing intermediate classifiers in between each or sets of layers (Fig. 1), to determine if the execution of the remainder of the network is required to produce an acceptable output. Park *et. al.* [17] propose a two DNN architecture. If the difference between the two largest outputs (softmax) of the first network exceeds a threshold, an early exit occurs; otherwise, the second model is executed. In classification tasks, confidence thresholds are typically denoted by the highest value in the softmax output [3], [18], although some works also consider measures such as entropy [9], [6] and score margin [17]. For Natural Language Processing (NLP) tasks, a concept of "model patience" [8] is used, which means that if the predictions for a particular instance remain constant, even after a series of classifiers, the inference process concludes.

In this paper, we use the approach from Teerapittayanon *et. al.* [9], which introduces intermediate classifiers between layers that perform early exiting according to confidence-based criteria. The classifiers are trained alongside the backbone network as a joint optimisation problem, using the softmax cross entropy loss function. Once the network is trained, entropy is used to measure how confident the branch at an exit point is. Entropy can be written as:

$$S(y) = \sum_{c \in C} y_c \cdot \log y_c \tag{2}$$

where y is the branch's output that contains the probabilities for every one of the C possible labels. Entropy is then compared over a vector V, containin the thresholds for every exit point. For example, in the *j-th* exit point after the L_n layer, entropy is compared with V_j. If it is smaller, the inference halts, and the branch's predicted class is selected. Otherwise, inference continues with the execution of the L_{n+1} layer.

B. Architecture of a DNN Accelerating System

FPGAs excel at accelerating DNNs due to their high parallel processing and customisation capabilities. These provide the ability to create hardware architectures tailored for DNN operations, facilitating highly optimized processing and low latency and making them indispensable for real-time applications. Moreover, FPGAs are energy-efficient, crucial for edge computing and embedded systems, and possess high memory bandwidth that is able to reduce data movement, overcoming common bottlenecks in deep learning. The large number of layers in DNNs, however, renders the mapping of entire networks onto a single FPGA very difficult. Instead, most FPGA accelerator designs use a layer-by-layer acceleration style [19], [20], [21]. A typical accelerator consists of the DNN accelerator, an external host processor, and off-chip memory. The DNN accelerator consists of modules for calculating convolutional and fully connected (FC) layers, storage for weights, inputs and outputs, and a controller. The programmability of FPGAs allows for full customisation of these modules, leading to designs that can adapt both the changing structures, sparsity, and convolution modes [22] and popular DNN compression algorithms (e.g. pruning [23]).

C. Early-exit Networks on FPGAs

Current research into realising early-exit dynamic DNNs on FPGAs focuses on exploring frameworks for automating the hardware design process. ATHEENA [24] is a framework that constructs a deeply pipelined, early-exit network accelerator for inference on FPGAs, focusing on resource and throughput trade-offs by leveraging the probabilistic nature of early exits. AdaPEx [25], a related approach, explores dynamic network inference on FPGAs using adaptive pruning and early exits to optimize resource and throughput trade-offs at runtime. Both

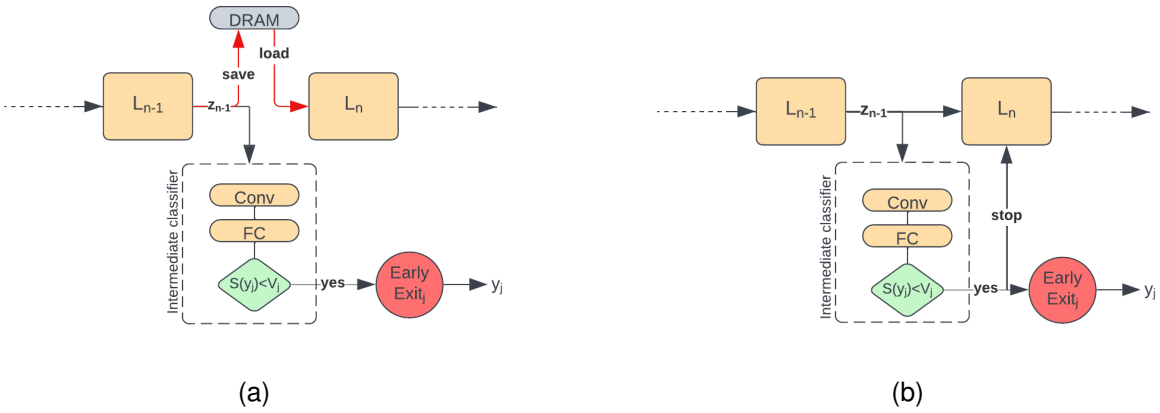

Fig. 2: Explored designs for (a) Pipeline and (b) Parallel approaches to Decision Early-Exit Networks.

approaches use frameworks (fpgaConvNet [26] and FINN [27] respectively) that generate model-specific designs. We aim to explore how the benefits of dynamic DNNs map to FPGAs, and a more general DNN accelerator that can support multiple models is necessary. Also, the efficiency of both approaches is limited by the dependencies on intermediate layers' outputs; we aim to resolve that by taking advantage of the FPGA parallelization capabilities.

III. FPGA Early-Exit Network Architectures

As discussed in Section II A, intermediate classifiers are responsible for making important data-dependent decisions that control the exit point of the network. They are also the key difference between normal and dynamic DNNs, and hence are the focus of this paper. They are small neural networks containing convolutional, fully connected, or pooling layers. For example in BranchyNet [9], the early exit modules consist of convolutional and FC layers alongside some activation functions, while on EPNet [28] and dynaexit [10] the convolutional layers are skipped. We propose and explore two approaches, *pipeline* and *parallel*, for efficiently realising early-exit dynamic DNNs on FPGAs.

A. Pipeline Design Approach

The *pipeline* approach (Fig. 2(a)) follows the traditional early exit architecture, where inference halts until the early exit decision is made. Inference is executed as normal until the layer before the intermediate classifier L_{n-1}. Then, it's paused, and the layer's output is fed to the branch's input. After executing the intermediate classifier and checking the confidence, the network either exits ($S(y_j) < V_j$) and requests a next input sample or the inference continues with the next layer L_n.

This architecture gives the opportunity of reusing the already designed IPs. With some careful setting of the control module on the FPGA design, an additional area overhead of close to 0% is achieved. This may lead to some underutilization of the existing components, but can be proven to be very useful for cases with very restricted resource.

However, using the intermediate output to make the early-exit decision generates an issue, as these data are lost after the intermediate classifier's execution. In order to maintain them, we can use registers or external memory. The use of registers is often not feasible, as these outputs can be very large, especially in deeper networks (4.7Mb on VGG19 [29], 1.6Mb on ResNet-32 [30]), while the use of external memory has a significant detrimental effect on the latency and memory footprint of inference.

B. Parallel Design Approach

FPGAs are inherently parallel devices, which means they can perform multiple operations simultaneously. They allow for a high degree of fine-grained parallelism, where multiple operations can be executed simultaneously at the gate level. To further exploit that, we propose an alternative *parallel* approach (Fig. 2(b)) that executes both the backbone and the intermediate classifier (L_n & $EarlyExit_j$) at the same time.

The output of the previous layer is fed to both the next layer of the backbone network and the intermediate classifier. They start executing simultaneously, but as the intermediate classifier is considerably smaller, it always finishes executing first. Dictated by the generated decision, the execution of the backbone either continues or stops and the network resets. This eliminates the latency overhead while computing the early exit branch but also, more importantly, the need to store the intermediate layer's output. However, the simultaneous activation of both paths requires the design of separate IPs for the intermediate classifier, increasing the area and power demands of the design.

C. Modeling the Energy-Latency Trade-off

To estimate the total energy consumption of the designs for the inference of I input samples, we use the following equation:

$$E_{inf} = \sum_{i=1}^{I} (P_{FPGA} \cdot T_i + E_{DRAM} \cdot M_i) \qquad (3)$$

where P_{FPGA} and T_i are the power consumption of the FPGA design and the total execution time of inference for the ith input. M_{i} is the number of Memory Accesses to the off-chip Dynamic Random Access Memory (DRAM), and can be calculated based on the systolic array size and the workload parallelisation approach presented in Section IV. Finally, E_{DRAM} is the typical per-bit access energy of DDR3 memory (70 pJ/bit[31]).

In early-exiting approaches however, the execution time is directly connected with every exit point's triggering rate. Essentially, this is the rate of the number of test inputs that trigger a target exit. This is dictated by one of the most influencing parameters in these dynamic approaches, which controls how harsh or lenient the decision to exit is. For example, in the target network BranchyNet, vector *V* is used to determine if an input i should exit at a specific exit point. It is calculated during training, using the proposed custom optimisation function, and controls the trade-off between accuracy and latency of the early-exit scheme. After training, the rate of samples that would trigger every exit point can be experimentally calculated, creating the following average per sample execution time $\overline{T}$ equation:

$$\begin{aligned}\overline{T} = a_1 \cdot (T_1 + \tau_1) + a_2 \cdot (T_1 + T_2 + \tau_1 + \tau_2) + ...+ \\ a_j \cdot (\sum_{i=1}^{j}(T_i + \tau_i)) + ... + a_J \cdot (\sum_{i=1}^{J}(T_i + \tau_i))\end{aligned} \tag{4}$$

where a_{j} is the rate of samples that exit in the jth exit point, J is the total number of the dynamic network's exits, T is the execution time of the backbone layers between the exit points, and τ is the execution time of each intermediate classifier. However, due to its different architecture, for the ***parallel*** approach, only the triggered exit point intermediate classifier execution time is added. So we can correctly rewrite equation (4) as:

$$\overline{T_{pipe}} = \sum_{k=1}^{J}(a_k \cdot \sum_{i=1}^{k}(T_i + \tau_i)) \tag{5}$$

$$\overline{T_{para}} = \sum_{k=1}^{J}(a_k \cdot (\sum_{i=1}^{k}(T_i) + \tau_k)) \tag{6}$$

The average execution time of the early-exit network executed on the ***pipeline*** approach is the sum of the rates a_k of every exit point multiplied by the sum of the execution time of the backbone network (T_i) and all intermediate classifiers up to that point (τ_i). For the ***parallel*** approach, the same function applies, but instead of adding the execution time of every intermediate classifier, we just add the one that led to the stop of inference τ_k. In this occasion, when no early-exit occurs, τ is 0.

Additionally to the difference in execution time, the ***parallel*** approach, as seen in Section III B, does not require storing the layer's output before an exit point like the ***pipeline*** approach, decreasing the memory access needs. Considering that we propose the following equations to calculate the energy consumption of the two designs:

$$E_{pipe} = \sum_{i=1}^{I}(P_{pipe} \cdot \overline{T_{pipe}} + E_{DRAM} \cdot (M_i + \sum_{k=1}^{i} M_{intk})) \tag{7}$$

$$E_{para} = \sum_{i=1}^{I}(P_{para} \cdot \overline{T_{para}} + E_{DRAM} \cdot M_i) \tag{8}$$

where M_{intk} is the memory access needs to store and load the intermediate layer output.

IV. Hardware Implementation

To investigate further the feasibility and benefits of dynamic DNNs on FPGAs and compare the two approaches introduced in the previous section, we developed a hardware implementation. It's essential first to convert real numbers like inputs, weights, and biases into a digital format. We use a fixed-point representation, as it minimizes the computational demands on hardware resources while maintaining the same levels of precision. This representation comprises three components: a sign bit, a two-bit integer, and a five-bit fractional value, which was found to minimise the computational demands and hardware resources while causing a very low accuracy drop (maximum of 1.3%).

An array of processing elements is used to accelerate matrix-vector multiplications. Systolic arrays are hardware structures that replace a pipeline structure with an array of homogeneous processing elements (PEs) that can perform a common mathematical operation. These elements are locally interconnected and able to communicate with each other in a synchronous manner. An architecture like this is very beneficial as it can naturally facilitate data reuse during matrix/tensor algebra operations. A significant portion of PEs in the array can perform their tasks without requiring communication with external memory. Consequently, a maximum of p I/O ports has the capacity to drive p^2 PE units, which substantially reduces in memory bandwidth demands.

Fig. 3(a) provides an overview of the design of the proposed hardware implementation. The core of the design is a 2D systolic array configuration containing a collection of identical and interconnected PEs accompanied by onboard memory/registers. Each PE (Fig. 3(b)) is constructed from fundamental arithmetic and register components, enabling it to execute multiply-accumulate operations. The data flows from the Input Buffer into the PEs (weights through regW and features through regI) and is multiplied and accumulated in the MAC module. Partial sums are calculated using bufc, and when all iterations are completed, the results are stored in the Output Buffer.

To improve the performance of the systolic array, it is necessary to consider how to explore the parallelism of the workload to drive multiple PEs to work simultaneously. A simple way is loop unrolling [32]. Algorithm 1, for example, describes the calculation process of a FC Layer, where d denotes the size of the mini-batch, c signifies the cth input, and g represents the gth output. We can flatten loop1 and

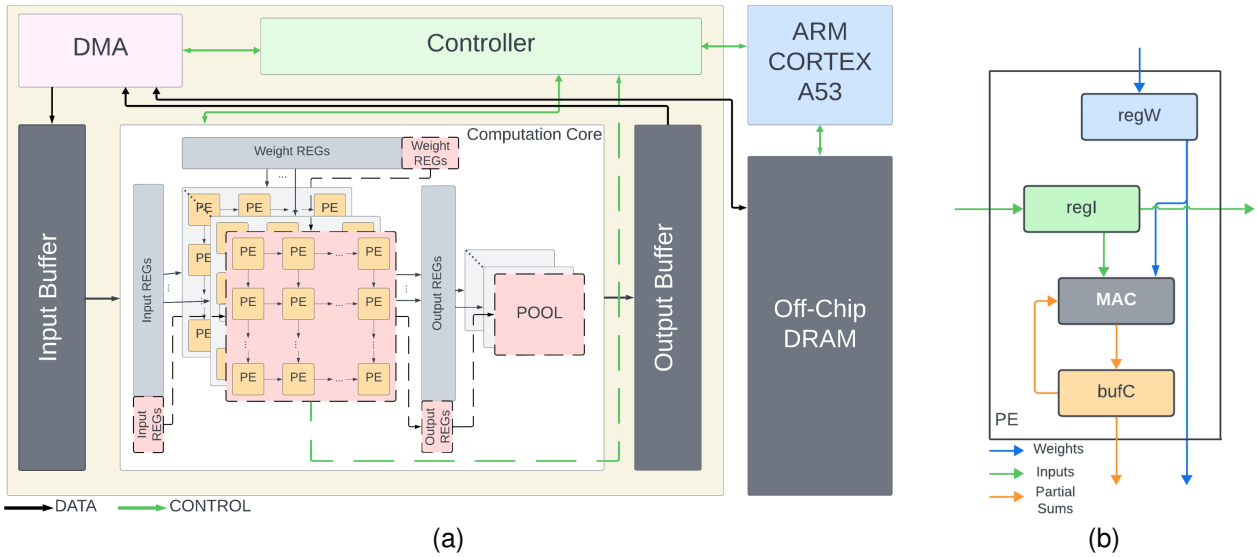

Fig. 3: Design overview of (a) the system and (b) the processing element.

Algorithm 1 3-nested loop of fully-connected (FC) layer

Input: $I[D, C]$ and $W[E, C]$
Output: $O[D, G]$

1: **for** $(d = 0;\ d < D;\ d++)$ **do**
2: **for** $(g = 0;\ g < G;\ g++)$ **do**
3: **for** $(c = 0;\ c < C;\ C++)$ **do**
4: $O[d, g] = O[d, g] + W[g, c] \cdot I[d, c]$

loop2 to the spatial hardware and compute these loop instances in parallel. To better describe this, we can use data-flow to represent which loops in the workload are unrolling. We are following the Output Stationary (OS) [33] approach, which refers to mapping output elements to the corresponding PEs, where each PE needs to complete all the computations required for an output element.

The rest of the design contains a controller module, a DMA (Direct Memory Access) module, a host processor and an Off-Chip DRAM. The controller ensures the synchronisation of the different processing units and coordinates the sequence of the dynamic DNN layer execution. It also communicates with the host processor, which for most networks is used to set-up the FPGA and post-process its results. In the case of ResNet-32, it is used to execute the model, too. Finally, the DMA is responsible for the efficient data movement between the accelerator buffers and the Off-Chip DRAM.

A. Layer realisation

1) Convolution Layer: To implement convolution, input data is fed from the buffers into the Systolic Array, as explained above. The PEs calculate input-kernel multiplication, and outputs are then moved to another buffer. The activation function (ReLU), a simple conditional branch, is promptly applied, and its results are stored in buffers that feed the subsequent layers.

2) FC Layer: The architecture of fully connected (FC) layers is akin to that of convolutional layers, but instead of using convolution, they rely on vector-matrix multiplication. The input vectors consist of numerous values generated by flattening the output of the previous layer, resulting in many multiplication operations. To handle this, the inputs and weights are divided into equal segments and computed separately.

3) Pooling Layer: Pooling layers take the values stored in buffers from the preceding layer and apply a sliding window with a size equal to the pooling filter and a step size determined by the specified stride value. The operation performed is either max or average pooling.

V. Experimental Results and Analysis

A. Experiment Setup

We implemented the dynamic network accelerator outlined in the previous section using VHDL on a Xilinx FPGA board for four different DNNs. To verify and compare the performance of the FPGA design the networks are also executed, using PyTorch, on a desktop computer with an Intel Core i9 9960X CPU and a Nvidia RTX 2080 TI GPU, and on the Nvidia Jetson Xavier NX embedded platform.

1) FPGA Setup: The target platform is Zynq UltraScale+ MPSoC ZCU106 Evaluation Kit, including 504K logic cells, 1728 digital signal processing (DSP) slices, 38Mb of Block RAM and one 4Gb DDR4 module. Table I compares the resource utilisation for the two architectures.

To calculate the latency and power consumption of the FPGA design, Xilinx Vivado Design Suite is utilized. For precise latency measurements, an Integrated Logic Analyzer (ILA) monitors selected internal signals in real-time. Power consumption is estimated using Vivado's Power Estimation tool, which uses a Simulation Activity file (SAIF) specifying environmental conditions, voltage levels, and signal activity rates. This enables the tool to analyze the design and provide a power estimate after implementation that has a $\pm 10\%$ difference over the actual value [34].

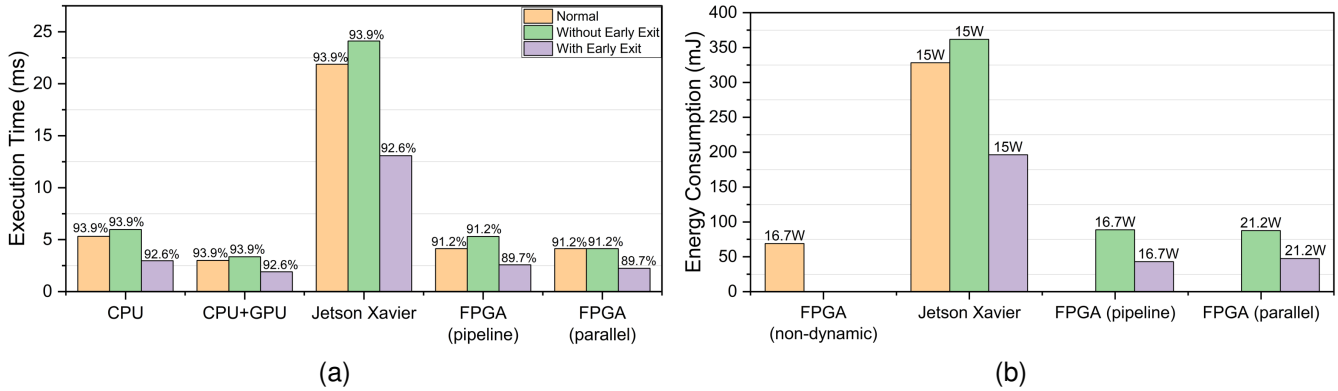

Fig. 4: Experimental results comparing average (a) Execution Time and (b) Energy Consumption per sample. Average values are calculated based on each exit point trigger rate (a_i).

TABLE I: Resource Utilisation

Design	Data	#PEs	FF	LUT	BRAM	DSP	Clock
Pipeline	8-bit	300	50%	66%	96%	78%	150 Mhz
Parallel	8-bit	300	58%	73%	89%	81%	150 Mhz

2) DNN Networks: We experiment with four of the most popular DNNs: LeNet-5 [35] , VGG19 [29], AlexNet [1] and ResNet [30]. The networks are modified to add early exits using the structure of BranchyNet [9], an open-source and popular early exit network. Explicitly, on LeNet-5 one branch consisting of 1 convolutional layer and a fully-connected layer is added after the first convolutional layer (L_1) of the main network. On AlexNet, one branch consisting of two convolutional layers and a fully-connected layer after the 2nd convolutional layer (L_2) of the main network. Finally on VGG19 and ResNet two branches are added containing a convolutional and a fully-connected layer, after the 2nd (L_2) and 10th (L_{10}) convolutional layers on VGG19 and after the 2nd (L_2) and 15th (L_{15}) convolutional layers on ResNet32. We have to note that for the execution of the ResNet32 network we employ an external host processor, while the rest of the networks are deployed entirely on the FPGA.

3) Datasets: We benchmark and compare our designs using three widely used and open-source datasets: MNIST [36], CIFAR-10 and CIFAR-100 [37]. MNIST contains 60,000 training images and 10,000 testing images, each of which is a 28x28 pixel gray-scale image of a handwritten digit. The two CIFAR datasets contains 32x32 pixel RGB natural images across 10 and 100 classes respectively with, 50,000 training and 10,000 validation samples.

B. Design Validation and Evaluation

To validate and evaluate the FPGA architectures for the two designs we deployed the four widely used DNNs described above, trained based on the BranchyNet [9] approach. For brevity Fig. 4(a) shows the per sample execution time of VGG19 on the Cifar-10 dataset, both for the original static (orange bars) and the dynamic network (green and purple bars). The green bars show the time needed to execute the early-exit network when the last exit is triggered, and comparing it with the static network (orange bars), highlights the additional latency introduced by the intermediate classifiers.

However, dynamic inference exits for 43% (a_1) of the samples on the first exit point, 45.6% (a_2) on the second and 11.4% (a_3) on the original output. The average execution time ($\overline{T}$) for the 10k test samples is portrayed on the 4(a) (purple bars). As illustrated by the difference of the Normal and Early Exit bar, and explained by equations (5) and (6), this leads to an execution speedup by at least 1.4x in all platforms, while for the FPGA approaches the dynamic network is at least 1.9x. Furthermore we see that the accuracy (top of bars) of the network is barely affected, being only 1.5% less than the original static network.

The effects on execution time of Dynamic DNNs however, also impacts their energy demands (equations (7) and (8)). Fig. 4(b) shows the per-sample energy consumption (E_{inf}) of VGG19 both for the original static (orange bars) and the dynamic network (green and purple bars) on the Jetson and the FPGA. Correspondingly to the execution time, in cases where an early exit does not occur, energy consumption of the dynamic network is higher than the static's. However, when the dynamic network is properly trained, the effects of early-exiting lead to a reduction in energy consumption over a static network by at least 1.4x on the Jetson and 1.8x on the FPGA. As illustrated on Fig. 4(b) the FPGA is able to achieve comparatively low energy consumption (at least 4x less than the Jetson), highlighting the capabilities and effectiveness of a custom hardware design.

We also assessed the performance against other FPGA-based DNN accelerators, as illustrated in Table IV. All of the designs employ systolic arrays of process elements and explore different techniques to accelerate DNNs on FPGAs, e.g. Winograd-GEMM [38], and pruning [39], [25]. Despite the fact that the targeted early-exit networks maintain the backbone network unchanged and actually burden it with the intermediate classifiers, the proposed accelerator achieves comparative power and better energy efficiency than other designs, further highlighting the benefits of dynamic approaches. The ATHEENA framework [24] achieves considerably lower latency, explained by the fact that it generates custom designs

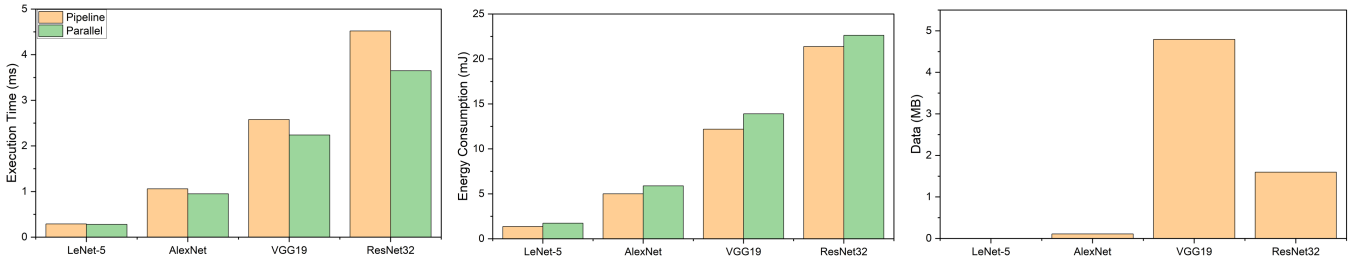

Fig. 5: Comparison of the ***pipeline*** and ***parallel*** designs over average (a) execution time, (b) energy consumption and (c) data movement across early-exit LeNet-5, AlexNet, VGG19 and ResNet32.

TABLE II: Execution Time (ms)

	LeNet-5			AlexNet			VGG19				ResNet32			
	Exit 1 (94.4%)	Exit 2 (5.6%)	Avg	Exit 1 (75.6%)	Exit 2 (24.4%)	Avg	Exit 1 (43%)	Exit 2 (45.6%)	Exit 3 (11.4%)	Avg	Exit 1 (48.4%)	Exit 2 (18.3%)	Exit 3 (33.3%)	Avg
CPU	0.3	1.32	0.36	0.75	3.3	1.38	1.92	3.21	5.98	2.97	2.54	8.24	12.74	6.98
CPU-GPU	0.15	0.62	0.17	0.38	1.55	0.66	1.28	2.14	3.35	1.91	1.37	3.53	5.75	3.22
Jetson	0.71	5.05	0.95	1.78	12.63	4.43	8.98	14.2	24.12	13.08	9.9	15.62	26.53	16.48
Pipeline	0.24	0.99	0.29	0.6	2.48	1.06	1.49	2.99	5.31	2.58	1.84	5.1	9.4	4.95
Parallel	0.24	0.82	0.28	0.6	2.05	0.95	1.49	2.48	4.13	2.24	1.84	4.54	7.89	4.01

TABLE III: Energy Consumption (mJ), with (w/ EE) and without (w/oEE) early exits

	FPGA - Static (16.7W)		Nvidia Jetson Xavier			**Pipeline (16.7W)**		**Parallel (21.2W)**	
	w/o EE	w/ EE	w/o EE	w/ EE	Power (W)	w/o EE	w/ EE	w/o EE	w/ EE
LeNet-5	16.5	n/a	48.5	9.1	9.6	13.7	4.8	17.4	5.9
AlexNet	34.2	n/a	142.7	50.1	11.3	41.4	17.7	45.5	21.1
VGG19	68.9	n/a	361.8	196.2	15	88.7	43.1	87.6	47.5
ResNet-32	99.3	n/a	398	247.2	15	143.82	75.7	141.8	85

for each network. Conversely, our approach targets the implementation of a general early-exit dynamic DNN accelerator. In fact, there is still adequate opportunity to enhance the accelerator's performance. We observe that DSP utilization is relatively low due to insufficient block RAM resources. By optimizing the storage structure or utilizing a platform with more resources, the processing element array can be expanded to enhance DSP utilization, thereby achieving higher throughput. Furthermore combining dynamic approaches with acceleration techniques, such as the ones used by the cited designs, latency and resource demands could be further reduced.

C. Evaluating pipeline vs parallel Approaches

Table II contains the average per sample execution time ($\overline{T}$) for each network and for every exit point, alongside the percentage of samples (a_i) that would trigger them (underneath every exit). Focusing on the FPGA architectures and comparing them with the rest of the platforms, we see that their performance is competitive compared to the CPU/CPU-GPU, and much better than the Jetson. We also notice the benefits of executing the intermediate classifiers in parallel with the backbone networks, as the ***parallel*** approach is constantly faster than the ***pipeline***.

To further explore and characterise the two proposed designs we compare their latency, energy and memory demands. Fig. 5(a) shows the average execution time ($\overline{T}$) of the deployed early-exit networks on the two approaches. The aforementioned ***parallel*** advantages are more pronounced on deeper networks; the latency of LeNet-5 is improved by 3.7%, while VGG19 and ResNet32 are improved by 15.2% and 23.9%, respectively. This is because the design avoids halting the backbone network, which is very effective in cases where there are many samples that would require the execution of deeper layers. That occurs when more complex data sets with a higher number of non-canonical samples are targeted. Also, deeper networks usually consist of larger layers with bigger feature maps and more filters. This unavoidably affects the intermediate classifiers, too, increasing their size and, consequently, the required computation. Conversely, smaller networks targeting simpler data sets have a high proportion of samples triggering exits in shallow layers, reducing the effectiveness of parallel execution.

Table III shows the energy per sample for every network with and without early exiting. Based on the previously stated rates of early-exiting at each branch, it can be seen that the average energy consumption is always at least 25% lower than the static network on the embedded device and almost 50% on the FPGA (E_{inf}). We also see that the ***parallel*** approach on average, is at least 15% more energy (Fig. 5b) demanding on smaller networks than deeper networks. This is explained by the same reasons elaborated on the previous paragraph. In simpler networks, fewer samples require inference at deeper layers, while the size of the intermediate classifiers are proportional to the size of the backbone, rendering the ***parallel*** approach less effective.

Furthermore, as explained above, the ***pipeline*** approach

TABLE IV: Performance Comparison With Existing Implementations

	[38]	[40]	[41]	[39]	[24]	[25]	Ours (pipeline)	Ours (parallel)
DNN model	AlexNet/ VGG16	ResNet50	AlexNet/VGG16 /ResNet50	LeNet/AlexNet /VGG16	LeNet/ AlexNet	AdaPEx (Cifar-10)	LeNet/AlexNet/ VGG19/ResNet32	LeNet/AlexNet/ VGG19/ResNet32
Device	FPGA VX690T	FPGA ZC706	FPGA ZCU102	FPGA XCVU9P	FPGA ZC706	FPGA ZCU104	FPGA ZCU106	FPGA ZCU106
Frequency (MHz)	200	200	200	300	125	100	150	150
Precision	Fixed 16	Fixed 16	Fixed 16	Fixed 8	Fixed 8	2	Fixed 8	Fixed 8
Power (W)	17.3	6.23	23.6	-	-	1.26	16.7	21.2
Performance (GOPs)	433.6/ 407.2	107.9	2068.5/2641 /1240	1490.5/913.9 /862.2	-	-	783.1/911.3 /1732.5/185.1	811.9/1009 /2011.6/243.9
DSP Efficiency (GOPs/DSP)	0.30/0.28	0.24	1.81/2.31/1.08	3.88/1.79/1.68	-	-	0.78/0.96 /1.82/0.20	0.66/0.82 /1.64/0.19
Latency (ms)	-	-	-	-	0.01/0.06	3.52	0.29/1.06 /2.58/4.95	0.28/0.95 /2.24/4.01

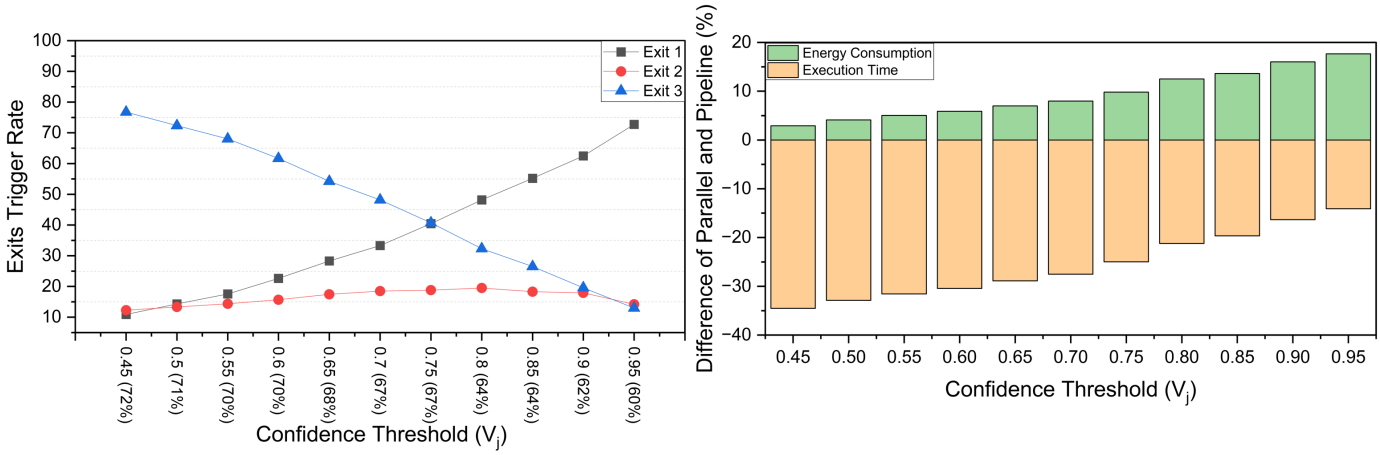

Fig. 6: On an 3 point early-exit Resnet-32 a) shows each exit's trigger rate and b) the percentage difference of *parallel* over *pipeline* approaches over Energy and Time for different Confidence Thresholds.

requires the storing of the intermediate layer's output (M_{intk}); the total data for each networks can be seen in Fig. 5(c). We notice that it is considerably larger on deeper, more complex networks. While smaller networks like LeNet-5 and AlexNet, the data can be buffered on-chip, VGG19 and ResNet32 can reach up to 4MB, rendering the use of external memory mandatory. This significantly affects the latency and energy demands of the ***pipeline*** approach, and illustrates the effectiveness of the ***parallel*** approach on deeper networks. Overall, however, for smaller networks, the extra energy demands that are introduced in the ***parallel*** approach overshadow its benefits, making the ***pipeline*** the better design.

D. Effect of Decision Threshold on Performance

Section III C. presented the modelling of the two approaches regarding energy consumption and execution time. The importance of the decision threshold (V) is highlighted, as it controls the effects of early-exiting in the DNN. The number and the location of the intermediate classifiers also have significant effects on the behaviour and exiting rates of the dynamic DNN; however, an exploration of this is outside the scope of this paper, and interested readers are directed to [42]. Fig. 6(a) shows the triggering rates (a_i) of 10k samples on the three exit points of early exit ResNet32, alongside the achieved accuracy. For higher confidence thresholds, most of the samples would trigger the first exit point (black line). Decreasing the confidence threshold results in a significant decrease in the number of samples that would exit on the first exit point (black line), while those that would exit on the third and final exit increase (blue line). Apart from affecting the exit point rates, by changing the threshold, we essentially change the amount of abstracted information needed to make a prediction. This consequently affects the accuracy of the network, too. As shown in Fig. 6(a), higher threshold values lead to accuracy drops, while the opposite happens when they are lower. This is explained by the fact that more samples exit at an earlier stage without the network being able to abstract enough information to make a correct classification.

Following equations (5) - (8), we also see that, by changing the thresholds, we can control the latency and energy demands of the network. Fig. 6(b) shows the difference between the two approaches over execution time and energy consumption. We see that while using higher thresholds, the ***parallel*** approach is considerably more energy-demanding without similar gains in execution time. On the contrary, when the threshold values are lower, deeper layers are executed more often, resulting in reducing the energy consumption difference between the two approaches and increasing the time difference, making the ***parallel*** approach considerably faster. Consequently, we see that, apart from the dynamic network architecture, changing the threshold of the decision to exit controls the accuracy-latency trade-off and affects the effectiveness of the designs

realising it. Overall, we see that using equations (5) - (8) and controlling the thresholds of the early-exit decision, the trade-offs between accuracy, latency and resource needs can be explored and help decide the best design approach for any application needs.

VI. Conclusions

This paper has presented the design and in-depth evaluation of early-exit dynamic DNNs on FPGAs. Two alternative approaches to the design of intermediate classifiers were explored: pipeline and parallel. Both approaches, when compared with conventional systems and an embedded device, showcased faster execution and lower energy consumption, highlighting the potential benefits of using FPGAs, especially when targeting resource-constrained application scenarios. Furthermore, the parallel approach showed how the parallelization capabilities of FPGAs can be used to eliminate the additional latency and memory requirements of intermediate layer dependencies of early-exit networks. The two approaches were compared over latency, energy consumption and memory use on four different DNN models, illustrating how their characteristics affect the hardware performance. An exploration of the effects of the decision threshold on the FPGA accelerators was also presented, which, in combination with the design comparison outcomes, provides suggestions for how designers might select the most appropriate implementation for a target application and design constraints.

Acknowledgments

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) under EP/S030069/1. Data associated with this paper is available on https://doi.org/10.5258/SOTON/D3174.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012.

[2] R. Joshi and V. Kannan, "Attention based end to end speech recognition for voice search in hindi and english," in *FIRE*, 2022.

[3] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *ICLR*, 2018.

[4] Z. Chen, Y. Li, S. Bengio, and S. Si, "Gaternet: Dynamic filter selection in convolutional neural network via a dedicated global gating network," *in CVPR*, 2018.

[5] L. Yang, Z. He, Y. Cao, and D. Fan, "Non-uniform dnn structured subnets sampling for dynamic inference," in *DAC*, 2020.

[6] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin, "DeeBERT: Dynamic early exiting for accelerating BERT inference," in *ACL*, 2020.

[7] R. Schwartz, G. Stanovsky, S. Swayamdipta, J. Dodge, and N. A. Smith, "The right tool for the job: Matching model and instance complexities," in *ACL*, 2020.

[8] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei, "BERT loses patience: Fast and robust inference with early exit," in *NeurIPS*, 2020.

[9] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," *ICPR*, 2016.

[10] M. Wang, J. Mo, J. Lin, Z. Wang, and L. Du, "Dynexit: A dynamic early-exit strategy for deep residual networks," in *SiPS*, 2019.

[11] A. Dimitriou, M. Hu, J. Hare, and G. V. Merrett, "Exploration of decision sub-network architectures for fpga-based dynamic dnns," in *DATE*, 2023.

[12] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, 2021.

[13] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *ICANN*, 2014.

[14] P. Dhilleswararao, S. Boppu, M. S. Manikandan, and L. R. Cenkeramaddi, "Efficient hardware architectures for accelerating deep neural networks: Survey," *IEEE Access*, 2022.

[15] S. Mittal, "A survey of fpga-based accelerators for convolutional neural networks," *Neural Comput. Appl.*, 2020.

[16] M. Putic, S. Venkataramani, S. Eldridge, A. Buyuktosunoglu, P. Bose, and M. Stan, "Dyhard-dnn: Even more DNN acceleration with dynamic hardware reconfiguration," in *DAC*, 2018.

[17] E. Park, D. Kim, S. Kim, Y. Kim, G. Kim, S. Yoon, *et al.*, "Big/little deep neural network for ultra low power inference," in *CODES+ISSS*, 2015.

[18] L. Yang, Y. Han, X. Chen, S. Song, J. Dai, and G. Huang, "Resolution adaptive networks for efficient inference," in *CVPR*, 2020.

[19] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, *et al.*, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Trans. on CAD.*, 2018.

[20] H. Fan, M. Ferianc, Z. Que, S. Liu, X. Niu, M. R. D. Rodrigues, *et al.*, "Fpga-based acceleration for bayesian convolutional neural networks," *IEEE Trans. on CAD.*, vol. 41, 2022.

[21] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ASPLOS*, 2014.

[22] S. Ryu, Y. Oh, and J.-J. Kim, "Mobileware: A high-performance mobilenet accelerator with channel stationary dataflow," in *ICCAD*, 2021.

[23] Y. Liang, L. Lu, and J. Xie, "Omni: A framework for integrating hardware and software optimizations for sparse cnns," *IEEE Trans. on CAD.*, vol. 40, 2021.

[24] B. Biggs, C.-S. Bouganis, and G. Constantinides, "Atheena: A toolflow for hardware early-exit network automation," in *FCCM*, 2023.

[25] G. Korol, M. G. Jordan, M. B. Rutzig, J. Castrillon, and A. C. S. Beck, "Pruning and early-exit co-optimization for cnn acceleration on fpgas," in *DATE*, 2023.

[26] S. I. Venieris and C.-S. Bouganis, "Fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, 2019.

[27] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. H. W. Leong, M. Jahre, *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," *FPGA*, 2017.

[28] X. Dai, X. Kong, and T. Guo, "Epnet: Learning to exit with flexible multi-branch network," in *IKM*, 2020.

[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[31] J. S. S. T. Association. "Jesd-79-3d: Jedec standard for three-dimensional integrated circuits (3dic) stacking." (), [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd-79-3d.

[32] H. T. Kung, "Why systolic architectures?" *Computer*, vol. 15, 1982.

[33] A. Samajdar, J. M. Joseph, Y. Zhu, P. N. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of DNN accelerators using scale-sim," in *ISPASS*, 2020.

[34] Xilinx Inc., *Vivado Design Suite User Guide: Power Analysis and Optimization (UG907)*, 2021.

[35] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, 1998.

[36] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, 2012.

[37] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[38] S. Kala, B. R. Jose, J. Mathew, and N. Sivanandan, "High-performance cnn accelerator on fpga using unified winograd-gemm architecture," *IEEE Transactions on VLSI*, 2019.

[39] C. Yang, Y. Meng, K. Huo, J. Xi, and K. Mei, "A sparse cnn accelerator for eliminating redundant computations in intra- and inter-convolutional/pooling layers," *Transactions on VLSI*, vol. 30, 2022.

[40] X. Li, H. Huang, T. Chen, H. Gao, X. Hu, and X. Xiong, "A hardware-efficient computing engine for fpga-based deep convolutional neural network accelerator," *Microelectron. J.*, vol. 128, 2022.

[41] Y. Liang, L. Lu, Y. Jin, J. Xie, R. Huang, J. Zhang, *et al.*, "An efficient hardware design for accelerating sparse cnns with nas-based models," *Trans. on CAD*, 2021.

[42] S. Laskaridis, A. Kouris, and N. D. Lane, "Adaptive inference through early-exit networks: Design, challenges and directions," *EMDL*, 2021.