FPGA Acceleration of Dynamic Neural Networks: Challenges and Advancements

Anastasios Dimitriou
School of Electronics and Computer Science
University of Southampton UK
ad1r20@soton.ac.uk

Benjamin Biggs
Department of Electrical and Electronic Engineering
Imperial College London UK
benjamin.biggs15@imperial.ac.uk

Jonathon Hare & Geoff V. Merrett
School of Electronics and Computer Science
University of Southampton UK
{gvm, jsh2}@ecs.soton.ac.uk

***Abstract*—Modern machine learning methods continue to produce models with a high memory footprint and computational complexity that are increasingly difficult to deploy in resource constrained environments. This is, in part, driven by a focus on costly, power-intensive GPUs, which has a feedback effect on the variety of methods and models chosen for development. We advocate for a transition away from the general purpose processing towards a more targeted, power-efficient, form of hardware, the Field-Programmable Gate Array (FPGA). These devices allow the user to programmatically tailor the model processing architecture, resulting in increased inference performance and lower power demands. Their resources however are limited, which leads to the necessity of simplifying the target deep machine learning models. Dynamic Deep Neural Networks (DNNs) are a class of models that go beyond limits of static model compression, by tuning computational workload to the difficultly of inputs on a per-sample basis. In spite of the model simplification capabilities of Dynamic DNNs and the provable efficiency of FPGAs, little work has been done towards accelerating Dynamic DNNs on FPGAs. In this paper we discuss why this occurs by highlighting the challenges and limitations, both at the software and hardware level. We detail the available efficiency, performance gains, and practical benefits of state-of-the-art Dynamic DNN implementations when FPGAs are adopted as the acceleration device. Finally, we present our conclusions and recommendations for continued research in this space.**

I. Introduction

DNNs have become popular due to their impressive performance across a range of Artificial Intelligence (AI) applications, including computer vision [1] and natural language processing [2]. Much of the efficacy of modern DNNs stems from increased capacity afforded by larger architectures. This often requires increasing the network's depth, resulting in higher computational demands. Traditional CPU platforms struggle to efficiently process DNN models at acceptable speeds for tasks like object detection that demand real-time processing. GPUs provide a solution in some circumstances. They provide massive potential for parallel computation that can speed up matrix-vector operations, the majority of computations in DNNs. However, these devices typically have prohibitively high power demands [3] for applications targeting embedded systems, IoT, etc.

For the low resource setting, custom hardware architectures can be developed specifically for DNNs, tailoring processing elements to the required computational characteristics.

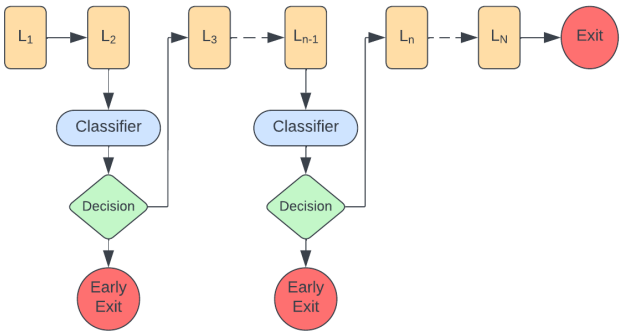


Fig. 1: Example of a Dynamic DNN architecture: Early-Exit Network.

The flexible nature and parallelization capabilities of Field-Programmable Gate Arrays (FPGAs) position them as strong contenders for accelerating neural network operations, while maintaining very low power consumption. However, modern DNN complexity makes their deployment very challenging. Large networks can contain approximately 40 billion operations [2] and hundreds of millions of parameters, for the inference of a single 224×224 image. Given these substantial computational and storage demands, attempting to map them onto FPGAs without compression is not feasible.

To reduce the inference workload and memory footprint of DNNs, various methods focus on parameter reduction. Quantization reduces data precision, while pruning [4] is an approach that removes a subset of weights and activations, shrinking model size at the cost of representation power and potentially requiring complex computations with sparse matrices. Knowledge distillation [5] transfers knowledge from a larger model to a smaller one and advancements in Neural Architecture Search [6] further reduce parameter redundancy. All of these approaches have a uniform effect on the samples processed during inference, which, intuitively, causes the greatest reduction in accuracy on the most difficult samples for a given task.

Dynamic DNNs [7] are approaches that strategically distribute computational workload based on their ability to modify network structure and parameters during the inference process. The fundamental concept is that different inputs have different computational demands. Dynamic methods exploit these differing computational demands, leading to a substantial reduction in computations, improvements in efficiency,

and compactness of the networks. The adaptability of these networks optimizes model performance for different inputs and resource constraints whilst maintaining accuracy. As such, Dynamic DNNs provide a compelling architectural choice for resource constrained applications.

Despite the efficiency and low power demands of FPGAs, and the compression capabilities of Dynamic DNNs, only small research effort has been put towards accelerating and realising them on this platform. The aim of this paper is to explore the reasons for this and to motivate the continued research of Dynamic DNNs on FPGAs. We outline the challenges of current software and hardware infrastructure in the support of dynamic characteristics and resource constrained devices. Furthermore, we present current advancements in this field, and state-of-the-art implementations. Lastly, we conclude and identify open research problems.

II. Related Work

To exemplify the practical benefits of Dynamic DNNs and FPGAs in a low resource setting, we survey existing advancements in both the mapping of standard DNNs onto FPGAs, and the implementation of dynamic methods to reduce DNN computational workload.

A. FPGAs

FPGAs are devices featuring a versatile array of configurable logic elements, highly optimized Digital Signal Processing (DSP) blocks, and flexible on-chip SRAM blocks (Block RAM, URAM), all interconnected by adaptable routing. The strategic arrangement of these components, coupled with the flexible routing capabilities, empowers FPGA engineers to leverage inherent parallelism within deep learning models. Their flexibility, task specificity, and efficiency are key motivators for FPGA use in DNN acceleration. However, FPGAs typically require device domain knowledge to make effective use of these benefits due to the complexities of tooling and required fine tuning of algorithm implementation. There are many tools that can map high-level expressions of DNNs directly to the device [8]. Examples include fpgaConvNet [9], a toolflow that can take a pre-trained CNN model, device specification, and user throughput requirements to construct an optimised hardware architecture for efficient inference. Similarly, HPIPE [10] is a framework for accelerating DNN inference on server-scale FPGAs, leveraging pruned network sparsity to increase the efficient utilisation of FPGA resources. Both frameworks utilise the streaming dataflow paradigm whereby inputs can be streamed in consecutively and operated on in parallel through the use of pipelining. This drastically increases throughput without exhausting memory bandwidth.

B. Network Compression Methods

DNNs are computationally and memory intensive applications and mapping modern architectures to FPGA devices without compression can be a difficult or even impossible task. Typical methods to reduce the inference workload and memory footprint of a DNN focus on parameter reduction. Quantization is the process by which the precision of the DNN weights and activations is reduced from the standard IEEE floating point format. Frameworks like FINN [11] and hls4ml [12] focus on the minimization of precision by reducing the width of parameters and operators in standard convolution and linear down to 3 bits or lower. For FPGAs, this enables the packing of multiple operations into a DSP block for performance gains. Alternatively, custom logic operations can reduce resource utilization [13]. Logic element and LUT based tools take this a step further by incorporating complex DNN function behaviour within the discrete building blocks of FPGA programmable logic [14]–[16], leading to low latency and resource utilization for resource-constrained DNN tasks. Additional fine tuning is often employed to minimise the impact on accuracy [17]. Using a higher precision such as 8 or 16 bit integer and fixed-point representations can mitigate accuracy drops and still achieve workload reduction.

Pruning a DNN is the process through which a subset of weights and activations is removed [4]. This results in a significant reduction in the model parameters [18] but also in permanently removing network parameters, leading to losses in representation power. Fully unstructured pruning can reduce a high percentage of network parameters but the random nature of the removed parameters often necessitates the computation of large, sparse matrix multiplications. For GPU computation, this can result in performance reductions, but FPGA architectures can be tailored to efficiently perform sparse operations, as with HPIPE.

Knowledge distillation [5] is a popular method to produce compact networks with the accuracy of larger-scale networks. During the training phase a smaller, simpler model (often referred to as the student model) is trained to mimic the behavior of a larger, more complex model (often referred to as the teacher model). The objective is to transfer the 'knowledge' learned by the teacher model to the student model in a more compressed form. These methodologies, coupled with the increasing use of Neural Architecture Search [6] and development of smaller hand-tuned network architectures [19] means that models are becoming less redundant. To continue improve efficiency and performance in this case, dynamic methods of parameter reduction are employed.

C. Dynamic DNNs

Unlike static DNNs, which have fixed architectures and parameters throughout training and deployment, Dynamic DNNs adapt their structures or parameters at runtime in response to input data or system changes. This adaptability allows dynamic approaches to allocate computational resources more efficiently, scaling up or down based on the complexity of the task or the available computational resources. This dynamic computational resource allocation enhances the inference efficiency of the network, improving performance whilst maintaining accuracy on large-scale datasets and complex tasks beyond what is capable with static methods alone. Furthermore, static network compression techniques like pruning or quantization typically occur during training or as a post-processing step. Dynamic methods can accommodate changes

in data characteristics or task requirements over time, leading to improved performance and robustness in real-world applications [20], [21]. By facilitating continual learning, they can adapt to new tasks or data distributions incrementally while retaining knowledge learned from previous experiences. This capability is particularly valuable in applications where the model needs to continuously learn from new data streams or adapt to evolving tasks, such as in online learning, adaptive systems, and autonomous agents.

In this paper we focus on dynamic depth approaches and more specifically in *early-exit* dynamic networks. Early-exiting is a structure-focused approach that enables inference to stop at an earlier stage, when the network is confident enough to produce an output. Early-exit networks consist of a backbone architecture, additional exit points or classifiers distributed throughout its depth (Fig. 1), and a method for quantifying task confidence. During the inference process, as a sample traverses the network, it passes through both the backbone and each exit in a sequential manner. The prediction output is determined by the exit that satisfies a predefined criterion, known as the exit policy. For example, an 'easy' input sample will confidently exit early, bypassing the remainder of the model. Training networks with multiple exits is a common practice observed in early-exiting dynamic networks [22], [23], where the primary focus lies in minimizing a weighted cumulative loss incurred by intermediate classifiers. We note that architectural choices can depend on factors such as target device capabilities, and workload. A given backbone can be adapted to fulfill specific runtime requirements [24], [25].

III. Dynamic DNN & FPGA Acceleration Challenges

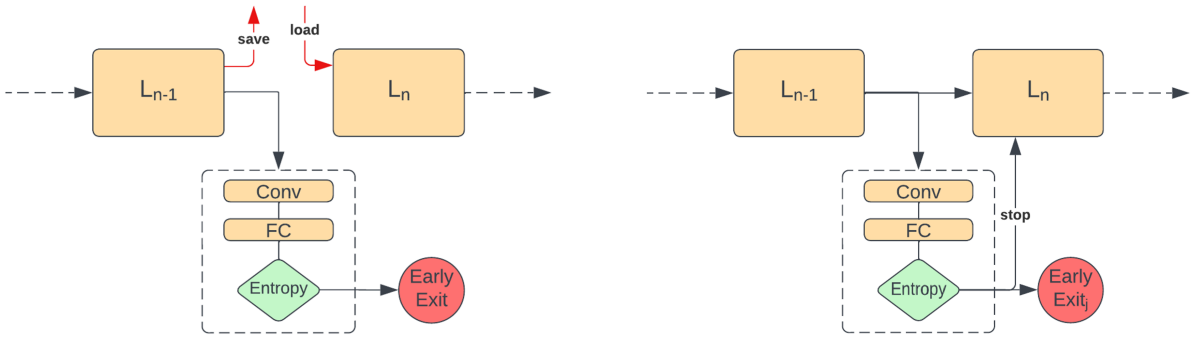
Integrating intermediate classifiers introduces complexity and additional computational load. In most cases, these components consist of standard DNN layers such as convolutional, fully-connected, activation, and pooling layers. From a computational perspective, these layers are required to rapidly down-sample the intermediate feature maps from shallower sections of the DNN in order to feed a classification layer (e.g. Softmax) with the appropriate number of logits. From a functionality perspective, these layers are tasked with coalescing the fine-grained features of shallower layers and making task decisions based on a more limited receptive field. In BranchyNet [26], the early-exit modules include convolutional and fully connected layers along with ReLU activation functions, while in EPNet [27] and DynExit [28], the convolutional layers are omitted. Through hand-crafted architecture design and training, a high percentage of samples trigger an early exit at shallow layers. BranchyNet achieves 90.2% of inputs to exit after the 2nd layer of an early-exit AlexNet, and more than 57% to exit after the 32nd on a early-exit ResNet-110. However training and deploying dynamic networks can be proven a very challenging task as current deep learning hardware and libraries are optimized for static models, and they can be proven very unfriendly towards their different architectures. For example Tensorflow uses frozen graphs which refers to a trained model that has been serialized into a single file containing both the model's architecture and trained parameters. This serialization process 'freezes' the graph, meaning that the architecture and model's parameters become constants and cannot be further trained or modified. In addition, dynamic architectures have a data-dependent inference procedure which usually requires a model to handle input samples sequentially. This poses a major challenge for the parallel processing of batched input samples.

A. Software Frameworks & Training

The architectural differences of dynamic networks require more complex training methodologies in order to achieve high accuracy on a given task. The majority of works concerning early-exit networks employ one of the following training methods. The **End-to-end** strategy jointly trains the backbone and intermediate classifiers with a loss function that is the weighted combination of losses at each exit. As noted in [22], this method has the advantage of allowing the development of a novel architecture and can lead to accuracy improvements. However, this method can be costly due to increased training times and has the potential for reduced convergence resulting from conflicting gradient information at intermediate classifiers. For larger networks, [26] uses a similar method but starts with a pre-trained backbone network. This can reduce the cost of training and improve convergence but limits the potential architectures to existing DNNs. MESS [30] and HAPI [24] use a further streamlined method of training called **IC-only training** in which, only the exit-specific computation and intermediate classifiers have their parameters updated during training and the backbone parameters are frozen. This allows the customisation of a dynamic neural network for different performance requirements based on the selection of available exits but can suffer from decreased accuracy at shallower classifiers. Training methodologies for dynamic networks are an ongoing area of research. As a result, there is limited optimised support in existing software frameworks, reducing the availability of well trained models for deployment on FPGAs.

B. Intermediate Feature Map Dependence

The dynamic nature of the network builds in an inherent reliance on control information to govern the data-flow across different network stages. From Fig 1 we note that the intermediate classifiers use the previous layer's output to reach a decision. These intermediate feature maps are required by both the intermediate classification layers or, in cases when an early exit does not occur, the remainder of the backbone of the dynamic network. Propagating the intermediate feature map through the exit branch destroys them, so to prevent unnecessary repeated computation of these feature maps, a method of preserving them is required in the form of intermediate on-chip buffering or routing to external memory. The viability of the first option is not always guaranteed for larger DNNs due to the memory requirements of intermediate feature maps in e.g



(a) (b)

Fig. 2: Explored designs for (a) Pipeline and (b) Parallel approaches to Decision Sub-Networks [29]

VGG, ResNet etc., while the second option can significantly impact the latency of a given inference.

C. Hardware Design & Toolflows

Additional layers, intermediate classifiers, and supporting components can dramatically increase the complexity of the hardware design task for Dynamic DNN deployment on FPGAs. Designers need to carefully balance the standard trade-offs between computational efficiency, latency, and resource utilization while ensuring proper synchronization and data-flow across different inference paths with increased routing congestion and resource overhead. Hand-crafted development and testing of FPGA-based accelerators for complex applications such as Dynamic DNNs presents many challenges and consumes considerable time. We note that numerous FPGA manufacturers and researchers have invested resources in developing mapping toolflows for *standard* DNNs, such as Xilinx's Vitis AI, Nvidia's NVDLA, Angel-Eye, among others. These tools are integrated with well-known static network libraries like Caffe, TensorFlow, and Torch, generating comprehensive, but often, inaccessible designs. Due to the control and data-flow nature of Dynamic DNNs, mapping the architecture requires the use of custom layer types, impacting the intermediate stages of the backbone network, resulting in the limited support of the existing frameworks in capturing and mapping the dynamic behaviour to FPGAs. We observe the need to continue to develop tooling in the area of dynamic DNN mapping to FPGAs. fpgaConvNet [9] and FINN [11] are two mapping tools that had no initial support for early-exit networks but continued development on these open-source tools has extended the support to dynamic networks, as detailed in Section IV.

IV. Advancements from Prior Works

We present an explanation of a selection of prior works that address some of the aforementioned challenges, highlighting the active development in the area and spaces for future research. We focus on the novel methods of utilizing Dynamic DNN methods, Early-Exit Networks, and their implementation on FPGA devices in the low resource setting.

Excluding the individual research efforts to develop support software for dynamic DNNs, there is little large-scale support

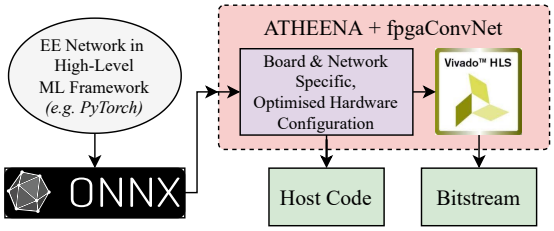


Fig. 3: High-level overview of the components of the ATHEENA tooflow [32].

in this area. Intel developed a tool for general DNN compression [31] which provides some support in this area but is limited to training and profiling the benefits of early-exits. The code does not utilization of the exits during inference but this is registered as future work.

Following the limitations of existing FPGA-based accelerators for Dynamic DNNs, [29] proposes two hardware architectures for early-exit networks designed from scratch. The pipelined approach, depicted in Figure 2 (a), adopts a traditional early-exit architecture. During a given inference, an input sample is processed normally until reaching the layer preceding the decision sub-network. The processing of the backbone is stalled until the results of the intermediate classification are computed and an early-exit decision is made. This architecture allows for the reuse of existing hardware modules, achieving nearly 0% additional area overhead. While this may result in some under-utilization of existing components, it proves beneficial in scenarios with limited resources.

However, this approach struggles with the need of saving the intermediate output, as interpreted in *SectionIII B*. To solve that a parallel design approach is proposed by (Figure 2 (b)) where the parallelisation capabilities of the FPGAs are exploited. In more detail, the early-exit branch is executed in parallel with the backbone network, utilising separate hardware modules. That way the the intermediate output is fed into both the next layer of the backbone network and the early-exit branch. This approach eradicates not only the latency overhead during early exit branch computation but also eliminates the necessity to store intermediate layer outputs.

An alternate work details the development of the toolflow, visualized in Figure 3. The toolflow investigates the resource/throughput trade-off that can be made when fac-

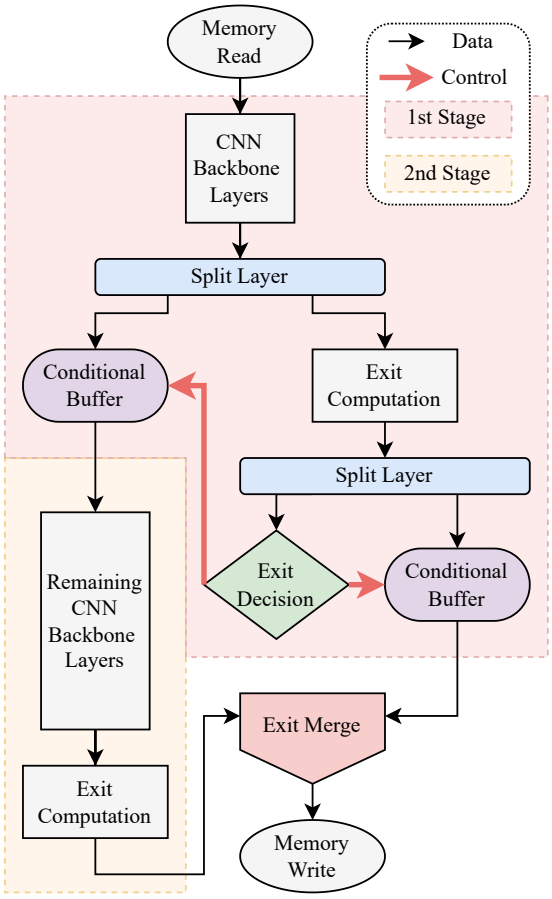


Fig. 4: A high-level visualization of the accelerator components included in the ATHEENA [32] accelerator and the separation between stages optimised to different throughput levels.

toring the probabilistic nature of the early-exit network. ATHEENA [32] is a toolflow that constructs a deeply pipelined, early-exit network accelerator to perform inference on an FPGA. The toolflow maps the full network onto the FPGA and optimises the level of parallel computation to maximise throughput under the resource constraints of a given device. This is achieved by utilising the open-source fpgaConvNet [9] as a baseline for tuning the parallelism of DNN-specific layers. ATHEENA builds on this work by profiling a given, trained, early-exit network to gauge the probability of an input sample using a given exit. Assuming the real-world task data is of a similar distribution to the training and profiling sets, then there is a high likelihood that the accelerator will have a similar utilisation of early-exits.

The early-exit network accelerator can then be split into stages, where each stage contains the backbone layers and intermediate classifiers, as shown in Figure 4. Here, a conditional buffer component is introduced to store intermediate feature maps between the network stages to reduce redundant computation. Now that we have some computations that no longer progress to later stages of the network, these layers have a lower utilisation, matching the profiled probability. This means that required throughput of these stages can be reduced by a factor of the profiled probability of exiting at that stage. This in turn means the resources of these stages can be allocated to earlier stages, improving throughput of the accelerator by $\approx 2-3\times$.

AdaPEx [33] is a related FPGA work that explore dynamic network inference on FPGAs making use of adaptive pruning and early-exit intermediate classifiers to further explore the trade-off of resources and throughput at run-time. AdaPEx is built on the FINN [11] HLS framework, used to construct multiple FPGA accelerators for a given traditional DNN. The key difference with this work is that the full system is coordinated by a runtime manager. Each of these accelerators has a different level of channel-wise pruning, with reduced accuracy corresponding to improved accelerator performance. The runtime manager is responsible for orchestrating the reconfiguration of the FPGA based on the user requirements for accuracy and throughput. Since reconfiguration can be costly, the work also provides runtime-parameterized versions of the HLS templated accelerators that can support multiple scales of pruning. This drastically reduces the latency cost of switching between models at the cost of additional resource overhead and reduced efficiency. In addition to pruning, this work augments a standard DNN with intermediate classifiers attached along the backbone of the network. Using these additional degrees of freedom, the work can explore the trade-off between accuracy, throughput and power-efficiency. One of the similarities with this and many previous works is the use of quantization to reduce the memory footprint of the parameters of DNNs deployed on the FPGA, as well as reducing the resources required for performing arithmetic operations.

V. Future Directions

Libraries. The continued development of libraries targeting dynamic networks focusing on optimization and integration with mainstream software (Pytorch, Tensorflow etc.). While training and profiling is possible with existing libraries, optimized execution that can leverage batched computation and high power compute is not well supported in mainstream tools.

Hardware-Software co-design. Existing FPGA hardware tools (hls4ml, finn etc.) have limited support in this area and even less for Dynamic DNNs. While there is an intuitive explanation for the performance improvements of Dynamic DNNs, an exploration of why they are effective in their current state will help to derive more performant architectures and adoption in different ML tasks.

Neural Models. The majority of research on Dynamic DNNs has focused on tasks of image classification utilising CNNs. However, a diverse number of models (RNNs, GANs etc.) are already efficiently deployed on FPGAs, tackling various tasks like object detection, regression and image captioning, for which dynamism could be very beneficial.

Training Strategies. Training DNNs is inherently challenging especially, when FPGAs are targeted. Developing hardware friendly strategies could enable on-device training leveraging the platform's low power and acceleration potential.

Distributed Execution. Beyond the resource constrained setting, the flexibility of both Dynamic DNNs and FPGAs can

be leveraged in multi-FPGA systems. Using runtime managers, reconfiguration, and high bandwidth interconnects, this system could tackle much larger-scale networks while continually adapting the level of computation on a per-sample basis.

VI. Conclusion

There are clear benefits of both FPGAs and dynamic DNN architectures as effective tools for reducing power consumption and flexibly adjusting computational workload to meet accuracy constraints in a resource constrained environment. This flexibility and customization is responsible for many of the key challenges preventing the widespread adoption of these types of networks, resulting in limited tools for both training and deployment. In spite of this, we highlight the existence, and continuing development, of custom implementations and tooling that address network support and hardware complexity issues, enabling users to begin to leverage some of the benefits of Dynamic DNNs on FPGAs. However, plenty of space remains for further research.

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/XX.XXXX/XXXXX/XXXXX.

References

[1] C. Shan, J. Zhang, Y. Wang, and L. Xie, "Attention-based end-to-end speech recognition on voice search," *IEEE ICASSP*, 2017.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd ICLR*, 2015.

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

[4] P. Molchanov, S. Tyree, T. Karras, T. Aila, *et al.*, "Pruning convolutional neural networks for resource efficient inference," in *ICLR*, 2017.

[5] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *ArXiv*, 2015.

[6] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, 2019.

[7] Y. Han, G. Huang, S. Song, L. Yang, *et al.*, "Dynamic neural networks: A survey," *IEEE transactions on pattern analysis and machine intelligence*, Oct. 2021.

[8] S. Veneieris, A. Kouris, and C.-S. Bouganis, "Toolflows for mapping convolutional neural networks on fpgas: A survey and future directions," *ACM Computing Surveys*, Feb. 2018.

[9] S. Veneieris and C.-S. Bouganis, "Fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE Transactions on Neural Networks and Learning Systems*, Feb. 2019.

[10] M. Hall and V. Betz, "HPIPE: heterogeneous layer-pipelined and sparse-aware CNN inference for fpgas," in *FPGA*, 2020.

[11] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," in *FPGA*, 2017.

[12] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, *et al.*, "Compressing deep neural networks on fpgas to binary and ternary precision with hls4ml," *Mach. Learn. Sci. Technol.*, no. 1, 2021.

[13] E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, *et al.*, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Comput. Surv.*, no. 2, 2019.

[14] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "Logicnets: Co-designed neural networks and circuits for extreme-throughput applications," in *FPL*, 2020.

[15] E. Wang, J. J. Davis, P. Y. K. Cheung, and G. A. Constantinides, "Lutnet: Learning fpga configurations for highly efficient neural network inference," *IEEE Transactions on Computers*, no. 12, 2020.

[16] M. Andronic and G. A. Constantinides, "Polylut: Learning piecewise polynomials for ultra-low latency fpga lut-based inference," in *International Conference on Field Programmable Technology*, 2023.

[17] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv: Learning*, 2014.

[18] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018.

[20] A. Sabet, J. Hare, B. M. Al-Hashimi, and G. V. Merrett, "Temporal early exits for efficient video object detection," *CoRR*, 2021.

[21] I. Leontiadis, S. Laskaridis, S. Venieris, and N. Lane, "It's always personal: Using early exits for efficient on-device cnn personalisation," in *22nd International Workshop on Mobile Computing Systems and Applications*, 2021.

[22] G. Huang, D. Chen, T. Li, F. Wu, *et al.*, "Multi-scale dense networks for resource efficient image classification," in *6th International Conference on Learning Representations*, 2018.

[23] L. Yang, Y. Han, X. Chen, S. Song, *et al.*, "Resolution adaptive networks for efficient inference," *CVPR*, 2020.

[24] S. Laskaridis, S. Venieris, H. Kim, and N. Lane, "Hapi hardware-aware progressive inference," *ICCAD*, 2020.

[25] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, *et al.*, "SPINN: synergistic progressive inference of neural networks over device and cloud," *CoRR*, 2020.

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

[27] X. Dai, X. Kong, and T. Guo, "Epnet: Learning to exit with flexible multi-branch network," in *29th ACM International Conference on Information and Knowledge Management*, 2020.

[28] M. Wang, J. Mo, J. Lin, Z. Wang, *et al.*, "Dynexit: A dynamic early-exit strategy for deep residual networks," in *IEEE International Workshop on Signal Processing Systems*, 2019.

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

[30] A. Kouris, S. I. Venieris, S. Laskaridis, and N. D. Lane, "Multi-exit semantic segmentation networks," in *Computer Vision - ECCV*, 2022.

[31] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, *et al.*, "Neural network distiller: A python package for DNN compression research," *CoRR*, 2019.

[32] B. Biggs, C.-S. Bouganis, and G. Constantinides, "Atheena: A toolflow for hardware early-exit network automation," in *Int. Symposium on Field-Programmable Custom Computing Machines*, 2023.

[33] G. Korol, M. G. Jordan, M. B. Rutzig, J. Castrillon, *et al.*, "Pruning and early-exit co-optimization for cnn acceleration on fpgas," in *DATE*, 2023.