RL-Agent-based Early-Exit DNN Architecture Search Framework

Mahdi Taheri[1,2], Parth Patne[1], Natalia Cherezova[2], Ali Mahani[3,4], Christian Herglotz[1], and Maksim Jenihhin[2]

[1]Brandenburg University of Technology, Cottbus, Germany
[2]Tallinn University of Technology, Tallinn, Estonia
[3]Shahid Bahonar University of Kerman, Iran
[4]York university, Toronto, CANADA

***Abstract*—This paper introduces a Reinforcement Learning (RL)-based framework for optimizing early-exit configurations in Deep Neural Networks (DNNs). By integrating RL with BranchyNet-inspired architectures, the framework dynamically determines optimal early exit placements and confidence thresholds, balancing inference time, energy consumption, and accuracy. Key contributions include an early-exit DNN architecture search, an RL-driven threshold optimization process during training, and a design-space exploration open-source framework. Experiments on models such as ResNet-18, VGG-16, and AlexNet, using benchmarks like CIFAR-10 and MNIST, reveal significant reductions in inference time (up to 69.7×) and power consumption while keeping accuracy drop within 1–2%. This work demonstrates that dynamic early-exit strategies can enhance DNN efficiency while maintaining performance, paving the way for resource-constrained applications.**

***Index Terms*—deep neural networks, dynamic DNNs, early exit**

I. Introduction

Dynamic neural networks have emerged as a transformative approach in deep learning [1], [2], addressing the challenge of computational inefficiency in static models [3]. Several works are trying to enhance the inference time of conventional deep neural networks by introducing different dynamic inference techniques. One of the most prominent strategies for achieving dynamic behavior in neural networks is using **early exits**. Early exits integrate auxiliary classifiers at intermediate layers, enabling inputs to exit the network prematurely if their classification confidence surpasses a predefined threshold. BranchyNet [4] leverages the observation that initial layers in a deep network are often sufficient for confidently classifying specific samples. Despite its advantages, BranchyNet introduces a trade-off between inference speed and accuracy. While most samples benefit from reduced execution time, some may experience longer inference times due to additional computations at each branch.

Determining optimal branch locations is crucial for maximizing BranchyNet's benefits. Current approaches rely on manually defining branch locations, which is challenging due to the vast number of possible configurations.

Although dynamic inference and early exit mechanisms, such as those employed in BranchyNet, are crucial for tackling the challenges of latency and energy consumption in modern DNNs, there is a gap in the literature regarding the optimal architecture of dynamic DNNs. This gap concerns the trade-offs between inference time, power consumption, and accuracy while also considering the need for a tailored training algorithm specifically designed for the emerging field of dynamic neural networks.

This paper presents a Reinforcement Learning-based Early exit DNN architecture search framework, to fill these gaps. The contributions of the paper are:

- Integration of a Reinforcement Learning algorithm with the training of the BranchyNet framework, enabling dynamic threshold calculation for Early Exits rather than relying on hard-coded thresholds, to achieve improved inference accuracy.
- A methodology for Network Architecture Search to determine the optimal position and number of Early Exits in a DNN architecture, considering trade-offs between inference time, power consumption, and accuracy.

The remainder of this paper is structured as follows: Section III presents the proposed methodology. Section IV discusses experimental results and Section V concludes the paper.

II. Proposed Methodology

The proposed methodology enables *early-exit* configuration search for deep neural networks (DNNs) while considering user-defined constraints on accuracy, inference time, and power consumption. The motivation is to reduce the overall inference cost by allowing certain inputs to exit the model early when partial computation satisfies the performance targets.

Our approach integrates the **BranchyNet** concept of multiple early exits inside a deep neural network architecture and optimizes the decision thresholds for each early exit via a **Reinforcement Learning** (RL) agent. This agent learns a policy to either take an early exit or continue deeper in the network, balancing overall accuracy with reduced inference cost.

A. Early-Exit Configuration Search

The proposed methodology focuses on optimizing the placement of *early exits* within convolutional neural networks (CNNs) to enable dynamic inference. The placement of these exits is determined based on the selected network **topology**, which defines how convolutional layers are grouped into *blocks*:

979-8-3315-2801-0/25$31.00 ©2025 IEEE

- **Individual Block Topology**: Each convolutional layer is treated as an independent block, allowing maximum granularity for early exit placement. For example, in **AlexNet**, 5 convolutional layers are treated as 5 separate blocks, with one potential exit per block.
- **Group Block Topology**: Consecutive convolutional layers are grouped into blocks (e.g., 2-3 layers per block), providing a balance between granularity and computational efficiency. For instance, in **VGG-16**, 13 convolutional layers are grouped into 5 blocks, with 2 or 3 layers per block.
- **Residual Block Topology**: Blocks are defined by residual connections (e.g., 2 BasicBlocks per residual block), preserving the advantages of residual learning while enabling early exits. For example, in **ResNet-18**, 17 convolutional layers are grouped into 4 residual blocks, each containing 2 BasicBlocks.

The placement of early exits is guided by the architecture of each model to balance complexity and efficiency. For ResNet-18, adding early exits after each of its 17 convolutional layers would introduce excessive complexity. Instead, we leverage its residual block design, placing exits after grouped layers with shortcut connections, preserving residual learning benefits. In VGG-16, convolutional layers are grouped into blocks (e.g., 2–3 layers per block) to strike a balance between granularity and computational efficiency. Conversely, AlexNet uses independent blocks for its simpler architecture. This block-based perspective ensures that each early-exit branch has access to sufficiently rich features for accurate classification, without incurring excessive overhead from too many exits.

The algorithm dynamically determines where to insert exits based on user-defined constraints, which ensure an optimal trade-off between efficiency and performance:

- **Accuracy drop** (δ_{acc}): Maximum acceptable decrease in accuracy compared to the static baseline model.
- **Inference time** (IF): Upper limit on the average inference time per sample.
- **Power consumption** (IP): Maximum permissible power consumption during inference.

Algorithm 1 outlines the search for valid early-exit configurations, leveraging training methods such as **Reinforcement Learning (RL)** and **BranchyNet**. The search begins with as many early exits as there are convolutional blocks in the network. If a configuration meets the constraints, it is added to a pool. Otherwise, the number of exits is reduced, prioritizing deeper blocks, until a suitable configuration is found or no exits remain.

The number of early exits is reduced based on the number of classifications made at each early exit during inference. The early exits with the highest number of classifications are retained, while those with fewer classifications during the inference of test data are omitted.

B. Early-Exit Decision Threshold Optimization

The *decision threshold* at each exit is optimized using an **RL-Agent**, which employs **Q-learning**, a model-free reinforcement learning algorithm. In this context:

Algorithm 1 Early-Exit Configuration Search

Require:
- Static model M with N convolutional **blocks**, based on the selected topology:
  - Individual Block
  - Group Block
  - Residual Block
- Baseline accuracy of static model: BaselineAcc.
- User-defined thresholds:
  - Accuracy drop: δ_{acc}
  - Inference time: IF
  - Inference power: IP

Initialize number of early exits: $E \leftarrow N$ {Potential exits based on topology (max $E = N$).}
Initialize pool of valid configurations: $Pool \leftarrow \emptyset$
while $E > 0$ **do**
  Place E early exits in model M (i.e., after the E most informative blocks).
  $(\text{acc}, \text{time}, \text{power}) \leftarrow \text{TrainAndEvaluate}(M, D, A)$
  if acc $\geq$ BaselineAcc $- \delta_{acc}$ **then**
    if time $\leq IF$ **then**
      if power $\leq IP$ **then**
        Add current configuration to $Pool$
      end if
    end if
  end if
  $E \leftarrow E - 1$ {Try fewer early exits (place them after the most critical blocks).}
end while
if $Pool \neq \emptyset$ **then**
  return $Pool$ {Pool of valid configurations.}
else
  Use the static model without early exits.
end if

- **Q-learning**: A reinforcement learning technique used to determine the optimal actions by maximizing cumulative rewards over time.
- **Q-value**: Represents the expected future reward for taking a specific action in a given state.
- **RL-Agent**: The decision-making entity that utilizes Q-learning to dynamically decide whether to exit early or continue processing deeper in the network at each layer.

The RL-Agent learns through experience to make optimal exit decisions based on the current state. The key components of reinforcement learning are:

- **State** s: Tuple of (*layer_index*, *confidence_bin*), where:
  - *layer_index*: Current exit point (0 to $E - 1$),
  - *confidence_bin*: Discretized softmax confidence (range: 0–10).
- **Action** a: Binary choice: {*exit*, *continue*}.
- **Reward Function**:

$$R(s, a) = r_{base} + r_{early}$$

where:

- r_{base}: +1.0 for correct classification, -1.0 for incorrect classification.
- r_{early}: $(E - i) \times 0.2$ bonus for exiting at layer i.

The coefficient 0.2 is chosen to provide a moderate incentive for early exits while ensuring the base reward (classification accuracy) remains the dominant factor.

The thresholds for early exits are not explicitly defined; instead, they are learned implicitly through the Q-learning process. The Q-value, which represents the total expected reward for an action in a given state, guides the agent's decisions. At each exit, the Q-learning agent compares these values for the actions *exit* and *continue*. A threshold emerges dynamically as the point where the reward for exiting surpasses the reward for continuing, reflecting the agent's confidence in the current state. This approach allows thresholds to adapt to the model architecture, dataset, and user-defined constraints.

The Q-learning agent updates its policy using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \quad (1)$$

where $Q(s,a)$ is the Q-value for taking action a in state s, $\alpha = 0.1$ is the learning rate, $\gamma = 0.9$ is the discount factor that defines the importance of future rewards [5], and $r = R(s,a)$ is the reward received for action a in state s, and $\max_{a'} Q(s',a')$ is the max Q-value for the next state s' across all possible actions a'. This iterative learning process ensures that the decision thresholds optimize both accuracy and efficiency.

III. Experimental Results

This section presents the evaluation of the proposed dynamic DNN exploration framework. The performance of static and dynamic models is compared based on accuracy, inference time, and power consumption. The configurations of the early exits are also detailed.

A. Experimental Setup

The framework was implemented in **Python** using **PyTorch**. All experiments were conducted on an **NVIDIA A100-PCIE-40GB** GPU. Power consumption was monitored using the **NVIDIA Management Library (NVML)**, ensuring precise measurements of GPU power usage.

The evaluation was performed on the following models and datasets:

- **Models**: AlexNet, ResNet-18, and VGG-16.
- **Datasets**: MNIST, CIFAR-10.

The architecture of the early-exit branches consisted of lightweight classifiers, with each branch including a global average pooling layer followed by a fully connected layer. The number and configuration of early exits depended on the convolutional layers of each model, with configurations represented as binary strings (e.g., **11111** for five exits).

In our experiments, we specified the following user-defined constraints for dynamic inference:

- **Accuracy drop** δ_{acc}: A maximum allowable drop of 1–2% relative to the baseline static model, ensuring performance remains comparable.
- **Inference time** IF: An upper bound on the *average* inference time per sample (e.g., 2–5 ms), aligning with real-time inference needs in embedded applications.
- **Power consumption** IP: A limit set to not exceed 40–50W on average for GPU power usage, monitored via NVML.

The exact values varied according to each model and dataset, reflecting practical scenarios where inference speed, accuracy, and power constraints must be jointly respected.

B. Performance evaluation

Table I presents a comparison between the *static model* (i.e., the original CNN without early exits) and various *dynamic early-exit configurations*. The static model serves as a baseline to illustrate accuracy, inference time, and power consumption when no early exits are present. Different exit configurations show how adding a certain number of early exits affects these metrics.

Strategically placed early exits can reduce power consumption by allowing many samples to exit earlier, minimizing computation. However, some configurations may slightly increase power use due to additional overhead or deeper processing for complex inputs.

Overall, Table I demonstrates that dynamic early-exit strategies can significantly reduce inference time and power consumption at a small cost in accuracy, underscoring the effectiveness of the proposed approach.

C. Comparison of Training Algorithms

Table II compares the performance of two training algorithms: Reinforcement Learning (RL) and BranchyNet.

- **Reinforcement Learning (RL):** RL is a mechanism for optimizing early-exit thresholds by learning exit decisions iteratively based on a reward mechanism. It performs efficiently on complex datasets by dynamically adapting to varying confidence distributions. Details of the RL-based methodology can be found in Section II-B.
- **BranchyNet:** BranchyNet provides a baseline for comparison, as we optimized our methodology based on its original framework. While BranchyNet introduces branch classifiers for early exits, our approach enhances this by improving exit placements and decision thresholds. The original BranchyNet methodology is cited from [4].

D. Exit Distribution Analysis

To evaluate the adaptability of our dynamic architecture, we analyzed sample exit patterns across different configurations on the CIFAR-10 dataset. Each configuration incorporates multiple early exits, with the final layer serving as the last exit. This analysis highlights the proportion of samples exiting at each branch, demonstrating the proposed methodology´s effectiveness in reducing computation while maintaining accuracy.

TABLE I: Performance Comparison of Static and Dynamic Models

Model	Dataset	Static			Dynamic				Improvements	
		Acc (%)	Inference (ms)	Power (W)	Exit Config	Acc (%)	Inference (ms)	Power (W)	Speedup	Power
AlexNet	MNIST	99.44	1.04	36.15	(11111)	99.46	0.05	34.86	20.8×	1.04×
					(11010)	99.36	0.11	38.20	9.45×	0.95×
					(11000)	99.49	0.06	34.86	17.33×	1.04×
AlexNet	CIFAR-10	79.17	0.98	36.53	(11111)	77.81	0.05	37.95	19.6×	0.96×
					(11010)	76.58	0.10	34.58	9.8×	1.06×
					(11000)	77.42	0.09	36.73	10.89×	0.99×
ResNet-18	CIFAR-10	88.63	4.18	50.02	(1111)	87.24	0.07	35.83	59.71×	1.4×
					(1110)	87.87	0.06	35.75	69.67×	1.4×
					(1010)	87.55	0.06	37.84	69.67×	1.32×
VGG-16	CIFAR-10	88.53	3.47	42.29	(11111)	87.89	0.07	35.14	49.57×	1.2×
					(11100)	88.66	0.06	37.45	57.83×	1.13×
					(11000)	88.45	0.05	39.13	69.4×	1.08×

TABLE II: Comparison of Training Algorithms: RL and BranchyNet

Model	Dataset	Algorithm	Static			Dynamic			Improvements	
			Acc (%)	Inference (ms)	Power (W)	Acc (%)	Inference (ms)	Power (W)	Speedup	Power
AlexNet	CIFAR-10	RL	89.17	0.98	36.53	87.81	0.05	37.95	19.6x	0.96x
		BranchyNet				89.93	0.28	40.25	3.5x	0.91x
ResNet-18	CIFAR-10	RL	90.17	4.18	50.02	88.24	0.07	35.83	59.7x	1.4x
		BranchyNet				90.62	0.37	54.12	11.3x	0.92x
VGG-16	CIFAR-10	RL	91.13	3.47	42.29	89.89	0.07	35.14	49.6x	1.2x
		BranchyNet				92.36	0.39	69.36	8.9x	0.61x

* Training accuracy.

Different exit pattern distributions for three configurations showed that:

- **5 early exits:** 60.6% of samples exit at the first branch, 18.2% at the second, 13.8% at the third, with progressively fewer samples at later exits.
- **3 early exits:** 61.05% exit at the first branch, 34.56% at the second, and 4.39% at the final exit.
- **2 early exits:** 60.89% exit at the first branch, 18.88% at the second, and 14.78% at the third.

These results show how early-exit strategies reduce computation by allowing most samples to exit early, maintaining accuracy while improving efficiency, especially in energy-constrained scenarios.

IV. Conclusion

This study presented a novel RL-based approach to enhance the efficiency of DNNs through dynamic early exits. The proposed methodology optimizes exit placements and thresholds, ensuring a balance between inference time, energy consumption, and accuracy. Experimental results showcased the framework's efficacy across multiple models and benchmarks, achieving inference speed-ups up to 69.7× and power reductions while maintaining accuracy drops within acceptable limits (1–2%). Comparative analyses with existing techniques highlighted the superiority of RL in adaptability and performance optimization. The contributions of this work provide a foundation for future advancements in resource-efficient DNN deployment, particularly for real-time and energy-sensitive environments.

V. Acknowledgement

This work was supported in part by the Estonian Research Council grant PUT PRG1467 "CRASHLESS", EU Grant Project 101160182 "TAICHIP" and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID "458578717".

References

[1] M. Taheri, "Dnn hardware reliability assessment and enhancement," *27th IEEE European Test Symposium (ETS).*, May 2022.

[2] M. Taheri, M. Daneshtalab, J. Raik, M. Jenihhin, S. Pappalardo, P. Jimenez, B. Deveautour, and A. Bosio, "Saffira: a framework for assessing the reliability of systolic-array-based dnn accelerators," in *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2024, pp. 19–24.

[3] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7436–7456, 2022.

[4] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," *arXiv preprint arXiv:1709.01686*, 2017.

[5] F. Pérez-Hernández, M. Alkhambashi, A. Al-Khayyat, and E. Alanazi, "Q-learning algorithms: A comprehensive classification and applications," *University of West England Xerte Publication*, 2023. [Online]. Available: https://xerte.uwe.ac.uk/USER-FILES/7297-me-perezhernandez-site/media/Q-Learning_Algorithms_A_Comprehensive_Classification_and_Applications.pdf