

Efficient Post-Training Augmentation for Adaptive Inference in Heterogeneous and Distributed IoT Environments

Max Sponner^{1,4}[0000-0002-4830-9440], Lorenzo Servadei²[0000-0003-4322-834X],
Bernd Waschneck³[0000-0003-0294-8594], Robert Wille²[0000-0002-4993-7860], and
Akash Kumar⁴[0000-0002-4830-9440]

¹ Infineon Technologies Dresden GmbH & Co. KG, Dresden 01099, Germany

² Chair for Design Automation - TU Munich, Munich 80333, Germany

³ Infineon Technologies AG, Neubiberg 85579, Germany

⁴ Chair of Processor Design, CFAED - TU Dresden, Dresden 01069, Germany

Abstract. Early Exit Neural Networks (EENNs) present a solution to enhance the efficiency of neural network deployments. However, creating EENNs is challenging and requires specialized domain knowledge, due to the large amount of additional design choices. To address this issue, we propose an automated augmentation flow that focuses on converting an existing model into an EENN. It performs all required design decisions for the deployment to heterogeneous or distributed hardware targets: Our framework constructs the EENN architecture, maps its subgraphs to the hardware targets, and configures its decision mechanism. To the best of our knowledge, it is the first framework that is able to perform all of these steps.

We evaluated our approach on a collection of Internet-of-Things and standard image classification use cases. For a speech command detection task, our solution was able to reduce the mean operations per inference by 59.67%. For an ECG classification task, it was able to terminate all samples early, reducing the mean inference energy by 74.9% and computations by 78.3%. On CIFAR-10, our solution was able to achieve up to a 58.75% reduction in computations.

The search on a ResNet-152 base model for CIFAR-10 took less than nine hours on a laptop CPU. The low search cost improves the accessibility of EENNs, with the potential to improve the efficiency of neural networks in a wide range of practical applications.

Keywords: Deep Learning · Early Exit Neural Networks · Network Architecture Search

1 Introduction

Early Exit Neural Networks (EENNs) are a possible solution to reduce the mean inference cost of Neural Networks (NNs). This approach involves inserting additional classifier branches between the network’s hidden layers that perform the

same task as the original classifier. By dynamically terminating the inference at one of these Early Exits (EEs), the computational cost and latency of the inference can be reduced. This makes them ideal for improving the energy efficiency of Deep Learning (DL) applications. However, designing and implementing an EENN for a specific scenario requires expertise in configuring the network architecture and at-runtime termination decision mechanism. An additional limitation of the design process is that an incorrect configuration can significantly increase inference costs or significantly degrade the prediction quality.

Network Architecture Search (NAS) frameworks can automate these tasks. Yet, they often rely on supernet or multi-tiered evolutionary algorithms, resulting in long search times and a large demand for compute resources. This makes such frameworks inaccessible for many developers who do not have the necessary resources.

Instead of performing NAS starting from a costly supernet, we aim to enable developers to utilize EENNs when deploying their already trained traditional NNs. Our proposed flow performs Network Augmentation (NA): it converts an existing model - designed and trained by an expert - into an EENN, maps its subgraphs to a heterogeneous or distributed device, and configures its decision mechanism. The flow performs all these required deployment steps automatically and in a efficient way to enable its usage on standard consumer-grade hardware. The goal is to make the benefits of EENNs accessible to more developers, by automating the complicated design steps while also maintaining a low cost for the conversion process.

Our framework is designed for Internet of Things (IoT) scenarios, with a focus on its accessibility for developers who do not have access to the compute capabilities of high-performance computing (HPC) clusters. To the best of our knowledge, our framework is the first to offer this range of functionality.

The paper is organized as follows: we review related work on EENNs in the Related Work section, describe the proposed NA framework in the Methodology section, present the results and analysis of the framework’s performance in the Evaluation section, and finally, summarize our findings and discuss the implications of our work in the Conclusion section.

2 Related Work

Early Exit Neural Networks (EENNs) and automated NAS for EENNs have been the subject of a growing body of research in recent years.

2.1 Early Exit Neural Networks

EENNs are a class of NNs that adapt the inference process at runtime based on the current circumstances. They are an extension of the concept of Big/Little neural networks, which combine a small and a larger model [14]. The underlying idea is to utilize the larger model only if the small model is sufficient for the current sample. EENNs extend this approach by sharing backbone layers between

these models and enabling for more than two models to be incorporated. The overall concept was introduced with BranchyNet [15] and has been visualized in Fig. 1.

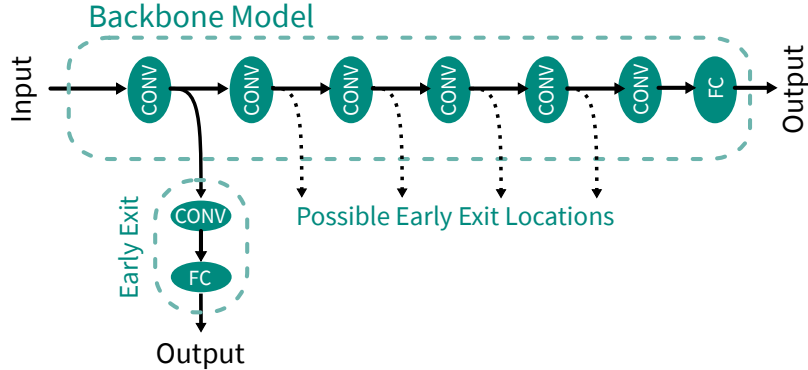


Fig. 1. An EENN architecture, which combines a backbone model with an EE. The considered but not utilized EE locations have been marked in the network graph.

An at-runtime decision mechanism performs the selection between classifiers and can be implemented in various ways. A common solution is to utilize the confidence of the already evaluated classifiers to decide on the current inference’s termination [15]. Other approaches include considering the available resource budget [7], or using an additional agent model that was trained to perform this decision [13].

EENNs provide several advantages over traditional NNs, including faster inference times, reduced energy consumption, and improved accuracy on certain tasks. However, designing effective decision mechanisms and finding optimal architectures and locations for the EEs that balance accuracy and efficiency remains a challenge.

2.2 Network Architecture Search for Early Exit Neural Networks

Designing an EENN requires the consideration of additional hyperparameters and design decisions. These include the locations, count, and architectures of the EEs; the decision mechanism and its thresholds; as well as the training strategy. To assist developers in this, NAS solutions automate these configurations. The range of covered functionalities varies between implementations, with some frameworks focused solely on the optimal branch location, which is already an NP-complete problem [4]. While others include the calibration of the output confidence score [12], mapping to heterogeneous platforms [3], or the incorporation of additional optimizations [3,2]. Table 1 provides a comparison of our novel

contribution in terms of features covered by other solutions. Our tool is - to the best of our knowledge - the first framework that also performs the configuration of exit-wise decision thresholds during the process.

Current solutions primarily utilize genetic search algorithms [3,2,5], although multi-objective bayesian search [12] and dynamic programming [4] have also been used. However, a key limitation of these solutions is the high search cost.

We want to implement our NA flow as a solution for data scientists and engineers that have already developed and trained a model architecture that suits their needs in terms of predictive performance and want to deploy it to a heterogeneous or distributed embedded/IoT environment while improving the efficiency by utilizing EENNs. This comes with different inputs, requirements and goals than the state-of-the-art NAS solutions, which is the reason why we named our approach NA as it augments an existing model with additional branches and the decision mechanism to select the most suitable classifier at runtime.

Our conversion tool appears to be the first automated flow that incorporates the ability to configure exit-wise confidence thresholds for the at-runtime decision as part of its functionality. Our publication focuses on addressing the high search cost by improving its speed and efficiency through reuse and on making EENNs more accessible by enabling developers to easily convert their existing standard models after training.

| Work | EE location | EE architecture | Decision Thresholds | MPSoC ⁵ Mapping | DVFS ⁶ | Layer Parallel |
|----------------------|-------------|-----------------|---------------------|----------------------------|-------------------|----------------|
| Optimal Location [4] | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ |
| EExNAS [12] | ✓ | ? | ✗ ⁷ | ✗ | ✗ | ✗ |
| HADAS [2] | ✓ | ✓ ⁸ | ✗ | ✗ | ✓ | ✗ |
| Map-and-Conquer [3] | ✓ | ? | ✗ | ✓ | ✓ | ✓ |
| EDANAS [5] | ✓ | ? | ✗ | ✗ | ✗ | ✗ |
| Our NA flow | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ |

Table 1. A comparison of features within the state-of-the-art solutions for EENN-NAS frameworks and our NA flow. Not all publications describe, if and how the EE branches are configured.

3 Methodology

In the previous research on NAS for EENNs, multi-tiered evolutionary search has been the primary focus. However, this approach is computationally expensive

⁵ MPSoC: Multiprocessor System-on-Chip

⁶ DVFS: Dynamic Voltage and Frequency Scaling

⁷ EExNAS calibrates the output confidence of the classifiers, but does not define a threshold value.

⁸ HADAS utilizes the same EE structure across all branches and backbone models.

and resource-intensive. To address this issue, we have developed a framework that is designed to reduce the search cost while still being able to find viable results.

The main input used by our framework is a pretrained base model. In addition, training and validation sets are provided, along with a simple hardware description for each processor. Lastly, the framework requires input on the order of processor usage, a description of the connections between processors, and the worst-case latency constraint. Optionally, a weight parameter that balances efficiency gains and accuracy reduction penalties can be set. During the search, the efficiency gain is defined as the share of the reduction in mean operations per inference compared to the original model.

The architecture search space is constructed from mutations of the base model. It is pruned by evaluating only those that are predicted to fit within the given latency constraint or the memory sizes of the targeted devices.

We use worst-case latency as a constraint because it is critical for many embedded applications, especially in control loops, where delays in processing can have severe consequences. Moreover, by considering worst-case latency, we ensure that the network can meet the latency requirements in all cases, not just on average.

We assume the EEs of an EENN to be independent. This assumption is based on the similarity of EENNs to IDK classifier cascades [1]. These classifier cascades consist of multiple different models that are always queried in the same order and only proceed with the next classifier stage, if the current stage returns IDK (“I don’t know”), otherwise the inference is terminated. For EENNs the decision mechanism is equivalent to the IDK label of these cascades. While the EE classifiers of an EENN share weights through the feature extraction layers of the backbone, they do not operate on the same representation as they are attached at different locations of the backbone and the classifier that operates on the extracted features is distinct from the backbone model as it is its own branch without shared weights. This enables the assumption that the predictions produced by the classifiers of an EENN are not correlated.

This assumption allows us to individually evaluate the EEs and treat architectures within the search space as cascades of its contained classifiers. This enables the flow to reuse the costly training steps across architectures but might limit their prediction performance without the joint-training step that is only applied to the found solution.

It also allows the NA to efficiently search the optimal exit-wise threshold configuration for each architecture. This is done by converting the confidence threshold search space into a directed graph. The nodes represent the different threshold configurations of each exit. The edges between the nodes are annotated with the impact on efficiency and accuracy when a specific threshold for an exit is combined with the thresholds of the other exits. This turns the search for the optimal decision mechanism into a shortest-path problem, which can be solved rather quickly.

When searching for the best solution on the evaluated options, each architecture is only considered with its best decision configuration. This drastically reduces the number of considered options during the final search step. Once a solution has been found, an optional joint training step can be applied to the EENN, which trains the model for one epoch to finetune the EEs and backbone. If this optional step is applied, another search for the threshold configuration is performed afterward.

3.1 Architecture Search

The architecture search step is the process of finding the best architectures and locations for EEs that can be attached to the base model. The first step of this process is translating the input model into two graph-level representations. The fine-grained representation operates on a layer-level. It is used to estimate the inference cost and derive a blueprint from the classifier of the base model. The coarse-grained block-level representation collapses residual blocks and fuses post-processing layers with compute layers into nodes called blocks. The block-level representation is used to identify possible locations for EEs. The creation of these fused blocks reduces the number of locations that need to be evaluated without impacting the quality of the found architectures. The nodes of both graphs are associated with their estimated cost in terms of Multiply-Accumulate (MAC) operations, inference latency, memory, and storage footprint. The current version of the framework uses simple approximations instead of accurate performance models to estimate the inference cost, as we focused on the search algorithm.

After identifying the possible locations for EEs, the EE branches are configured for each location. The architecture of each EE is based on the classifier blueprint that was extracted from the backbone model. Depending on the size of the intermediate feature map (IFM) at the location, additional downsampling layers might be inserted before the EE classifier. This process is rule-based and ensures that the added resource consumption of the EE branches stays well below the overall inference cost of the model. As the main focus is IoT scenarios, the downsampling is applied aggressively to minimize the additional cost of the EE branches. To the best of our knowledge, our framework is the first to construct the EEs based on the original classifier, which ensures that they are able to perform the same task as it. It also enables the construction of EEs that are optimized for the required task and our framework can be extended to support tasks outside of the currently supported (binary) classification if a confidence metric can be defined for their results.

We limited the maximum number of classifiers that can be inserted to the number of target processors. This limitation is due to the lack of support for conditional execution in most DL frameworks and dedicated accelerators, especially in the embedded field. Our framework adheres to these limitations to ensure compatibility with standard toolchains and simplifies the targeting of IoT scenarios. The alignment of EEs with the boundaries between processors is also intended to maximize the efficiency on platforms with separate power domains. This stems from the observation, that deeper EEs tend to achieve higher accuracy

scores and is intended to reduce the utilization of the following processors by maximizing the prediction quality and termination rate of the classifier assigned to the current processor.

Side effects of these steps are the shrinking of the search space, a simplification of the deployed control flow, and a limitation of the overhead that can be introduced into the inference process. The mapping of the possible solutions has been visualized in Figure 2, which illustrates the distribution of classifiers across the targeted processors.

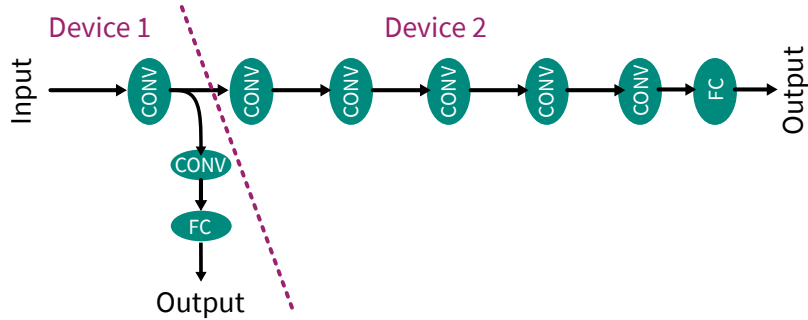


Fig. 2. The EENN architecture from Figure 1 has been mapped to a platform that contains two processing targets.

The search process starts by generating a set of all EENN versions of the base model that are estimated to lie within the given constraints and adhere to the maximum number of classifiers. The assumed independence of EEs enables each EENN to be evaluated as the weighted sum of the performance of the contained classifiers. The used weights are the estimated termination rates of the classifiers. Each evaluated EEs is trained and evaluated individually on the frozen base model. Freezing the weights of the shared layers has the benefit of drastically reducing the training costs, as the EEs are very small compared to the backbone, enabling their training on consumer hardware. Additionally, it prevents interactions between the classifiers during the training, maintaining their independence during this step, which is necessary to enable their reuse across evaluated options. While related work like HADAS [2] also trains the EEs on frozen backbone weights during its inner optimization loop, it seemingly does not reuse their evaluation results across the different EENN architectures within the search space and does not perform any joint training of the found solution. These optimizations can greatly reduce the computational resources required and accelerate the search process.

Once all EEs of a possible solution have been evaluated, their threshold configuration is searched. This is done for each evaluated EENN architecture, as the final performance largely depends on the decision mechanism. The relevant metrics of each possible architecture are predicted based on the performance of

the contained classifiers and their termination thresholds. These metrics include accuracy, early termination rates, mean inference time, and cost (in terms of MAC operations). By combining these metrics, we can associate each EENN option with a scalar cost value that is the sum of the individual costs.

In the final step of the process, the evaluated EENNs are sorted by their cost, and the element with the lowest value is selected as the returned solution. An optional fine-tuning step is intended to jointly optimize the weights of the shared feature extraction layers, as training the individual EEs on a frozen backbone can limit their achievable performance. However, this step is optional to enable systems that do not have the compute capabilities to train the backbone model to still perform the NA.

3.2 Decision Mechanism Configuration

The configuration of the decision mechanism for an EENN is important for the final performance of the model, as it significantly impacts accuracy, latency, and energy efficiency.

The search is performed by evaluating the relevant metrics on the validation set for a discretized range of possible thresholds on each EE.

The validation set or a similar set of unseen samples is used to get confidence values of the EEs for unseen samples. If no such set is available, our framework will fall back to the training set. This fallback will result in overly optimistic decision configurations due to the higher confidence on samples that the model was trained on. This is compensated by scaling the found thresholds with a correction factor.

A search graph is constructed based on the evaluated values, where each node represents a tuple of the classifier and its associated threshold. Such a search graph has been visualized in Fig. 3. The edges represent the change in relevant metrics between these nodes and are associated with a scalar cost value during the search. We use the Bellman-Ford shortest path algorithm to search for the optimal decision mechanism, as edges do not necessarily have to be associated with a positive value. However, the size of the search graph is small enough that the difference in cost compared to Dijkstra is negligible.

The search graph is limited in size as thirteen nodes are inserted per early classifier that is part of the EENN architecture. For an EENN with two EEs and one final classifier, the resulting search graph for an architecture option has 28 nodes: two EEs, one input node, and one node for the final classifier that is associated with a threshold of zero, as all remaining samples will terminate there.

The optional second search step for the selected EENN allows for the consideration of significantly more thresholds.

4 Evaluation

The evaluation focuses on assessing the performance of EENNs in typical embedded DL scenarios. To this end, we convert and deploy a speech command

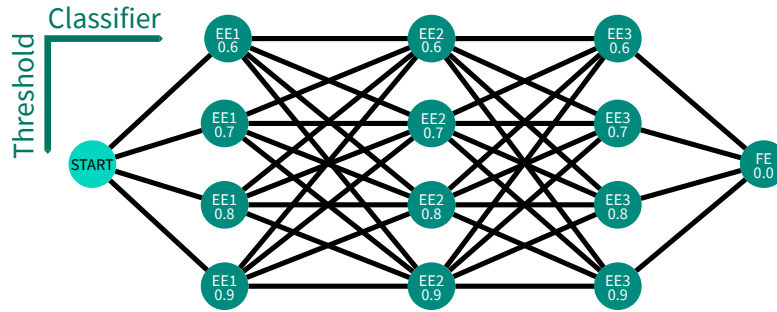


Fig. 3. An illustration of the graph structure that is used for the search of exit-wise confidence thresholds for an EENN with three EEs. This illustration has limited the range of evaluated thresholds for readability purposes.

recognition and Electrocardiography (ECG) classification model to a heterogeneous microcontroller (MCU).

The Infineon PSoC 6[®] MCU was selected, as it contains an ARM Cortex-M0Plus[®] (M0) and an additional Cortex-M4F[®] (M4F) processor that share up to 1MB of SRAM memory and up to 2MB of Flash storage. The M0 core operates at 100 MHz, while the M4F operates at 150 MHz when active. Despite its multi-core architecture, the memory is single-ported, which prevents the cores from running concurrently. Only one core can be utilized at a time, leaving the other processor in a sleep state.

The energy values reported are estimates based on the measured runtimes and the reported power consumption for the associated states of the MCU according to the technical documentation of the platform⁹.

We propose mapping EENNs to the MCU by using the M0 core as an always-on device that performs permanent monitoring within a small energy envelope. In cases where uncertainty arises, the M4F core will be used to provide additional prediction capabilities. This allows for high accuracy in permanent monitoring while maintaining high energy efficiency. This approach is expected to result in extended battery life, making it suitable for scenarios where this is a critical factor.

All searches were performed on an Intel Core i5-1240P mobile CPU with 64 GiB of system memory, TensorFlow 2.13 for all training and prediction steps, and without GPU acceleration. An overview of the experimental results can be found in Tab. 2.

⁹ PSoC6 CY8C624ABZI-D44 data sheet: https://www.infineon.com/dgdl/Infineon-PSOC_6_MCU_CY8C62X8_CY8C62XA-DataSheet-v16_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0ee7d03a70b1

| Model | ARM-DS | 1D-CNN | ResNet-152 | | | | | | | |
|--------------|-------------------------------------|---------------------------|--|----------------------------|---------------------------|---------------------------|----------------------------|----------------------------|---------------------------|--------------------------|
| | (L) | | CIFAR-10 | | | | CIFAR-100 | | | |
| Calibration | val. | val. | 1 | 2/3 | 1/2 | val. | 1 | 2/3 | 1/2 | val. |
| Dataset | GSC | MIT-BIH | CIFAR-10 | | | | CIFAR-100 | | | |
| Target | PSoC6: Cortex-M0P, Cortex-M4F | | Cortex-A55+A76, Mali G610, LTE Uplink (50Mbps), RTX 3090 Ti | | | | | | | |
| Training | 89 min | 17 min | pretrained | | | 1,119 min | pretrained | | | 1,076 min |
| Search | 18 min | 7 min | 504 min | | | 270 min | 561 min | | | 266 min |
| Acc. | 84.45% -12.96 | 96.5% -3.1 | 92.81% -1.18 | 86% -7.99 | 72.72% -21.25 | 92.66% -0.32 | 72.56% +0.02 | 72.49% -0.05 | 71.85% -0.69 | 70.48% +0.65 |
| Prec. | 91.6% -7.77 | 99.4% -0.2 | 92.84% -1.25 | 86.2% -8.09 | 73% -21.08 | 92.68% -0.31 | 73.04% -2.31 | 72.92% -2.43 | 72.29% -3.06 | 70.72% +0.52 |
| Recall | 84.4% -14.93 | 98.6% -1.0 | 92.81% -1.11 | 86% -7.89 | 72.74% 21.15 | 92.66% -0.32 | 72.56% +1.11 | 72.49% +1.04 | 71.85% +0.4 | 70.48% +0.65 |
| Mean MACs | 11.8M -59.67% | 0.33M -78.3% | 318.12M -11.3% | 226.04M -36.99% | 147.99M -58.75% | 330.97M -7.75% | 357.24M -0.43% | 349.4M -2.61% | 342.73M -4.47% | 358.28M -0.13% |
| Mean Latency | 1.06 sec +34.37% | 0.62 sec -58.9% | 16.2 ms -9.18% | 12.35 ms -30.79% | 8.6 ms -51.79% | 16.28 ms -8.77% | 21.05 ms +19.72% | 19.57 ms +11.33% | 18.94 ms +7.76% | 17.19 ms -2.2% |
| Mean Energy | 21.3 mJ -13.6% | 11.83 mJ -74.9% | - | - | - | - | - | - | - | - |
| Early Term. | 83.4% | 100% | 36.99% | 86.97% | 95.4% | 31.16% | 13.69% | 61.65% | 74.39% | 0.33% |

Table 2. The performance of the created EENNs compared to placing the entire original network on a single processor of the target platform (either the M4F core or the Mali GPU). The difference in relevant performance metrics to the submitted model are marked in bold.

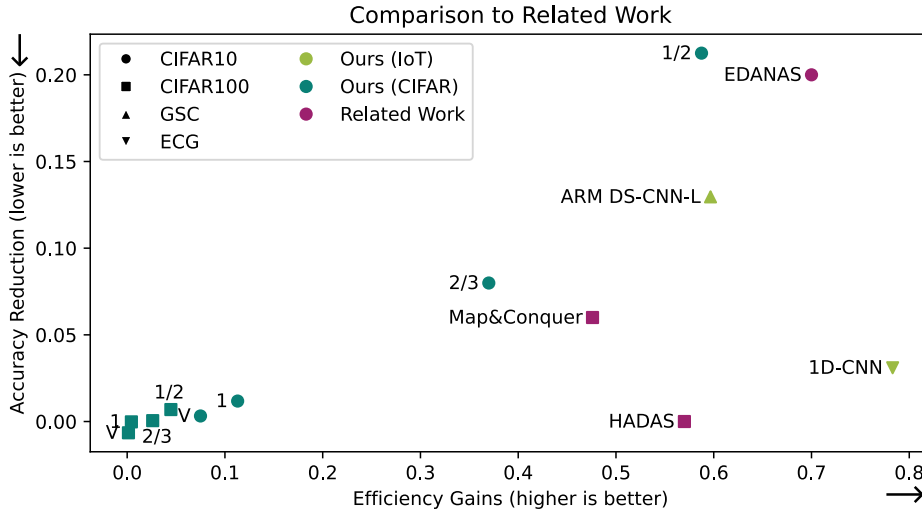


Fig. 4. Performance comparison of our framework to previous work using different base models on the same datasets.

4.1 Speech Command Detection

The first experiment was the detection of speech commands on audio data. The Google Speech Command dataset [16] was used in combination with a depthwise-separable CNN from ARM in its largest version [17]. The model performs a classification into eleven different classes: nine commands, silence, and background noise, mostly consisting of spoken words that are not part of the set of commands. Possible applications for this are either wake word detection or accessibility features.

The reference model achieves 99.33% accuracy, 99.37% precision, and 99.33% recall. The search was configured with a desired worst-case latency of 2.5 seconds. The required hardware description specifies the estimated processing speed of both cores in MAC operations per second, their interconnect speed, and the available memory size. The interconnect speed was defined as the theoretical speed of the used memory. The processing speeds of the cores were estimated at 10 million MAC operations per second for the M0 and 75 million for the M4F. The search was parameterized with a weight of 0.9 on reduction of the inference cost and 0.1 on maintaining accuracy.

The search space consists of six possible architectures, one of which was rejected before its accuracy evaluation due to being estimated to be outside of the latency constraint.

The selected solution adds an EE after the second convolutional layer of the backbone and configures its confidence threshold to 0.6. The EE on its own achieved 74.99% accuracy, 86.34% precision, and 74.93% recall. The inference of the entire EENN takes 967.99 ms for the M0 subgraph and an additional 521 ms for the second subgraph that has been mapped to the M4F. This results in a worst-case latency of 1.5 seconds, which lies well within the set constraint.

The execution of the M0 workload requires 18.53 mJ of energy, and the M4F subgraph requires an additional 16.65 mJ if it needs to be executed.

Despite the significant decrease in mean MAC operations per inference, the M0's capabilities limit the achievable efficiency gains. In contrast to the M4F, there is no dedicated instruction or unit to efficiently perform MAC operations on the M0 core. As a result, the performance improvements are modest in comparison to the reduction in MAC operations. Due to the promising reduction in MAC operations, we remain optimistic about the use with future generations of MCUs that feature more advanced cores within the always-on power domain.

4.2 ECG Classification on Wearable Devices

The second experiment targets the classification of single-lead ECG signals using a convolutional NN. The MIT-BIH dataset [11] was used and a fully convolutional NN was utilized as the backbone model [8]. The data was split into dedicated train, validation, and test sets. The model classifies the ECG signals into six classes: normal heartbeat, atrial premature beat, premature ventricular contractions, right-bundle or left-bundle branch block beat, and paced beat. Premature beats and block beats are indicators of underlying health conditions

that medical experts should investigate. The model achieved 99.29% accuracy, 99.33% precision, and 99.25% recall on the test set. A possible application is the deployment to wearable devices like smartwatches to enable constant monitoring.

The previous experiment’s hardware description and search configuration were adapted to account for additional reshaping operations inserted by the embedded DL toolchain (TensorFlow Lite for microcontrollers).

The resulting EENN inserted the EE after the first convolutional block and configured its threshold to 0.6. The execution of the first subgraph takes 618 ms on the M0, and the second subgraph 1.376 seconds on the M4F.

The high termination rate indicates that the model might be overparameterized. However, the option to execute the entire EENN is beneficial for healthcare applications as it ensures a high accuracy of the classification results if needed.

4.3 Image Classification

To ensure that our results can be compared to established work, we conducted experiments using the CIFAR-10 and -100 datasets [9] and converted ResNet-152 [6] accordingly. We used a distributed system consisting of a Rockchip RK3588[®]-based platform, which includes four Cortex-A76 and four Cortex-A55 cores, along with a Mali G610 GPU and 16 GiB of shared system memory. The CPU cores are grouped into a single target for our framework so as not to interfere with the operating system’s scheduler and optimized kernels for this platform. Additionally, the local device has a 50 Mbps connection to a workstation with an Nvidia GeForce RTX 3090 Ti to simulate a smartphone connected to a cloud backend.

We opted to use ResNet-152 as a particularly challenging test case. This model’s depth creates 74 potential EE locations, resulting in a total of 2,776 possible EENN architectures for our target system. Each of these architectures has up to 169 threshold configuration options, resulting in a search space of approximately 450,000 configurations. We found that reducing the latency constraint did not significantly decrease the overall search time. Even with fewer architectures to evaluate, the majority of the search time is spent on training and evaluating all potential EEs, which are still contained in the remaining architectures.

We performed two evaluations of the conversion process: one with a dedicated calibration/validation set (20% of the training set) and another without it. In the latter, we used pretrained models [10] and calibrated the thresholds using the training data. A correction factor is applied after calibration to account for the increased confidence compared to unseen samples. Different correction factors have been evaluated.

Our results indicate that using a correction factor and only the training set can be a viable alternative to a calibration set, particularly when the latter is unavailable. The solutions that used a dedicated calibration set achieved the lowest reduction in prediction quality and the fastest search times. The reduction in search time was achieved as the framework uses the calibration set to terminate the evaluation of EEs early. This early termination uses the calibration

set performance after the first training epoch of the EE to decide if the classifier is able to achieve a meaningful prediction quality. On the other hand, the EENNs models that were created using only a training set and a correction factor achieved significantly higher efficiency gains and local termination rates but showed a larger decrease in prediction quality. This result is caused by generating threshold configurations that are lower than the lower boundary of the search space by applying the correction factor. This showcases that some of the design decisions within our framework that are intended to target low-energy IoT applications with smaller network sizes and a limited number of classes interfere with the conversion of larger computer vision models like ResNet-152 and the 100 classes of CIFAR-100.

Despite these limitations our framework was able to discover solutions for this larger problem sizes using consumer-grade hardware in a short time frame. Additionally, all found solutions achieved (limited) efficiency gains. While the created EENNs did not always show the same level of improvement as established work that also utilized additional optimizations (see Tab. 1), as depicted in Fig. 4, they still achieved usable results.

The design decisions that limit the performance include the rule-based system to adapt the EE classifiers to the backbone, which aims to minimize the overhead. This system was designed for IoT applications with shallower models and smaller IFMs and resulted in EENNs whose branches total up to a footprint of less than 0.5% of the backbone’s MAC operations. This design choice reduces the predictive capabilities of EE classifiers in the first half of EE locations, which limits their potential contribution. This limited the performance of the solutions within the search space. More research is needed to extend the rule-based system to larger IFM and network sizes. Additionally, the threshold search space was designed for the lower class count of most embedded use cases, which limited the quality of its results on the CIFAR-100 dataset.

As shown in Table 2, our approach requires up to 9.4 hours of search time for this problem on the used mobile CPU to analyze the search space of 2,776 possible EENN architectures and their optimal decision mechanism configuration. Related work relied on clusters of twelve to 32 unnamed GPUs, accumulating between twelve hours and three days of search time to find an EENN without its optimal decision mechanism [3,2]. The CPU used for the evaluation of our NA flow has a peak power consumption of less than 70 W, which is significantly lower than the sustained power consumption of data center GPUs used by previous publications. A direct comparison between our NA and the previous work is not meaningful due to the difference in functionality and features. However, the exhaustive training and evaluation of the 2,776 EENN architectures would require about 86.75 days of compute time on the used CPU. This duration was estimated based on the duration of the single fine-tuning epoch applied to the found solution (540 seconds for one epoch), the assumption that five epochs would be necessary for each architecture in the search space and 2,776 contained architectures. This time does not include the steps necessary to configure the

decision mechanism for each option or the selection of the solution from the search space.

5 Conclusion

Our framework has demonstrated the ability to augment and map submitted models for the use as EENNs on distributed or heterogeneous devices. By modifying submitted standard models, it allows developers to quickly deploy their models to heterogeneous and distributed IoT environments without the need for dedicated supernets or expert knowledge on adaptive inference. It can often find EENN versions of the backbone model in a shorter time frame than the training of the backbone itself requires and also fully configures the required decision mechanism within this time window.

We have applied the framework to a range of applications and target devices and have shown that it can identify viable solutions that reduce the mean inference cost, even for workloads it was not designed for. To the best of our knowledge, our NA framework is the first to cover the conversion into an EENN architecture, its mapping to a heterogeneous platform and the configuration of the optimal confidence-based decision mechanism in a fully automatic process.

Moving forward, there are several areas that could be explored to improve our framework. One possibility is to enhance the search algorithm to further increase the quality of the found solutions. Additionally, the framework could be adapted to better support the conversion of larger NNs beyond the IoT domain. Lastly, the training and evaluation of the EE classifiers is a bottleneck in the search process. Future work could explore the option of predicting the possible performance of an EE based on the backbone model, the attachment location, and the EE architecture. This approach could significantly reduce search time and be used to guide the search towards options that are predicted to be of higher quality.

Acknowledgement

The project “RadarSkin” has received funding from the German Federal Ministry of Education and Research (BMBF) under the call “Electronic Systems for Edge Computing” (grant number 16ME0543). The responsibility for the content of this publication lies with the author.

References

1. et al., X.W.: IDK cascades: Fast deep learning by learning not to overthink. In: UAI 2018 (2018), <http://auai.org/uai2018/proceedings/papers/212.pdf>
2. Bouzidi, H., Odema, M., Ouarnoughi, H., Faruque, M.A.A., Niar, S.: HADAS: hardware-aware dynamic neural architecture search for edge performance scaling. In: Design, Automation & Test in Europe Conference & Exhibition, DATE 2023. pp. 1–6. IEEE (2023). <https://doi.org/10.23919/DATE56975.2023.10137095>, <https://doi.org/10.23919/DATE56975.2023.10137095>

3. Bouzidi, H., Odema, M., Ouarnoughi, H., Niar, S., Faruque, M.A.A.: Map-and-conquer: Energy-efficient mapping of dynamic neural nets onto heterogeneous mpsoes. In: 60th ACM/IEEE Design Automation Conference, DAC 2023, San Francisco, CA, USA, July 9-13, 2023. pp. 1-6. IEEE (2023). <https://doi.org/10.1109/DAC56929.2023.10247722>, <https://doi.org/10.1109/DAC56929.2023.10247722>
4. Chiang, C., Liu, P., Wang, D., Hong, D., Wu, J.: Optimal branch location for cost-effective inference on branchynet. In: 2021 IEEE International Conference on Big Data (Big Data). pp. 5071-5080. IEEE (2021). <https://doi.org/10.1109/BIGDATA52589.2021.9671948>, <https://doi.org/10.1109/BIGDATA52589.2021.9671948>
5. Gambella, M., Roveri, M.: EDANAS: adaptive neural architecture search for early exit neural networks. In: International Joint Conference on Neural Networks, IJCNN 2023, Gold Coast, Australia, June 18-23, 2023. pp. 1-8. IEEE (2023). <https://doi.org/10.1109/IJCNN54540.2023.10191876>, <https://doi.org/10.1109/IJCNN54540.2023.10191876>
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (Jun 2016). <https://doi.org/10.1109/cvpr.2016.90>, <http://dx.doi.org/10.1109/cvpr.2016.90>
7. Hu, H., Dey, D., Hebert, M., Bagnell, J.A.: Learning Anytime Predictions in Neural Networks via Adaptive Loss Balancing (May 2018). <https://doi.org/10.48550/arXiv.1708.06832>
8. Issa, M.F., Yousry, A., Tuboly, G., Juhasz, Z., AbuEl-Atta, A.H., Selim, M.M.: Heartbeat classification based on single lead-II ECG using deep learning. *Heliyon* **9**(7), e17974 (2023). <https://doi.org/10.1016/j.heliyon.2023.e17974>
9. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep. (2009)
10. Lee, W.: Pretrained models for resnet cifar. GitHub repository (2018), https://github.com/BIGBALLON/pretrained_models
11. Moody, G.B., Mark, R.G.: MIT-BIH Arrhythmia Database (1992). <https://doi.org/10.13026/C2F305>
12. Odema, M., Rashid, N., Faruque, M.A.A.: EExNAS: Early-Exit Neural Architecture Search Solutions for Low-Power Wearable Devices. In: IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED 2021. pp. 1-6. IEEE (2021). <https://doi.org/10.1109/ISLPED52811.2021.9502503>
13. Odena, A., Lawson, D., Olah, C.: Changing Model Behavior at Test-Time Using Reinforcement Learning (Feb 2017). <https://doi.org/10.48550/arXiv.1702.07780>
14. Park, E., Kim, D., Kim, S., Kim, Y., Kim, G., Yoon, S., Yoo, S.: Big/little deep neural network for ultra low power inference. In: 2015 International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS 2015. pp. 124-132. IEEE (2015). <https://doi.org/10.1109/CODESISSS.2015.7331375>, <https://doi.org/10.1109/CODESISSS.2015.7331375>
15. Teerapittayanon, S., McDanel, B., Kung, H.T.: Branchynet: Fast inference via early exiting from deep neural networks. In: 23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016. pp. 2464-2469. IEEE (2016). <https://doi.org/10.1109/ICPR.2016.7900006>, <https://doi.org/10.1109/ICPR.2016.7900006>
16. Warden, P.: Speech commands: A dataset for limited-vocabulary speech recognition. arXiv [abs/1804.03209](https://arxiv.org/abs/1804.03209) (2018), <http://arxiv.org/abs/1804.03209>

17. Zhang, Y., Suda, N., Lai, L., Chandra, V.: Hello edge: Keyword spotting on microcontrollers. arXiv [abs/1711.07128](https://arxiv.org/abs/1711.07128) (2017), <http://arxiv.org/abs/1711.07128>