Adaptive Inference through Early-Exit Networks: Design, Challenges and Directions

Stefanos Laskaridis†, Alexandros Kouris‡, Nicholas D. Lane†‡
†Samsung AI Center, Cambridge ‡University of Cambridge {stefanos.l,a.kouris,nic.lane}@samsung.com

ABSTRACT

DNNs are becoming less and less over-parametrised due to recent advances in efficient model design, through careful hand-crafted or NAS-based methods. Relying on the fact that not all inputs require the same amount of computation to yield a confident prediction, adaptive inference is gaining attention as a prominent approach for pushing the limits of efficient deployment. Particularly, early-exit networks comprise an emerging direction for tailoring the computation depth of each input sample at runtime, offering complementary performance gains to other efficiency optimisations. In this paper, we decompose the design methodology of early-exit networks to its key components and survey the recent advances in each one of them. We also position early-exiting against other efficient inference solutions and provide our insights on the current challenges and most promising future directions for research in the field.

CCS CONCEPTS

• Computing methodologies → Neural networks; • Human-centered computing → Ubiquitous and mobile computing systems and tools.

KEYWORDS

Early-exit networks, Dynamic inference, Adaptive computing, Deep learning, Mobile systems

1 INTRODUCTION

During the past years, there has been an unprecedented surge in the adoption of Deep Learning in various tasks, ranging from computer vision [22] to Natural Language Processing (NLP) [69] and from activity recognition [72] to health monitoring [62]. A common denominator and, undoubtedly, a key enabler for this trend has been the significant advances in hardware design [1, 31] (e.g. GPU, ASIC/FPGA accelerators, SoCs) along with the abundance of available data, both enabling the training of deeper and larger models. While the boundaries of accuracy are pushed year by year, DNNs often come with significant workload and memory requirements, which make their deployment on smaller devices cumbersome, be it smartphones or other mobile and embedded devices found in the wild. Equally important is the fact that the landscape of deployed devices is innately heterogeneous [78], both in terms of capabilities (computational and memory) and budget (energy or thermal).

To this direction, there has been substantial research focusing on minimising the computational and memory requirements of such networks for efficient inference. Such techniques include architectural, functional or representational optimisations in DNNs [71], aiming at faster forward propagation at a minimal cost. These include custom – hand or NAS-tuned – blocks [26, 66], model weights sparsification and pruning [19] as well as low-precision representation and arithmetics [46, 59]. Given there is no free lunch in Deep Learning, most of the aforementioned approaches trade off model accuracy for benefits in latency and memory consumption. Moreover, while some of these approaches may work out-of-the-box, others do require significant effort during or post training to maintain performance or to target different devices [19].

A complimentary family of solutions further pushing the efficiency envelope exploits the accuracy-latency trade-off at runtime, by adapting the inference graph [11, 25, 75, 79] or selecting the appropriate model [20, 43] for the device, input sample or deadline at hand. This category includes early-exiting (EE) [68]. Early-exit networks leverage the fact that not all input samples are equally difficult to process, and thus invest a variable amount of computation based on the input’s difficulty and the DNN’s prediction confidence; an approach resonating with the natural thinking mechanism of humans. Specifically, early-exit networks consist of a backbone architecture, which has additional exit heads (or classifiers) along its depth (Fig. 1). At inference time, when a sample propagates through the network, it flows through the backbone and each of the exits sequentially, and the result that satisfies a predetermined criterion (exit policy) is returned as the prediction output, circumventing the rest of the model. As a matter of fact, the exit policy can also reflect the target device capabilities and load and dynamically adapt the network to meet specific runtime requirements [41, 42].

Reaping the benefits of early exiting upon deployment, however, is not as trivial as jointly training a backbone network with randomly placed exits. One needs to carefully design the network and the training sequence of the exits relative to the backbone before choosing the exit policy for the deployment at hand. These decisions can be posed as a Design Space Exploration problem that can be efficiently traversed through a “train-once, deploy-everywhere” paradigm. This way, the training and deployment processes of early-exit networks can be detached from one another [41].

This paper provides a thorough and up-to-date overview of the area of early-exit networks. Specifically, we first describe the typical architecture and major components of these networks across modalities. Next, we survey the state-of-the-art techniques and bring forward the traits that make such models a compelling solution. Last, we conclude by discussing current challenges in existing systems, the most promising avenues for future research and the impact of such approaches on the next generation of smart devices.

2 EARLY-EXIT NETWORKS

DNNs can be thought as complex feature extractors, which repre-
sent inputs into an embedded space and classify samples based on the separability of classes in the hyperplane manifold. Typically, shallow layers extract low-level features, such as edges, whereas deeper ones build upon higher level semantics. Under that framework, early exits can be thought as early decisions based on the shallower representations. The hypothesis behind their operation lays on the fact that such features on easier samples might be enough to offer the desired distinguishability between classes.

Several important decisions arise when designing, training and deploying such networks, however, as different designs affect the dynamics the network precision, performance and efficiency. In this and the following sections we go through the workflow of deploying an early-exit network (Fig. 2).

2.1 Designing the architecture

Model & Exit Architecture. Initially, one needs to pick the architecture of the early-exit model. There are largely two avenues followed in the literature: i) hand-tuned end-to-end designed networks for early-exiting, such as MSDNet [30], and ii) vanilla backbone networks, enhanced with early exits along their depth [12, 35, 41, 68]. This design choice is crucial as it later affects the capacity and the learning process of the network, with different architectures offering varying scalability potential and convergence dynamics.

In the first case, networks are designed with progressive inference carved into their design. This means that the model and the architecture of its early exits are co-designed – and potentially trained jointly. Such an approach allows for more degrees of freedom, but potentially restricts the design’s performance across different circumstances and deployment scenarios, since this decision needs to be made early in the design process. For example, the existence of residual connections spanning across early exits can help generalisability of the network. On the other hand, some properties, such as maintaining multiple feature size representations, can prove detrimental in terms of model footprint [30].

On the other hand, when disentangling the backbone network’s design from the early exits, one can have the flexibility of lazily selecting the architecture of the latter ones. Although this might not yield the best attainable accuracy, since the two components are not co-designed, it enables case-driven designs of early-exits that can be potentially trained separately to the main network and selected at deployment time [41].

It is worth noting that early exits can adopt a uniform or non-uniform architecture, based on their placement. While the latter enlarges the design space of early-exit networks, it creates an interesting trade-off: The number (and type\footnote{Type can refer to the type of convolutions, such as regular vs. depthwise separable.}) of exit-specific layers accuracy vs. their overhead. While the adaptive and input-specific nature of early-exit networks is highly praised, when an early output does not meet the criteria for early-stopping, the runtime of the exit-specific layers are essentially an overhead to the inference computation. As such, the early exits need to be designed in comparison with the backbone network (i.e. relative cost) and with the exit policy at hand (i.e. frequency of paying that cost).

Number & Position of Early-exits. In parallel with the architecture, one also needs to select the number and positioning of early exits along the depth of the network. This decision not only affects the granularity of early results, but also the overall overhead of early-exiting compared to the vanilla single-exit inference. Too densely placed early exits can yield an extreme overhead without justifying the gains achieved by the extra classifiers, whereas too sparse placements can offer large refinement period until the next output is available. Moreover, having too many early classifiers can negatively impact convergence when training end-to-end. With respect to positioning a given number of early exits, they can be placed equidistantly or at variable distances across the depth of the network. The decision depends on the use-case, the exit rate and the accuracy of each early exit. It is worth noting that this inter-exit distance is not actual “depth”, but can be quantified by means of FLOPs or parameters in the network.

2.2 Training the network

After materialising its architecture, the early-exit model needs to be trained on given dataset. As hinted, there are largely two ways to train early-exit networks: i) end-to-end (E2E) and ii) intermediate classifiers (IC) only. Each approach presents different trade-offs in terms of achieved accuracy vs. flexibility for target-specific adjustments. Here, we discuss these trade-offs along with orthogonal training techniques that can boost the overall accuracy of the model.

2.2.1 End-to-end vs. IC-only training.

End-to-end training. The approach comprises jointly training the network and early exits. Normally, a joint loss function is shaped which sums intermediate and the last output losses ($L^{(i)}_{\text{task}}$) in a weighted manner (Eq. 1) and then backpropagates the signals to the respective parts of the network. While the achieved accuracy of this approach can be higher both for the intermediate ($y_{i < N}$) and the last exit ($y_{N}$), this is not guaranteed due to cross-talk between exits [47]. Concretely, the interplay of multiple backpropagation signals and the relative weighting ($w_i$) of the loss components [27] needs to be carefully designed, to enable the extraction of reusable features across exits. As such, while offering a higher potential, E2E training requires manual tuning of the loss function as well as co-design of the network architecture and the populated exits [35].

$$L_{\text{E2E}}(y_0, \ldots, y_N, y) = \sum_{i=0}^{N} w_i \cdot L^{(i)}_{\text{task}}(y_i, y)$$

IC-only training. Alternatively, the backbone of the network and the early exits can be trained separately in two distinct phases. Initially, the backbone of the network, which may or may not be early-exit aware, is trained - or comes pretrained. In the subsequent phase, the backbone network is frozen\footnote{Meaning that the weights of this submodel are not updated through backpropagation.}, early-exits are attached at
different points of the network and are trained separately (Eq. 2). This means that each exit is only fine-tuning its own layers and does not affect the convergence of the rest of the network. Therefore, the last exit is left intact, there is neither cross talk between classifiers nor need to hand-tune the loss function. As such, more exit variants can be placed at arbitrary positions in the network and be trained in parallel, offering scalability in training while leaving the selection of exit heads for deployment time [41]. Thus, a “train-once, deploy-everywhere” paradigm is shaped for multi-device deployment. On the downside, this training approach is more restrictive in terms of degrees of freedom on the overall model changes, and thus can yield lower accuracy than an optimised jointly trained variant.

\[ L_{ic-only}^{(i)}(y_i, y) = L_{task}^{(i)}(y_i, y) \]  
\[ L_{distil}^{(i)}(y_i, y_j, y) = L_{task}^{(i)}(y_i, y) + \alpha L_{KL}(y_i, y_j) \]  

2.2.2 Training with distillation. An ensuing question that arises from the aforementioned training schemes is whether the early-exits differ in essence from the last one and whether there is knowledge to be distilled between them. To this direction, there has been a series of work [44, 47, 53, 58, 88] that employ knowledge distillation [23] in a self-supervised way to boost the performance of early classifiers. In such a setting, the student is typically an early exit and the teacher can be a subsequent or the last exit \( j \geq i \). As such, the loss function for each exit is shaped as depicted in Eq. 3 and two important hyperparameters emerge, to be picked at design time; namely the distillation temperature \( T \) and the alpha \( \alpha \). The temperature effectively controls how “peaky” the teacher softmax (soft labels) should be while the alpha parameter balances the learning objective between ground truth \( y \) and soft labels \( y_j \).

\[ L_{task}^{(i)}(y_i, y) = L_{task}^{(i)}(y_i, y) + \alpha L_{KL}(y_i, y_j) \]  

2.2.3 Training personalised early-exits. Hitherto, early exits have been trained for the same task uniformly across exits. However, when deploying a model in the wild, user data are realistically non-IID\(^3\) and may vary wildly from device to device. With this in mind, there has been a line of work [33, 44] that personalises early exits on user data, while retaining the performance of the last exit in the source global domain. In [44], this is effectively accomplished through IC-only training, where the backbone network is trained on a global dataset and early-exits are then trained on user-specific datasets in a supervised or self-supervised manner. In the latter case, data labels are obtained from the prediction of the last exit. Orthogonally, knowledge distillation can still be employed for distilling knowledge from the source domain to the personalised exits, by treating the last exit as the teacher [33, 44].

3 DEPLOYING FOR ADAPTIVE INFERENCE

This mode offers progressive refinement of the result through early-exiting and latency gains for easier samples.

3.1 Deploying for adaptive inference. Exit policy is defined as the criterion upon which it is decided whether an input sample propagating through the network exits at a specified location or continues. Picking the appropriate depth to exit is important both for performance and to avoid “overthinking”\(^\text{5}\) [35]. Overall, there are i) rule-based and ii) learnable exit policies.

Rule-based early-exiting. Most works in progressive inference have been employing the softmax of an exit to quantify the confidence of the network for a given prediction [2]. On the one hand, we have approaches where the criterion is a threshold on the entropy of the softmax predictions [68]. Low entropy indicates similar probabilities across classes and thus a non-confident output whereas higher entropy hints towards a single peak result. On the other hand, other approaches use the top-1 softmax value as a quantification of confidence. An overarching critique for using confidence-based criteria, however, has been the need to manually define an arbitrary threshold, along with the overconfidence of certain deep models. Solutions include calibrating the softmax output values [18] or moving to different exit schemes. Alternative rule-based exit policies include keeping per class statistics at each layer [17], calculating classifiers’ trust scores based on sample distances to a calibration set [32] or exiting after \( n \) exits agree on the result [89].

Learnable exit policies. Expectedly, one may wonder why not to learn network weights and the exit policy jointly. To this direction there has been work approaching the exit policy in differentiable [6, 60] and non-differentiable [8] ways. In essence, instead of explicitly measuring the exit’s confidence, the decision on whether to exit can be based on the feature maps of the exits themselves. The exit decision at a given classifier can be independent of the others (adhering to the Markov property) or can be modelled to also account for the outputs of adjacent exits.

3.2 Deploying the network

At this stage, an early-exit network has been trained and ready to be deployed for inference on a target device. There are largely three inference modes for early exits, each relevant to different use cases:

\( ^\text{3}\)Non Identically and Independently Distributed.

\( ^\text{5}\)Overthinking refers to the non-monotonic accuracy of ICs; i.e. later classifiers can misclassify a sample that was previously correctly classified.
throughput or accuracy objectives. In essence, tweaking i) the classifier architecture, ii) the number and positioning of early-exits and iii) exit-policy to the hardware at hand can be posed as a Design Space Exploration (DSE) problem with the goal of (co-)optimising latency, throughput, energy or accuracy given a set of restrictions, posed in the form of execution SLOs [41]. Accurately modelling this optimisation objective subject to the imposed restrictions is important for yielding efficient valid designs for the use-case at hand and shaping the Pareto front of optimal solutions.

Traversing this search space efficiently is important, especially since it needs to be done once per target device. Therefore, end-to-end training is usually avoided in favor of the more flexible IC-only approach. It should be noted, though, that the search is run prior to deployment, and it cost is amortised over multiple inferences.

Instead of searching for the optimal network configuration for fixed hardware, another set of approaches is to design the hardware specifically for early-exit networks [13, 36, 37] or co-design the network and hardware for efficient progressive inference [57, 73].

4 ADAPTIVE INFERENCE LANDSCAPE

Offline accuracy-latency trade-off. DNNs have been getting deeper and wider in their pursuit of state-of-the-art accuracy. However, such models still have to be deployable on devices in the wild. As such, optimising DNNs for efficient deployment has been an extremely active area of research. Approaches in the literature exploit various approximation and compression methods [71] to reduce the footprint of these models, including quantisation of network weights and activations [16, 46, 59] or weight sparsification and pruning [9, 19, 51]. A common denominator amongst these techniques is that they inherently trade off latency or model size with accuracy. This trade-off is exploited offline in a device-agnostic or hardware-aware [84] manner. Alongside, recent models tend to become less redundant, and thus more efficient, through careful hand-engineering [26], or automated NAS-based design [50, 66] of their architecture. These approaches remain orthogonal to adaptive inference, thus offering complementary performance gains.

Dynamic Networks. Techniques in this family take advantage of the fact that different samples may take varying computation paths during inference [52], either based on their intricacy or the capacity of the target device. Such methods include dynamically selecting specialised branches [55], skipping [70, 75, 79] or "fractionally executing" (i.e. with reduced bitwidth) [63] layers during inference, and dynamically pruning channels [11, 15, 25, 49] or selecting filters [7, 85, 86]. These approaches typically exploit trainable gating/routing components in the network architecture. This, however, complicates the training procedure and restricts post-training flexibility for efficient deployment on different hardware.

Inference Offloading. Orthogonally, there has been a series of work on adaptive inference offloading, where part of the computational graph of a DNN is offloaded to a faster remote endpoint for accelerating inference to meet a stringent SLO [34]. Some [42, 45] even combine early-exit networks with offloading.

Model Selection & Cascades. More closely related to early-exiting come approaches that train a family of models with different latency-accuracy specs, all deployed on the target device. This is achieved by trading off precision [38], resolution [83] or model capacity [20, 74] to gain speed, or by incorporating efficient specialised models [77]. At inference, the most appropriate model for each input is selected through various identification mechanisms [43, 67], or by structuring the model as a cascade and progressively propagating to more complex models until a criterion is met [39]. Although seemingly similar to early-exiting, "hard" samples may propagate through numerous cascade stages without re-use of prior computation.

Early-exiting. Early-exiting has been applied for different intents and purposes. Initially, single early-exits were devised as a mechanism to assist during training [65], as a means of enhancing the feedback signal during backpropagation and to avoid the problem of vanishing gradients. In fact, these exits were dropped during inference. Since then, however, early-exits have proven to be a useful technique of adaptive inference and have been applied successfully to different modalities and for different tasks. These previously discussed techniques are concisely presented in Table 1 and organised by their optimisation goal, input modality and trained task.

Other surveys. There have been certain previous surveys touching on the topic of early-exiting, either only briefly discussing it from the standpoint of dynamic inference networks [21] or combining it with offloading [54]. To the best of our knowledge, this is the first study that primarily focuses on early-exit networks and their design trade-offs across tasks, modalities and target hardware.

5 DISCUSSION & FUTURE DIRECTIONS

Having presented how early-exiting operates and what has been accomplished by prior work, here we discuss the main challenges and most prominent directions for future research in the field.

5.1 Open Challenges.

Modalities. A lot of research efforts in early exits have focused on the task of image classification through CNNs, and only most recently NLP through Transformer networks. However, a large variety of models (e.g. RNN, GAN, seq2seq, VAE) are deployed in the wild, addressing different tasks including object detection, semantic segmentation, regression, image captioning and many more. Such models come with their own set of challenges and require special handling on one or more of the core components of early-exit networks, which remain largely unexplored to date.

Early-exit Overhead. Attaching early exits to a backbone network introduces a workload overhead for the samples where the exit at hand cannot yield a confident enough prediction. This overhead heavily depends on the architecture of the exit, its position in the network, the effectiveness of the exit policy and the task itself. Hence, instantiating the optimal configuration of early exits on a backbone network, which balances this overhead against the performance gains from exiting early, remains a challenging task.

Architectural Search Space. As previously established, there is a large interplay between the building blocks of early-exit networks. It is therefore desirable to co-optimise many design and configuration parameters such as exit number, placement, architecture and policy. This inflates the architectural search space and makes it computationally challenging to traverse, in search for optimal configurations. Structured or NAS-based approaches to explore this space could provide an efficient solution to this end.

Training Strategy. Training early-exit networks is inherently challenging. Normally, early layers in DNNs extract lower-level appearance features, whereas deeper ones extract higher-level semantics,
important for the respective task. In the realm of early-exiting, classifiers placed shallowly are natively pushing for the extraction of semantically strong features earlier in the network, which causes tension between the gradients of different exits, and may harm the overall accuracy in the case of e2e training. Conversely, IC-only trained early exits may lead to inferior accuracy or increased overheads. Developing a training strategy that can combine the best of both worlds remains an open question.

**Exit Policy.** Current hand-tuned exit policy treat prediction “confidence” as a proxy to accuracy. Exit placement and training strategy may cause this predictions to become naturally under- or over-confident, leading to a probability distribution over layers that does not reflect the network’s innate uncertainty [18]. Developing exit strategies that better reflect the networks readiness-to-exit and potential ability to-improve its prediction by propagating to the next exit is a challenging area of research. Additionally, it is important to allow such methodologies to remain adaptable post-training, in order to facilitate efficient deployment to use-cases with varying requirements and devices with different computational capabilities.

### 5.2 Additional future directions

**Temporal-Awareness.** In video or mobile agent applications, strong correlations typically exist between temporally and spatially adjacent input samples, and hence their underlying predictive difficulty given previous predictions [28]. There is therefore space to integrate historical or codec information to further optimise early-exiting.

**Hierarchical Inference.** In latency critical applications, having some higher-level actionable result from early on may be more important than waiting for an accurate finer-grained classification prediction. Early-exit networks can facilitate this paradigm, through hierarchical inference, with earlier exits providing more abstract – and therefore easier to materialise ~ predictions (e.g. “dog”), before specialising their prediction in deeper exits (e.g. “beagle”) [3, 87].

**Personalisation.** At deployment time, deep learning models often meet narrower distributions of input samples than what they have been originally trained for (i.e. detecting a single user or their relatively stationary environment). In such cases, early exits can act as a self-acceleration mechanism, trained on the spot, e.g. through knowledge (self)-distillation from the final exit [44], to maximise their performance by specialising to the target distribution.

**Heterogeneous Federated Learning (FL).** In FL deployments, participating devices can have very heterogeneous computational and network capabilities [25]. As such, submodels of varying depth or computation may be distributed to clients to train on their local dataset, thus improving participation and fairness while avoiding stragglers.

**Probabilistic Inference.** Probabilistic models (e.g. Bayesian Neural Networks) have a native way of quantifying the predictive uncertainty of the network across all stages of inference [14]. This property of stochastic models can be exploited by the exit policy, rendering BNNs a natural fit for early exiting methodologies.

### REFERENCES

[1] Mario Almeida et al. 2019. EmBench: Quantifying Performance Variations of Deep Neural Networks Across Modern Commodity Devices. In EMDL.
[2] Konstantin Berestizhevsky et al. 2019. Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence. In ICANN.
[3] Alsallakh Bilal et al. 2017. Do convolutional neural networks learn class hierarchy? IEEE Transactions on Visualization and Computer Graphics 24, 1 (2017).
[4] Vanderlies Bonato and Christos Bouganis. 2021. Class-specific early exit design methodology for convolutional neural networks. Applied Soft Computing (2021).
[5] Sanyuan Chen et al. 2021. Don’t shoot butterfly with rifles: Multi-channel Continuous Speech Separation with Early Exit Transformer. In ICASSP.
[6] Xinshi Chen et al. 2020. Learning to stop while learning to predict. In ICML.
