Balancing Accuracy and Efficiency in Budget-Aware Early-Exiting Neural Networks

Youva Addad[0009−0009−6994−6818], Alexis Lechervy[0000−0002−9441−0187], and Frédéric Jurie[0000−0002−2686−0020]

GREYC, Normandy University, UNICAEN, ENSICAEN, UMR CNRS 6072, France
first_name.last_name@unicaen.fr

Abstract. This paper presents an Early Exit Neural Network (EENN) architecture, which enables budgeted classification by dynamically selecting the most relevant exit point for each input sample of a dataset to achieve the best performance while adhering to a pre-defined computational budget. The key contribution of this work is a novel method that jointly learns the classifier model and the sample exiting policy, in contrast to prior approaches that treated these components separately. Specifically, the paper introduces a bi-level optimization framework that simultaneously optimizes the cross-entropy loss of the classifier and the probabilities of each sample exiting at different stages of the network. This joint learning approach allows the classifier parameters and the sample-dependent exiting policy to be mutually optimized, leading to improved classification accuracy under computational constraints. The proposed EENN method is evaluated on three computer vision benchmarks - CIFAR-10, CIFAR-100, and ImageNet - and demonstrates state-of-the-art results in budgeted classification compared to existing early exit strategies. The code for this work will be made publicly available upon acceptance of the paper.

Keywords: Computer Vision, Dynamic Neural Network, Weighting Sample, Early Exit

1 Introduction

Recent image classification algorithms have achieved impressive results on benchmarks such as the Large Scale Visual Recognition Challenge (LSVRC) [20]. However, this progress has come at the cost of increasingly large and computationally intensive models. In traditional deep learning architectures, each input sample follows a fixed path through the entire network, regardless of its inherent difficulty. This approach introduces unnecessary computational overhead, especially when dealing with datasets that contain a mix of easy and challenging samples. This limitation makes real-time inference on resource-constrained platforms, such as smartphones, wearable devices, robotics, and other edge devices, almost unattainable.

To address this issue, researchers have explored various approaches to improve the inference efficiency of deep convolutional neural networks (CNNs). These include efficient architecture design [7, 36, 14], network pruning [11, 34], weight quantization [18, 17], knowledge distillation [13, 21], adaptive inference [15, 10, 33, 32, 25], and efficient deployment on hardware [30, 5].

Dynamic early-exiting networks, a type of adaptive inference, address the inefficiencies of traditional deep learning architectures by introducing branch points at different depths within the network [29]. The key idea is to intelligently select network segments to execute based on the input sample: easy-to-classify inputs can exit the network earlier, reducing computation, while complex inputs pass through deeper layers. This adaptive decision-making allows dynamic early-exiting networks to balance computational efficiency and predictive accuracy. By selectively executing only necessary segments, these models can significantly reduce computational cost while maintaining overall performance.

Furthermore, dynamic early-exiting networks can tailor the inference process to available resources and latency requirements. In time-constrained scenarios, the model can prioritize early exits for faster predictions; in resource-rich settings, it can explore deeper branches to improve accuracy. This flexibility makes dynamic early-exiting networks compelling for deployment across devices, from resource-constrained edge to powerful server systems. By intelligently allocating resources based on input, these networks can deliver efficient, high-performing inference for diverse real-world applications.

The importance of dynamic early-exiting networks has driven substantial research into improving their mechanisms. Key advancements in this active area include maximizing reuse of computations between classifiers [15], leveraging self-distillation techniques to efficiently transfer knowledge between network exits [23, 25, 9], and processing of successive small input regions to enable more dynamic exiting decisions [32]. Researchers have also explored resolution-adaptive network architectures [33, 37] and developed calibration and sample weighting methods to improve the early exit decisions [10, 26]. Furthermore, the field has seen exploration of transformer-based models as an alternative to convolutional networks for dynamic early-exiting [31, 4, 9]. This steady stream of innovations has led to ever-increasing sophistication in dynamic early-exiting network mechanisms, further enhancing their efficiency and effectiveness.

The fundamental challenge in this domain stems from the inherent train-test mismatch [8]. The classification network is typically trained without considering the constraints of a limited inference budget. Each exit classifier is optimized across the entire training dataset, without accounting for the fact that during inference, not all classifiers may encounter all types of testing data. In scenarios with resource limitations or easily manageable inputs, only the shallow layers and classifiers are activated, leading to a disparity in the data distribution between training and testing. While "easy" examples may contribute to regularizing deep classifiers during training, overemphasizing these samples can exacerbate the distribution mismatch issue.

In this work, we assume that the inference budget will be allocated by specifying the ratio of samples that must exit at each checkpoint. This means the classifiers must not only infer the correct classes but also rank the samples appropriately to meet the specified exit criteria within the given overall budget. This requirement calls for a specialized training procedure, which we address by introducing a novel gater network that is independent of both the backbone and the classifiers. The gater network takes various confidence measures as input and generates probabilities for selecting the best exit. These probabilities are then transformed into sample weights. Since the predictor network and gater network operate independently, we propose a bi-level optimization approach to co-train them.

The contributions outlined in this paper can be summarized as follows:

- We propose a novel gater network that takes multiple confidence measures as input and produces probabilities for selecting the optimal exit point during training.
- We introduce an alternative approach to modeling the likelihood of early exiting, which plays a crucial role in achieving the desired balance between accuracy and computational efficiency.
- We formulate the training process as a bi-level optimization problem, enabling the simultaneous training of the predictor network and the gater network. This optimization scheme evaluates both the accuracy and the inference cost during the training phase.
- We conduct extensive experiments on three widely-used datasets: CIFAR-10, CIFAR-100, and ImageNet. These comprehensive evaluations demonstrate the effectiveness of our proposed approach in delivering efficient and high-performing dynamic early-exiting inference.

2 Related Work

Dynamic Early Exiting is an emerging technique in deep learning that focuses on improving inference efficiency [15, 25, 33, 32, 10, 9]. It allows models to exit prediction early for certain input instances, reducing unnecessary computation without significant loss of accuracy. Dynamic Early Exiting uses adaptive criteria, such as confidence thresholds [29] or uncertainty measures [26], to decide whether to exit the inference process early for an input. This approach has several advantages, including reduced inference time [15], improved scalability [8] for resource-constrained environments [22], and potential energy savings [22]. While dynamic early exit has attracted attention for its benefits, challenges remain in finding the right criteria to balance computational gains and accuracy preservation. However, recent research [9, 10, 31, 15] has shown promising results and has been applied to several domains, including computer vision [3], natural language processing [12] and speech recognition [27]. Some authors have attempted to deviate from the multi-exit framework, for example in [35], where the authors proposed a boosting-like neural architecture, but the performance is comparable to multi-exit approaches such as MSDNet [15].

The *best performing approaches* in multi-exit architectures are based on BranchyNet [29], which was one of the first papers to propose such an efficient architecture, improved later by MSDNet [15], which introduced the concepts of *anytime classification* and *budgeted batch classification* with multiple classifiers applied adaptively during test time. Building on the foundations of this architecture, [1] has proposed a more appropriate way of fusing the network's intermediary outputs. In [33], the Resolution Adaptive Network (RANet) introduced the idea of performing resolution adaptive learning in deep CNNs within this multi-exit framework. In contrast, in L2W [10] the authors observed that MSDNet treats all samples for all exists during training, ignoring the early-exit behaviour that occurs during testing, and proposed to compensate for this by weighting training samples according to their difficulty. In the very recent paper [26], the authors proposed a novel method for estimating uncertainty in dynamic neural networks, which allows to better distinguish between easy and hard examples. This question was also investigated in [2]. It is also worth mentioning the approach presented in [24], which proposes an online knowledge distillation mechanism for multi-exit networks. Another approach using attention is the Dynamic Vision Transformer (DVT) [31] and the Coarse-to-Fine Vision Transformer (CF-ViT) [4]. Both methods share a common principle, emphasising that it is suboptimal to process all samples with the same number of tokens. The most recent model, Dynamic Perceiver (Dyn-Perceiver) [9], advocates disentangling the feature extractor and classifier branches due to the problem of classifier interference.

One limitation of these existing methods is that they do not specifically optimize the backbone network for budgeted inference. In contrast, our approach directly addresses this by jointly training the backbone and gating mechanisms to make efficient early-exiting decisions under resource constraints.

3 Presentation of the Contributions

As mentioned earlier, the key idea of our approach is to jointly train the classification network and the early exits. However, this presents a challenge, as the two components are interdependent: the performance of the early exits depends on the parameters of the classification network, and vice versa. This mutual dependence complicates the optimization process, as the optimal solution for one component cannot be determined independently of the other. Fig. (1) provides an illustration of the overall training procedure.

Early Exiting at Inference Time. Once the multi-exit neural network is trained, early exit is performed during inference by defining thresholds $\eta^{(k)}$, where k refers to the k-th exit. When processing input examples x_i, the examples are sequentially passed through each classifier exit f^k until the maximum predicted class probability, denoted as $\max_c \text{softmax}\left(f^k(x_i)\right)$, exceeds the corresponding threshold $\eta^{(k)}$. At this point, the network returns the predicted class c.

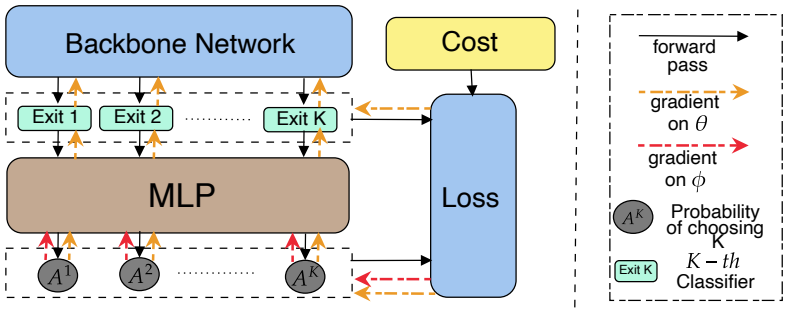

Fig. 1: Our Training Method: An Overview. The yellow arrow in the architecture represents the gradient utilized to update the entire backbone and exits. On the other hand, the red arrow signifies the gradient employed for updating the MLP scorer. The values of A^K are determined using the formula provided in Eq. (5). It's worth noting that the cost can be viewed as a scalar regularization term.

The threshold values $\eta^{(k)}$ are usually determined using validation data, with the goal of ensuring that the overall processing remains within the specified computation budget. The typical approach, which we also employ in this work, is to set the thresholds such that a fixed fraction q, with $0 < q \leq 1$, of the samples reaching a classifier will obtain a confident enough classification to exit. This fixed probability q is applied consistently across all exits. Effectively, setting q is equivalent to defining the desired computation budget, as it directly determines the expected number of samples that will exit at each stage of the network. The key point is that the classification network should be trained to provide higher classification scores for the examples that should be the first to exit.

Training Multi-Exit Networks. We have established that the classification scores provided by the network should be higher for examples that are intended to exit the network earlier. This can be the case for examples that are easier to classify or examples where further propagation through the network would not yield significant performance gains given the computational budget. This means the classification network needs to be trained with the understanding that some examples will be prioritized for earlier exit.

Let $\mathcal{D}$ be a set of training samples (x_i, y_i) for i ranging from 1 to N, where x_i denotes the input features (e.g., images) and $y_i \in \mathcal{C}$ represents the corresponding class labels. The set $\mathcal{C} = \{1, 2, \ldots, C\}$ encompasses all potential classes. The objective is to train the Multi-Exit Neural Network (MENN) model f parameterized by θ.

Let $k \in [0, K]$ be the index of each exit, where f^k (parameterized by θ^k) represents a sub-network responsible for generating the k-th output and aiming to predict the target using different computational resources. In practice, f^k often shares layers with lower-cost networks, allowing for partial reuse of computations.

This design choice enables the network to efficiently allocate computational resources based on the requirements of each input sample.

The objective stated above translates into the loss:

$$\mathcal{L}(\theta;\mathcal{D}) = \mathbb{E}_{x\sim \Pr(x)}\left[\mathbb{E}_{k\sim \Pr(k|x)}\left(\ell^{CE}(\hat{y}^k, y)\right)\right], \tag{1}$$

where $\hat{y}^k$ is the label predicted by f^k, $\mathcal{L}^{CE}(\theta;\mathcal{D}) = \mathbb{E}_{x\sim \Pr(x)}\left[\ell^{CE}(\hat{y}^k, y)\right]$ is the standard cross-entropy loss where $\ell^{CE}(\hat{y}^k, y) = -\log \Pr_{\hat{y}=y}(x;\theta)$, and $\Pr(k|x)$ represents the probability for the input x to exit at the k-th exit.

Traditionally, Early Exit Neural Network (EENN) methods have assumed a fixed probability of exiting at each layer, i.e., $\Pr(k|x) = \frac{1}{K}$. This simplifies the loss function to the average of the cross-entropies at each output. We relax this assumption of a fixed exit probability. Instead, we consider a sample-dependent probability of exiting, $\Pr(k|x)$, which can vary across samples. This sample-dependent exit probability is important because it allows the network to adapt its computational budget more flexibly to each input, while still maintaining the overall budget constraint, at train time.

Addressing Early Exit during Training. As mentioned in the previous section, we want the classification network to be learned by taking into account the fact that not all samples have an equal probability of exiting at each exit and must respect a given budget. Leveraging the work in [16], we integrate the probability of exiting in the loss, which also leads to the introduction of a per-exit cost function, as the early exiting probabilities depend on the overall budget. We propose to translate this into the following loss:

$$\mathcal{L}(\theta;\mathcal{D}) = \mathbb{E}_{x\sim \Pr(x)}\left(\mathbb{E}_{k\sim \Pr(k|x)}\left(\ell^{CE}(\hat{y}^k, y) + \lambda Cost^k\right)\right). \tag{2}$$

where λ is a hyperparameter that determines the weight given to additional costs associated with the model's predictions. It controls the emphasis placed on these costs compared to the cross-entropy loss. We will show later how to compute $\Pr(k|x)$, which is not a constant here.

Regularizing the loss. We observed that using the loss given in Eq. 2, the model frequently favors a single output based on cost considerations. This tendency limits the model's capacity to generalize effectively due to the lack of diversity in output selection. To address this limitation, we introduce an additional term in the loss function, which is the unweighted cross-entropy, as follows:

$$\mathcal{L}(\theta;\mathcal{D}) = \mathbb{E}_{x\sim \Pr(x)}\left(\mathbb{E}_{k\sim \Pr(k|x)}\left(\ell^{CE}(\hat{y}^k, y) + \lambda Cost^k\right) + K\mathbb{E}_{k\sim \text{Uniform}(K)}\left(\ell^{CE}(\hat{y}^k, y)\right)\right). \tag{3}$$

Given that $Cost^k$ is a scalar value independent of the training parameters, the loss can be written as:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(\Pr(k|x) + 1\right)\left(\ell^{CE}(\hat{y}_i^k, y_i) + \lambda Cost^k\right). \tag{4}$$

Computing the probability of early stopping. As discussed before, once the network achieves sufficient confidence, it may no longer require further consideration of subsequent exits. This rationale underlies our modeling of $\Pr(k|x_i)$, which is the likelihood for the sample x_i of exiting at exit k:

$$\Pr(k|x_i) = \begin{cases} P_i^k & \text{if } k = 1 \\ P_i^k \prod_{j=1}^{k-1}(1 - P_i^j) & \text{if } 1 < k < K \\ \prod_{j=1}^{K-1}(1 - P_i^j) & \text{if } k = K \end{cases}. \tag{5}$$

where P_i^k is the probability that $p_i^{(k)}$ is greater than the exit threshold.

Eq. (5) defines this probability differently for three cases: the first exit ($k = 1$), the last exit ($k = K$), and all other exits in between. The probability of an input exiting at the first exit, denoted as $\Pr(k = 1|x)$, is equivalent to the probability of the input being correctly confident at that exit. Conversely, for the final output, the input will only exit if it has not exit at any of the preceding output. This scenario is captured by multiplying the probabilities of not exiting at each of the previous outupt. For all intermediate exits, the input will exit if two conditions are met: first, it must be correctly confident at the current exit, and second, it must not have been correctly confident at any of the preceding outputs. This situation is represented by the product of the probability of correct confident at the current exit and the probabilities of not exiting at each of the previous exits. In summary, this equation serves as a probabilistic decision-making framework, determining the optimal exit point for an input. It strikes a balance between the goal of achieving a correct confident and the objective of minimising the computational expense associated with processing the input.

Note that it can be readily confirmed that the summation $\sum_{k=1}^{K} \Pr(k|x) = 1$.

Estimating the probability of early stopping at train time. Since we cannot know the values of the thresholds $\eta^{(k)}$ during training, it is not possible to calculate the probabilities P_i^k directly. We propose to replace this term by an action function A_i^k, which selects the outputs according to the multiple confidence scores $s_{i;\theta}^k$ and the backbone parameters represented by θ. This function, parameterised by ϕ, dictates the response of the model to the multiple confidence scores. This is achieved by employing a Multi-Layer Perceptron (MLP) on an aggregated confidence score, represented as $s_{i;\theta}^k$. Therefore, the probability $P_{i;\phi}^k = \sigma\left(\text{MLP}(s_{i;\theta}^k)\right)$, given the aggregated confidence score. σ is the sigmoid function.

The aggregation $s_{i;\theta}^k$, serves as a comprehensive measure of the model's certainty across various aspects, thereby providing a more robust estimate of the probability P_i^k. This approach allows for a more nuanced understanding of the model's performance, as it takes into account a variety of confidence scores, rather than relying on a single one. To accomplish this, we established an aggregation of confidence measures, which encompasses maximum confidence, maximum merging, and entropy. The formulation of these concepts is as follows

(inspired by the work of Ilhan et. al. [16]):

$$s_{i;\theta}^{l,max} = p_i^{(k)}, \tag{6}$$

$$s_{i;\theta}^{l,entropy} = \sum_{c\prime=0}^{C} \text{softmax}_{c'}\left(f^k(x_i)\right) \log(\text{softmax}_{c'}\left(f^k(x_i)\right)), \tag{7}$$

$$s_{i;\theta}^{l,margin} = p_i^{(k)} - \max_{c\prime \neq c} \text{softmax}_{c'}\left(f^k(x_i)\right). \tag{8}$$

In our practical implementation, we utilize Equation (9) to regulate the smoothness of probability distributions within our model. This equation embodies the softmax operation with a modification introduced by the temperature parameter (T). Integrating T into the equation provides us with precise control over how probabilities are distributed among the exits, leveraging the input data x_i. The normalization step within the equation guarantees that the resulting probabilities sum up to 1, ensuring a valid probability distribution. In our experimental setup, we set the temperature parameter to $T = 0.5$. We find this temperature value works well through a grid search.

$$A_{i;\phi}^k = \frac{Pr(k|x_i)^{1/T}}{\sum_{j=1}^{K} Pr(j|x_i)^{1/T}} \tag{9}$$

Optimizing Φ and θ.

The training of the multi-exit network involves optimizing two sets of parameters: θ, which are the parameters of the classification networks, and Φ, which are the parameters of the action function that estimates the probability for a sample to exit at a particular exit of the network. This constitutes a bi-level optimization problem, where we have two interconnected optimization problems, with one nested within the other.

In this bi-level optimization setup, the solution to the outer problem depends on the resolution of the inner problem. The process of minimizing the bi-level optimization can be described as follows:

$$\begin{aligned} &\min_{\phi} \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \left(\ell_{\theta^*}^{CE}(\hat{y}_i^k, y_i) + \lambda \text{Cost}^k\right) \left(A_{i;\phi}^k(s_{i;\theta^*}^k)\right) \\ &\text{s.t.} \min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \left(\ell_{\theta}^{CE}(\hat{y}_i^k, y_i) + \lambda \text{Cost}^k\right) \left(A_{i;\phi}^k(s_{i;\theta}^k) + 1\right) \end{aligned} \tag{10}$$

Our approach to bi-level optimization differs from that of [10], as we optimize the same loss function in both the inner and outer optimization stages, with only the parameters varying. Now, directing attention to the derivatives of the objectives concerning the parameters θ, and ϕ, the derivatives are as follows:

$$\frac{\partial \mathcal{L}^{inner}}{\partial \theta} = \sum_{i=1}^{N}\sum_{k=1}^{K} \left(A^k_{i;\phi}(s^k_{i;\theta}) + 1\right) \frac{\partial \ell^{CE}_{\theta}(\hat{y}^k_i, y_i)}{\partial \theta} + \left(\ell^{CE}_{\theta}(\hat{y}^k_i, y_i)) + \lambda Cost^k\right) \frac{\partial A^k_{i;\phi}(s^k_{i;\theta})}{\partial \theta}, \tag{11}$$

$$\frac{\partial \mathcal{L}^{outer}}{\partial \phi} = \sum_{i=1}^{N}\sum_{k=1}^{K} (\ell^{CE}_{\theta^*}(\hat{y}^k_i, y_i) + \lambda Cost^k) \frac{\partial A^k_{i;\phi}(s^k_{i;\theta^*})}{\partial \phi}. \tag{12}$$

4 Experiments

This section presents the experimental validation of the proposed method on CIFAR-10/100 [19] and ImageNet [6] datasets, and provides comparisons with recent methods in the literature. Inline with the literature on the domain, we experiment our approach in the *budgeted batch classification mode* as well as in the *anytime classification mode* (see definitions in [15]).

On the influence of $\boldsymbol{\lambda}$ The classification network is trained using a cost function that manages a trade-off between budget and classification performance. Ideally, we would need to train a different network for each value of λ, i.e., for each level of budget allocated, which would require training and storing numerous networks to cover all the possible trade-offs. However, we have observed that it is possible to find a single value of λ that offers an acceptable compromise, regardless of the budget allocated for inference. This value, $\lambda = 2$, was estimated empirically through a performance analysis on the CIFAR-100 database. All the following experiments have been conducted using a single classification network learned with $\lambda = 2$.

On the importance of training the classification network with budget-aware constraints The main idea of this paper is that it is important to train the classification network while considering budget constraints, as opposed to methods that first train the classifier and then add techniques to refine the output rules. To validate this hypothesis, we conducted an experiment on the CIFAR databases, the results of which are part of Figure 3. This figure compares the performance of our proposed method (called "MSDNet+Ours") with the same method when the classification network is first learned and then frozen (called "MSDNet frozen+Ours"). We observe that the gain is significant, on the order of +2% on CIFAR 100, across various budget levels.

On the features used for estimating the probability of early stopping at train time As discussed in the previous section, during training we cannot know which example will output on which layer during inference. Instead, we infer

Input	Top-1 Acc (%).			
	30M	50M	70M	90M
Confidence Only	70.84	76.39	79.32	80.08
Margin Only	71.49	76.07	78.73	79.6
Entropy Only	71.55	76.84	79.43	80.37
Confidence and Margin	71.63	76.48	78.91	79.73
Confidence and Entropy	**72.28**	76.81	**79.53**	**80.41**
Margin and Entropy	71.67	76.56	78.79	79.7
All	71.57	**76.89**	79.19	79.7

Table 1: Accuracy on CIFAR-100 in budgeted batch classification, comparing different combinations of features used to determine the probability of exiting the network: 'confidence', 'margin', and 'entropy' (as described in Section 3).

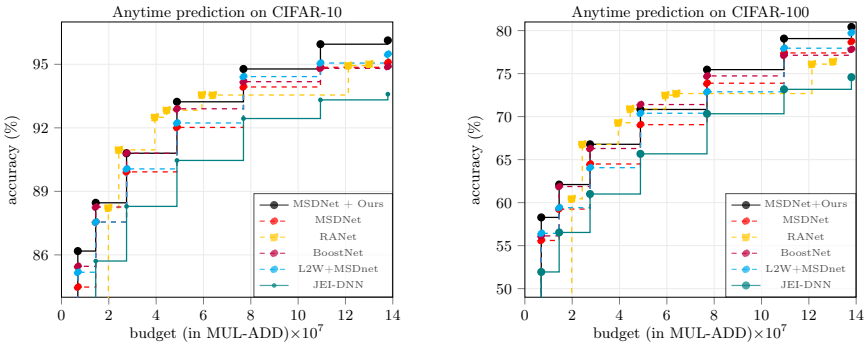

Fig. 2: Accuracy (top-1) of anytime prediction models as a function of computational budget on CIFAR-10 (top) and CIFAR-100 (bottom) . Higher is better.

this information from certain features that we believe correlate with the early stopping probability, namely: 'confidence', 'margin', and 'entropy' (as described in Section 3). Table 1 shows the performance of the classifier for different budgets as a function of the information used to infer the probability of exiting. The best model across all budgets is obtained by using both the 'confidence' score and 'entropy' as inputs. In general, incorporating entropy into the input results in the highest score compared to using other features alone. This is the setting used in the following experiments.

Comparisons with state-of-the-art multi-exit achitectures We compare our model to the best performing related works, namely MSDNet [15], RANet [33], BoostNet [35], L2W [10], and JEI-DNN [28]. In addition, we compare our results with those of post-hoc methods, specifically Calibrated-DNN proposed by Meronen et al. [26] and EENet proposed by Ilhan et al. [16].

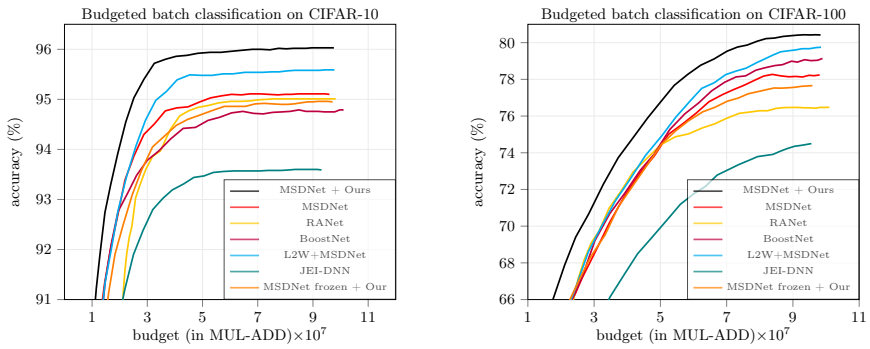

Fig. 3: Accuracy (top-1) of budgeted batch classification models as a function of average computational budget per image on CIFAR-10 (left) and CIFAR-100 (right).

For Anytime Prediction, our proposed training method coupled with MSDNet [15] surpasses the previous state-of-the-art for both CIFAR-10 and CIFAR-100 datasets, even for the first exits. The only exception is exit 3 on CIFAR-10, where RANet performs slightly better. Our method is depicted by the black curve in Fig. 2 entitled 'MDSNet+Ours'. On CIFAR-10, our method achieves an accuracy of 96.13% for its last exit with $137M$ FLOPs. For a budget of $27M$ FLOPs, we achieve an accuracy of 90.80% compared to RANet's 90.96% with a similar number of FLOPs. In comparison to JEI-DNN [28] trained under the same conditions, we achieve an improvement of approximately 2% to 3% in accuracy across all exits. On CIFAR-100, our method attains an accuracy of 80.43% with $137M$ FLOPs. Compared to the traditional training of MSDNet, our proposed method achieves an improvement of approximately 1% to 1.5% in accuracy, depending on the exit.

In the context of budgeted batch classification, our proposed method demonstrates superior performance compared to all other methods for both CIFAR-10 and CIFAR-100 datasets. For CIFAR-10, our method achieves a classification rate of 95.03% with a budget of only $25M$ FLOPs, which is significantly more efficient than MSDNet, requiring twice as many FLOPs to achieve the same level of performance. Furthermore, RANet and BoostNet require approximately $75M$ FLOPs to attain the same performance level as our method. To reach the same accuracy as JEI-DNN, which is approximately 93.4%, our method requires only $28M$ FLOPs, which is about 3× fewer FLOPs. For CIFAR-100, our method requires only $90M$ FLOPs to attain an accuracy of 80.43%, and only $58M$ FLOPs to achieve the same level of performance as MSDNet, which takes approximately 1.5× more FLOPs.

In addition to comparing our method with other approaches, we have also evaluated it against post-hoc methods, which involve applying post calibration

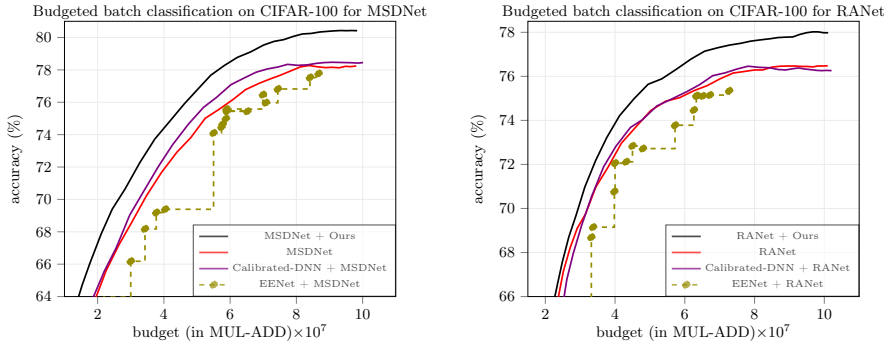

Fig. 4: Compare MSDNet and RANet using both the post-hoc method and our training approach on CIFAR 100 dataset. Higher is better.

on the scores. Fig. 4 illustrates the comparison with two such methods, namely Calibrated-DNN [26] and EENet [16]. The figure presents two models, with MSDNet [15] on the left and RANet [33] on the right. Our proposed method demonstrates significant improvement in the performance of both architectures and surpasses the current state-of-the-art methods in both cases. In the scenario of MSDNet [15], Calibrated-DNN noticeably improves inference performance, particularly in budgets exceeding $60M$ Flops, although still falling short of our method's enhancement. When MSDNet [15] is combined with EENet [16], the resulting method exhibits inferior performance compared to the classic approach. As for RANet [33], our approach enhances its performance by a margin of 0.5% to 1.5% across all budgets. Notably, Calibrated-DNN achieves nearly equivalent performance to traditional training. Similarly, applying EENet [16] to RANet [33] yields poor result as classical training.

Finally, Fig. 5 reports the performance of the proposed method on the ImageNet dataset. Once again, our proposed method demonstrates superior performance compared to other state-of-the-art approaches, whether for Anytime Prediction or Budgeted Batch Classification. In Anytime Prediction, our method outperforms L2W-MSDNet and BoostNet, two approaches that bear the closest resemblance to ours. In the realm of Budgeted Batch Classification, our primary focus, our approach achieves a slightly higher accuracy of around 0.5% to 1% compared to alternative methods. Noteworthy is our method's efficiency, requiring only 1.95×10^9 Flops to achieve peak accuracy, representing a 25% reduction in Flops compared to L2W-MSDNet. Regarding BoostNet, our method achieves the maximum performance reached by BoostNet, approximately 78.54%, with only 1.87×10^9 Flops, marking a reduction of approximately 30% in Flops.

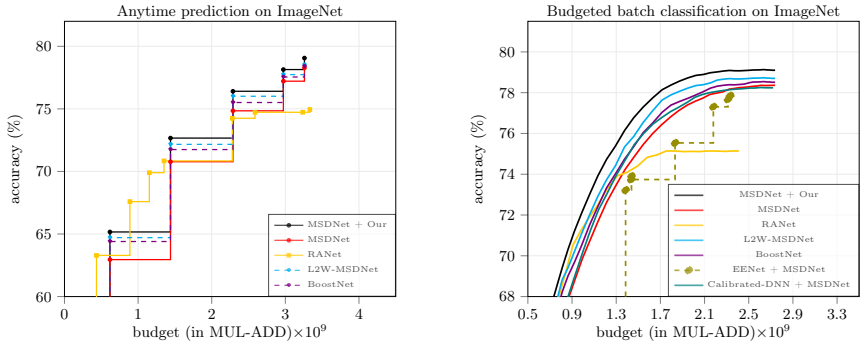

Fig. 5: Top-1 accuracy on ImageNet, plotted as a function of computational budget.

5 Conclusions

This paper introduces a training approach for Early Exit Neural Networks (EENN) specifically designed for budgeted classification tasks. The primary goal of our method is to align the behavior of the training and inference steps. By enhancing traditional Early Exit models with the integration of the forward cost and a network that calibrates sample difficulty, we achieve improved classification accuracy while respecting computational constraints. Extensive evaluations on CIFAR-10, CIFAR-100, and ImageNet benchmarks demonstrate that our approach sets a new state-of-the-art for budgeted classification, consistently outperforming existing early exit strategies.

Acknowledgments. Research reported in this paper was supported by the ANR under award number ANR-19-CHIA-0017 and was performed using computing resources of CRIANN.

References

1. Addad, Y., Lechervy, A., Jurie, F.: Multi-exit resource-efficient neural architecture for image classification with optimized fusion block. In: ICCV Workshops (2023)
2. Agarwal, C., D'souza, D., Hooker, S.: Estimating Example Difficulty using Variance of Gradients. In: CVPR (2022)
3. Bi, Y., Xue, B., Mesejo, P., Cagnoni, S., Zhang, M.: A Survey on Evolutionary Computation for Computer Vision and Image Analysis: Past, Present, and Future Trends. IEEE Transactions on Evolutionary Computation **27**(1) (2023)
4. Chen, M., Lin, M., Li, K., Shen, Y., Wu, Y., Chao, F., Ji, R.: Cf-vit: A general coarse-to-fine method for vision transformer. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 37 (2022)

5. Chen, T., Moreau, T., Jiang, Z., Shen, H., Yan, E.Q., Wang, L., Hu, Y., Ceze, L., Guestrin, C., Krishnamurthy, A.: Tvm: end-to-end optimization stack for deep learning. arXiv preprint arXiv:1802.04799 (2018)
6. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
7. Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., Xu, C.: Ghostnet: More features from cheap operations. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
8. Han, Y., Huang, G., Song, S., Yang, L., Wang, H., Wang, Y.: Dynamic neural networks: A survey. IEEE Transactions on Pattern Analysis & Machine Intelligence **44**(11), 7436–7456 (nov 2022). https://doi.org/10.1109/TPAMI.2021.3117837
9. Han, Y., Han, D., Liu, Z., Wang, Y., Pan, X., Pu, Y., Deng, C., Feng, J., Song, S., Huang, G.: Dynamic perceiver for efficient visual recognition (Jun 2023)
10. Han, Y., Pu, Y., Lai, Z., Wang, C., Song, S., Cao, J., Huang, W., Deng, C., Huang, G.: Learning to weight samples for dynamic early-exiting networks. In: Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI. pp. 362–378. Springer-Verlag (11 2022)
11. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
12. Hedderich, M.A., Lange, L., Adel, H., Strötgen, J., Klakow, D.: A survey on recent approaches for natural language processing in low-resource scenarios. In: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 2545–2568. Association for Computational Linguistics, Online (Jun 2021)
13. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS Deep Learning and Representation Learning Workshop (2015)
14. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (Apr 2017), arXiv:1704.04861 [cs]
15. Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., Weinberger, K.: Multi-scale dense networks for resource efficient image classification. In: International Conference on Learning Representations (2018)
16. Ilhan, F., Chow, K.H., Hu, S., Huang, T., Tekin, S., Wei, W., Wu, Y., Lee, M., Kompella, R., Latapie, H., Liu, G., Liu, L.: Adaptive deep neural network inference optimization with eenet. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV). pp. 1373–1382 (January 2024)
17. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)
18. Jung, S., Son, C., Lee, S., Son, J., Han, J.J., Kwak, Y., Hwang, S.J., Choi, C.: Learning to quantize deep networks by optimizing quantization intervals with task loss. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
19. Krizhevsky, A.: Learning multiple layers of features from tiny images pp. 32–33 (2009)
20. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.) Advances in Neural Information Processing Systems. vol. 25 (2012)

21. lan, x., Zhu, X., Gong, S.: Knowledge distillation by on-the-fly native ensemble. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018)
22. Laskaridis, S., Kouris, A., Lane, N.D.: Adaptive inference through early-exit networks: Design, challenges and directions. In: Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning (2021)
23. Lee, H., Lee, J.S.: Students are the Best Teacher: Exit-Ensemble Distillation with Multi-Exits (Apr 2021). https://doi.org/10.48550/arXiv.2104.00299
24. Lee, H., Lee, J.S.: Rethinking Online Knowledge Distillation with Multi-exits. In: Wang, L., Gall, J., Chin, T.J., Sato, I., Chellappa, R. (eds.) Computer Vision – ACCV 2022, vol. 13846, pp. 408–424. Springer Nature Switzerland (2023)
25. Li, H., Zhang, H., Qi, X., Yang, R., Huang, G.: Improved techniques for training adaptive deep networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019)
26. Meronen, L., Trapp, M., Pilzer, A., Yang, L., Solin, A.: Fixing Overconfidence in Dynamic Neural Networks (Apr 2023), arXiv:2302.06359 [cs]
27. Prabhavalkar, R., Hori, T., Sainath, T.N., Schlüter, R., Watanabe, S.: End-to-End Speech Recognition: A Survey (Mar 2023), arXiv:2303.03329 [cs, eess]
28. Regol, F., Chataoui, J., Coates, M.: Jointly-learned exit and inference for a dynamic neural network. In: The Twelfth International Conference on Learning Representations (ICLR) (2024)
29. Teerapittayanon, S., McDanel, B., Kung, H.T.: Branchynet: Fast inference via early exiting from deep neural networks. 2016 23rd International Conference on Pattern Recognition (ICPR) pp. 2464–2469 (2016)
30. Thakkar, M., Thakkar, M.: Introduction to core ml framework. Beginning Machine Learning in iOS: CoreML Framework (2019)
31. Wang, Y., Huang, R., Song, S., Huang, Z., Huang, G.: Not all images are worth 16x16 words: Dynamic transformers for efficient image recognition. In: Advances in Neural Information Processing Systems (NeurIPS) (2021)
32. Wang, Y., Lv, K., Huang, R., Song, S., Yang, L., Huang, G.: Glance and focus: A dynamic approach to reducing spatial redundancy in image classification. In: Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020
33. Yang, L., Han, Y., Chen, X., Song, S., Dai, J., Huang, G.: Resolution adaptive networks for efficient inference. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
34. Yang, L., Jiang, H., Cai, R., Wang, Y., Song, S., Huang, G., Tian, Q.: Condensenet v2: Sparse feature reactivation for deep networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3569–3578 (June 2021)
35. Yu, H., Li, H., Hua, G., Huang, G., Shi, H.: Boosted dynamic neural networks. In: Williams, B., Chen, Y., Neville, J. (eds.) Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023,. pp. 10989–10997. AAAI Press (2023)
36. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2018)
37. Zhu, M., Han, K., Wu, E., Zhang, Q., Nie, Y., Lan, Z., Wang, Y.: Dynamic Resolution Network. In: Advances in Neural Information Processing Systems. vol. 34, pp. 27319–27330. Curran Associates, Inc. (2021)

Balancing Accuracy and Efficiency in Budget-Aware Early-Exiting Neural Networks Supplementary Materials

A Datasets

The CIFAR-10/100 dataset consist of 32×32 RGB natural images, respectively representing 10 and 100 distinct classes. Each dataset comprises 50,000 training images and 10,000 testing images. Following the approach detailed in [15], a validation set is formed by selecting 5,000 training images to determine the optimal confidence threshold for adaptive inference. In contrast, the ImageNet dataset offers 1.2 million images distributed across 1,000 classes for training, along with an additional 50,000 images for validation purposes. In tasks involving adaptive inference, the initial validation set is repurposed as the testing dataset. Moreover, a separate validation subset is created by extracting 50,000 training images to ascertain the optimal confidence threshold.

B Training and Data Augmentation

Our training scheme employs stochastic gradient descent (SGD) for CIFAR-10/100 and AdamW for ImageNet, both with a cosine learning rate schedule. The batch sizes are set to 64 for the CIFAR datasets and 1024 for the ImageNet dataset. The training protocols encompass a momentum value of 0.9 and a weight decay of 10^{-4} for CIFAR. For ImageNet, the weight decay is set at 0.05, and there are 20 epochs of warm-up. Notably, for the CIFAR datasets, model training is started from scratch, spanning 300 epochs, starting with a learning rate of 0.1 and decreasing it to 1×10^{-3}. In contrast, for the ImageNet dataset, an initial learning rate of 10^{-3} is used decreased to 1×10^{-6}. To enrich the datasets, we employ data augmentation **(DA)** strategies following the methodology. This includes randomly cropping images to 32×32 pixels after applying zero padding (4 pixels on each side). Additionally, images are horizontally flipped with a 50% probability, and RGB channels are normalized by subtracting the respective channel mean and dividing by its standard deviation. To further improve the model performance, we integrate established techniques such as Mixup with $\lambda = 0.8$, Cutmix with $\lambda = 1.0$, RandAugment, the variant labeled as 'rand-m9-mstd0.5-inc1' is used, Random Erasing with $p = 0.25$, color jitter and network regularization with label smoothing which has 0.1 for value. We employ Exponential Moving Average (EMA) with momentum equal to 0.9999 because it mitigates overfitting in larger models.

In our training setup, we employed specific configurations tailored to our task. For CIFAR-10/100, we utilized Confidence and Entropy as input features,

while for ImageNet, we included Confidence, Margin, and Entropy. Our MLP architecture consisted of a single hidden layer with a dimensionality of 500 for both datasets. We trained the model with a learning rate initially set to 1×10^{-3}, which was gradually reduced using a cosine scheduler to 1×10^{-5} for both datasets. Additionally, we applied regularization with parameter λ to control the cost function, and temperature of T for the probability calculation. Due to the ease of training a Multilayer Perceptron (MLP), we opted to update our network, parameterized with ϕ, at intervals of 10 for CIFAR-10/100 and intervals of 100 for ImageNet.

C Model Configuration

We compare our model to the best performing related works, namely MSDNet [15], RANet [33], and L2W-MSDNet[10]. Furthermore, we assess various alternative post-hoc methods, including calibrated-DNN [26] and EENet [16]. Throughout all experiments, we strictly follow the predefined architecture and employ the provided code across all methods, maintaining identical hyperparameters. Given our primary aim of improving Budgeted Batch Classification, we maintain consistency by employing the identical architecture for Anytime Prediction. For the CIFAR dataset, we employ MSDNet [15] using three distinct scales: 32×32, 16×16, and 8×8. The associated input channels are $(16, 32, 64)$, and growth rates are $(6, 12, 24)$. Following training, the MSDNet [15] consist of 7 classifiers. Each classifier corresponds to $\sum_{i=1}^{K} i$-th layer. Regarding the ImageNet dataset, the MSDNet [15] architecture employs four scales. Here, the k-th classifier is situated at the $(7 \times k + 7)$-th layer, where $k = 5$. For RANet [33] applied to CIFAR10/100, we utilized Model-C-3, which comprises four base scales: 32×32, 16×16, 8×8, and 8×8. The associated input channels are $(16, 16, 32, 64)$, with growth rates of $(6, 6, 12, 24)$, respectively. For ImageNet, we employed Model-I-2, featuring four scales with base feature channel numbers set at 64, 128, 128, and 256. The growth rates for these scales were specified as 16, 32, 32, and 64, respectively.