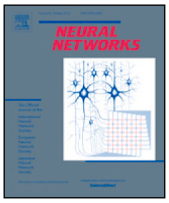
Neural Networks 168 (2023) 580–601

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Zero time waste in pre-trained early exit neural networks

Bartosz Wójcik [a,b,e,*], Marcin Przewięźlikowski [a,b,e,1], Filip Szatkowski [c,e,1], Maciej Wołczyk [a,b], Klaudia Bałazy [a,b], Bartłomiej Krzepkowski [d,e], Igor Podolak [a], Jacek Tabor [a], Marek Śmieja [a], Tomasz Trzciński [c,a,e,f]

[a] *Faculty of Mathematics and Computer Science, Jagiellonian University, Poland*
[b] *Doctoral School of Exact and Natural Sciences, Jagiellonian University, Poland*
[c] *Warsaw University of Technology, Poland*
[d] *University of Warsaw, Poland*
[e] *IDEAS NCBR, Poland*
[f] *Tooploox, Poland*

ARTICLE INFO

Keywords:
Early-exiting networks
Dynamic neural networks
Conditional computation
Zero waste models
Deep learning

ABSTRACT

The problem of reducing processing time of large deep learning models is a fundamental challenge in many real-world applications. Early exit methods strive towards this goal by attaching additional Internal Classifiers (ICs) to intermediate layers of a neural network. ICs can quickly return predictions for easy examples and, as a result, reduce the average inference time of the whole model. However, if a particular IC does not decide to return an answer early, its predictions are discarded, with its computations effectively being wasted. To solve this issue, we introduce Zero Time Waste (ZTW), a novel approach in which each IC reuses predictions returned by its predecessors by (1) adding direct connections between ICs and (2) combining previous outputs in an ensemble-like manner. We conduct extensive experiments across various multiple modes, datasets, and architectures to demonstrate that ZTW achieves a significantly better accuracy vs. inference time trade-off than other early exit methods. On the ImageNet dataset, it obtains superior results over the best baseline method in 11 out of 16 cases, reaching up to 5 percentage points of improvement on low computational budgets.

1. Introduction

Deep learning models achieve tremendous successes across a multitude of tasks, yet their training and inference often yield high computational costs and long processing times (He, Zhang, Ren, & Sun, 2016; Krizhevsky, Sutskever, & Hinton, 2017). For some applications, however, efficiency remains a critical challenge, *e.g.* to deploy a reinforcement learning (RL) system in production the policy inference must be done in real-time (Dulac-Arnold, Mankowitz, & Hester, 2019), while the robot performances suffer from the delay between measuring a system state and acting upon it (Schuitema, Buşoniu, Babuška, & Jonker, 2010). Similarly, long inference latency in autonomous cars could impact their ability to control the speed (Hester & Stone, 2013) and lead to accidents (Grigorescu, Trasnea, Cocias, & Macesanu, 2020; Jung, Hwang, Shin, & Shim, 2018).

Typical approaches to reducing the processing complexity of neural networks in latency-critical applications include compressing the model (Lee, Kim, Kim, Jo, & Yoo, 2021; Livne & Cohen, 2020; Zhang, He, & Li, 2019) or approximating its responses (Kouris, Venieris, Rizakis, & Bouganis, 2019). For instance, Livne and Cohen (2020) propose to compress a RL model by policy pruning, while Kouris et al. (2019) approximate the responses of LSTM-based modules in self-driving cars to accelerate their inference time. While those methods improve processing efficiency, they still require samples to pass through the entire model. In contrast, biological neural networks leverage simple heuristics to speed up decision making, *e.g.* by shortening the processing path even in case of complex tasks (Ariely & Norton, 2011; Gigerenzer & Gaissmaier, 2011; Kahneman, 2017).

This observation led a way to the inception of the so-called *early exit* methods, such as Shallow-Deep Networks (SDN) (Kaya, Hong, & Dumitras, 2019) and Patience-based Early Exit (PBEE) (Zhou et al., 2020), that attach simple classification heads, called internal classifiers (ICs), to selected hidden layers of neural models to shorten the processing time. If the prediction confidence of a given IC is sufficiently high, the response is returned, otherwise, the example is passed to the subsequent classifier. Although these models achieve promising results,

* Corresponding author at: Faculty of Mathematics and Computer Science, Jagiellonian University, Poland.
E-mail address: b.wojcik@doctoral.uj.edu.pl (B. Wójcik).
[1] Equal contribution.

https://doi.org/10.1016/j.neunet.2023.10.003
Received 22 May 2023; Received in revised form 29 August 2023; Accepted 4 October 2023
Available online 9 October 2023

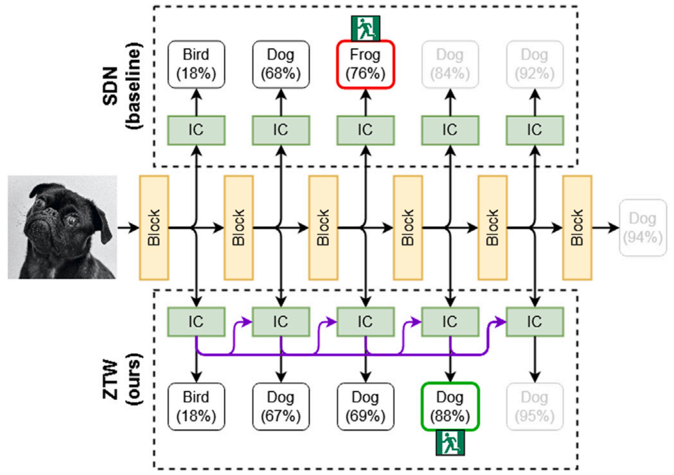


Fig. 1. Comparison of the proposed ZTW (bottom) with a conventional early-exit model, SDN (top). In both approaches, internal classifiers (ICs) attached to the intermediate hidden layers of the base network allow us to return predictions quickly for examples that are easy to process. While SDN discards predictions of uncertain ICs (*e.g.* below a threshold of 75%), ZTW reuses computations from all previous ICs, which prevents information loss and waste of computational resources.

they discard the response returned by early ICs in the evaluation of the next IC, disregarding potentially valuable information, e.g. decision confidence, and wasting computational effort already incurred.

Motivated by the above observation, we postulate to look at the problem of neural model processing efficiency from the *information recycling* perspective and introduce a new family of *zero waste models*. More specifically, we investigate how information available at different layers of neural models can contribute to the decision process of the entire model. To that end, we propose Zero Time Waste (ZTW), a method for an intelligent aggregation of the information from previous ICs. A high-level view of our model is given in Fig. 1. Our approach relies on combining ideas from networks with skip connections (Wang, Yu, Dou, Darrell and Gonzalez, 2018), gradient boosting (Bentéjac, Csörgő, & Martínez-Muñoz, 2020), and ensemble learning (Fort, Hu, & Lakshminarayanan, 2019; Lakshminarayanan, Pritzel, & Blundell, 2017). Skip connections between subsequent ICs (which we call *cascade connections*) allow us to explicitly pass the information contained within low-level features to a deeper classifier, which forms a cascading structure of ICs. In consequence, each IC improves on the prediction of previous ICs, as in gradient boosting, instead of generating them from scratch. To give the opportunity for every IC to explicitly reuse predictions of all previous ICs, we additionally build an ensemble of shallow ICs. With these improvements, we aim to decrease the computational cost of pre-trained models while maintaining the desired performance on a given task.

We evaluate our approach on standard classification benchmarks, such as ImageNet, as well as on the more latency-critical applications, such as reinforcement-learned models for interacting with sequential environments. Results show that ZTW is able to save much more computation while preserving accuracy than current state-of-the-art early exit methods. In order to better understand where the improvements come from, we introduce Hindsight Improvability, a metric for measuring how efficiently the model reuses information from the past.

In this paper we extend our previous work (Wołczyk et al., 2021) in the following regards:

- We extend the evaluation by adding two recent early-exit methods, updating the backbone models to modern and state-of-the-art computer vision (CV) architectures, using larger datasets, and including the Natural Language Processing (NLP) domain in our evaluation.
- We investigate the behavior early exit models in the transfer learning setting, showing how ZTW is able to outperform other methods by reusing features from shallower layers.
- We demonstrate how performance of ZTW can be improved with knowledge distillation between ICs.
- We present an empirical analysis of optimal head architecture and placement choice, and perform detailed ablation studies that justify the design choices of ZTW.
- We identify the overconfidence of the early IC phenomenon as the cause of the classification failures specific to early-exit models.

2. Related work

The drive towards reducing computational waste in deep learning literature has mainly concentrated on reducing the inference time. Numerous approaches for accelerating deep learning models focus on building more efficient architectures (Howard et al., 2017; Liu et al., 2021; Tan & Le, 2019, 2021), reducing the number of parameters (He, Zhang, & Sun, 2017) or distilling knowledge to smaller networks (Hinton, Vinyals, & Dean, 2015; Jiao et al., 2020; Sanh, Debut, Chaumond, & Wolf, 2019). Thus, they decrease inference time by reducing the overall complexity of the model instead of using the conditional computation framework of adapting computational effort to each example. As such we find them orthogonal to the main ideas of our work, e.g. we show that applying our method to architectures designed for efficiency, such as EfficientNet (Tan & Le, 2021), leads to even further acceleration.

Early Exiting The work of Panda, Sengupta, and Roy (2016) was one of the first that proposed to use linear classifiers as early exiting heads. In BranchyNet (Teerapittayanon, McDanel, & Kung, 2016) a loss function consisting of a weighted sum of individual head losses was utilized in training, and entropy of the head prediction was used for the early exit criterion. Berestizshevsky and Even (2019) proposed to use confidence (maximum of the softmax output) instead. Recently, Yu, Li, Hua, Huang, and Shi (2023) proposed to train each block of the network, along with its corresponding head, in an additive manner inspired by gradient boosting. A broader overview of early exit methods is available in Scardapane, Scarpiniti, Baccarelli and Uncini (2020).

Several works proposed specialized architectures for conditional computation which allow for multi-scale feature processing (Huang et al., 2018; Wang et al., 2020; Yang et al., 2020), and developed techniques to train them more efficiently by passing information through the network (Li, Zhang, Qi, Ruigang, & Huang, 2019; Phuong & Lampert, 2019). However, in this paper, we consider the case of increasing inference speed of a pre-trained network based on an architecture which was not built with conditional computation or even efficiency in mind. We argue that this is a practical use case, as this approach can be used to a wider array of models. As such, we do not compare with these methods directly.

Shallow-Deep Networks (SDN) (Kaya et al., 2019) is a conceptually simple yet effective method, where the comparison of confidence with a fixed threshold is used as the exit criterion. The authors attach internal classifiers to layers selected based on the number of compute operations needed to reach them. The answer of each head is independent of the answers of the previous heads, although in a separate experiment the authors analyze the measure of disagreement between the predictions of final and intermediate heads.

Zhou et al. (2020) propose Patience-based Early Exit (PBEE) method, which terminates inference after t consecutive unchanged answers, and show that it outperforms SDN on a range of NLP tasks. The idea of checking for agreement in preceding ICs is connected to our approach of reusing information from the past. However, we find that applying PBEE in our setting does not always work better than SDN.

Concurrently to our original work, Liao et al. (2021) proposed to approximate states of deeper heads, aggregate them with previous states, and use it for the final prediction of the head. In their method, the modules that approximate *future states* are trained by minimizing the cosine similarity between approximations and actual states. The relative contribution of past states and approximated future states is weighted with a separate head. The approach is similar to ours in reusing the information from previous heads, but it additionally attempts to approximate future states. Due to its similarity we consider it as an appropriate baseline for the evaluation of our method.

Han et al. (2022) point out the discrepancy between training and testing of early exiting networks, and propose a framework for weighting losses of each sample between different exits. To achieve this, they employ meta-learning to train a *weight prediction network*, which directly outputs the weights for each head. This approach, in turn, is completely different than ours, which is also the reason for including it in our comparison.

Conditional Computation The broader field of conditional computation was first proposed for deep neural networks in Bengio, Léonard, and Courville (2013) and Davis and Arel (2013), and since then many sophisticated methods have been developed in this field. A recent survey categorizes the various approaches into several groups (Han et al., 2021). While most works use conditional computation for reducing computational cost, there are other applications of this paradigm such as federated learning (Sun & Ochiai, 2022; Zhang et al., 2021) or continual learning (Lin, Fu, & Bengio, 2019).

Graves (2016) proposes to calculate a halting score in a RNN to dynamically control the effective depth of the model. Banino, Balaguer, and Blundell (2021) improve upon this by introducing probabilistic approach that stabilizes the training. Each recurrent module returns an additional scalar, which is then used to randomly decide whether to halt or to continue applying the same module. Wang, Yu et al. (2018) and Veit and Belongie (2018) also use a variant of skipping residual blocks, but by using a gating network that takes the current feature representation as input.

Another line of works dynamically reduces the width of the model, for example by selecting the filters to be executed in a convolutional layer (Herrmann, Bowen, & Zabih, 2020; Lin, Rao, Lu, & Zhou, 2017; Liu, Wang, Han, Xu, & Xu, 2019). Li et al. (2021) improve upon these and make it more hardware-efficient by selecting a consecutive slice of the filters. For transformer-based architectures, the common Mixture of Experts layers (Shazeer et al., 2017) can be adapted after training to dynamically adjust the amount of computation depending on the difficulty of the input (Nie et al., 2021).

It is also possible to exploit the spatial redundancies and allocate different amount of compute to different regions of the input, for example by computing a halting score for each position of the input image (Figurnov et al., 2017). Verelst and Tuytelaars (2020) introduce a residual block that applies the convolutional filters only to the selected positions of the input. Alternatively, a similar effect can be achieved with an additional network, which is trained as a reinforcement learning agent, that repeatedly selects the small patches of the original image to use for the refinement of the previous classification (Wang et al., 2020).

While being adaptive in compute, these approaches either modify or propose a novel architecture that requires training the model from scratch. On the other hand, the early exit methods are directly applicable to existing architectures and thus allow use of pre-trained weights for models that would otherwise be prohibitively costly to train.

Ensembles Ensembling is typically used to improve the accuracy of machine learning models (Dietterich, 2000). Lakshminarayanan et al. (2017) showed that it also greatly improves calibration of deep neural networks. There were several attempts to create an ensemble from different layers of a network. Scardapane, Comminiello, Scarpiniti, Baccarelli and Uncini (2020) adaptively exploit outputs of all internal classifiers, albeit not in a conditional computation context. Phuong and Lampert (2019) used averaged answers of heads up to the current head for anytime-prediction, where the computational budget is unknown. Besides the method being much more basic, their setup is notably different from ours, as it assumes the same computational budget for all samples no matter how difficult the example is. Finally, contemporary work of Sun et al. (2021) also propose to construct an ensemble out of individual heads, but additionally enforce diversity among its members and use voting as the exit criterion.

3. Zero Time Waste

Our goal is to reduce computational costs of neural networks by minimizing redundant operations and information loss. To achieve it, we use the conditional computation setting, in which we dynamically select the route of an input example in a neural network. By controlling the computational path, we can decide how the information is stored and utilized within the model for each particular example. Intuitively, difficult examples require more resources to process, but using the same amount of compute for easy examples is wasteful.

We assume a practical setting in which the dynamic model is built using a pre-trained model. In addition to enabling the reuse of published weights of almost any trained model, it allows us to easily apply the same techniques to setting such as reinforcement learning environments. In order to adapt already trained models to conditional computation setting, we attach and train early exit classifier heads on top of several selected layers, without changing the parameters of the base network. During inference, the whole model exits through one of them when the response is likely enough, thus saving computational resources.

Formally, we consider a multi-class classification problem, where $x \in \mathbb{R}^D$ denotes an input example and $y \in \{1, \dots, K\}$ is its target class. Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^K$ be a pre-trained neural network with logit output designed for solving the above classification task. The weights θ will not be modified.

3.1. Model overview

Following typical early exit frameworks, we add M shallow Internal Classifiers, $\mathrm{IC}_1, \dots, \mathrm{IC}_M$, on intermediate layers of f_θ. Namely, let g_{ϕ_m}, for $m \in \{1, \dots, M\}$, be the mth IC network returning K logits, which is attached to hidden layer f_{θ_m} of the base network f_θ. The index m is independent of f_θ layer numbering. In general, M is lower than the overall number of f_θ hidden layers since we do not add ICs after every layer.

Although using ICs to return an answer early can reduce overall computation time (Kaya et al., 2019), in a standard setting each IC makes its decision independently, ignoring the responses returned by previous ICs. As we show in Section 5.1, early layers often give correct answers for examples that are misclassified by later classifiers, and hence discarding their information leads to waste and performance drops. To address this issue, we need mechanisms that collect the information from the first $(m-1)$ ICs to inform the decision of IC_m. For this purpose, we introduce two complementary techniques: *cascade connections* and *ensembling*, and show how they help reduce information waste and, in turn, accelerate the model.

Cascade connections directly transfer the already inferred information between consecutive ICs to avoid the need to recompute it again. In this way they improve the performance of initial ICs that lack enough predictive power to classify correctly based on low-level features. *Ensembling* of individual ICs improves performance as the number of members increases, thus showing greatest improvements in the deeper part of the network. The full model is visualized in Fig. 2 where cascade connections are used first to pass already inferred information to later ICs, while ensembling is utilized to conclude the IC prediction. The details on these two techniques are presented in the following paragraphs.

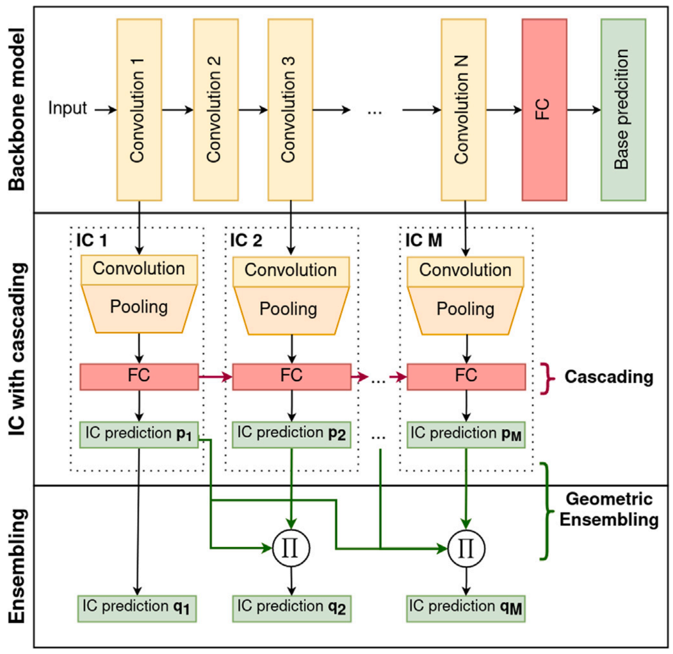


Fig. 2. Detailed scheme of the proposed ZTW model architecture. Backbone network f_θ lends its hidden layer activations to ICs, which share inferred information using cascade connections (red horizontal arrows in the middle row) and give predictions p_m. The inferred predictions are combined using ensembling (bottom row) giving q_m.

3.2. Cascade connections

Inspired by the gradient boosting algorithm and the literature on cascading classifiers (Viola & Jones, 2004), we allow each IC to improve on the predictions of previous ICs instead of inferring them from scratch. The idea of cascade connections is implemented by adding skip connections, resulting in IC_m combining the feature representation from layer f_{θ_m} of the base model with logits from IC_{m-1}:

$$l_m(x) = \begin{cases} g_{\phi_m}(f_{\theta_m}(x), sg(l_{m-1}(x))) & \text{if } m > 1 \\ g_{\phi_1}(f_{\theta_1}(x)) & \text{if } m = 1 \end{cases} \tag{1}$$

where $l_m(x)$ are logits returned by the mth IC, and sg is the stop gradient operation. The probability distribution produced by IC_m is then:

$$p_m(x) = \text{softmax}(l_m(x)) \tag{2}$$

IC_1 uses only the information coming from the layer f_{θ_1}, which does not need to be the first hidden layer of f_θ. Fig. 2 shows the skip connections as red horizontal arrows.

Each IC_m is trained in parallel (with respect to ϕ_m) to optimize the prediction of all output classes using an appropriate loss function $\mathcal{L}(p_m)$, e.g. cross-entropy for classification. However, during the backward step it is crucial to stop the gradient of a loss function from passing to the previous IC. Allowing the gradients of loss $\mathcal{L}(p_m)$ to affect ϕ_j for $j \in 1, .., m-1$ leads to a significant performance degradation of earlier layers due to increased focus on the features important for IC_m, as we show in Section 5.4.3.

3.3. Ensembling

Ensembling in machine learning models reliably increases the performance of a model while improving robustness and uncertainty estimation (Fort et al., 2019; Lakshminarayanan et al., 2017). The main drawback of this approach is its wastefulness, as it requires to train multiple models and use them to process the same examples. However, in our setup we can adopt this idea to combine predictions which were already pre-computed in previous ICs, with near-zero additional computational cost.

To obtain a reliable zero-waste system, we build ensembles that combine outputs from groups of ICs to provide the final answer of the mth classifier. We consider two variants of ensemble models: additive and multiplicative. While the additive ensemble uses the arithmetic mean to obtain final prediction, the multiplicative one applies the geometric average.

Standard additive ensemble relies on applying arithmetic mean to the individual classifiers. Since in our case we aggregate information from subsequent ICs, we introduce additional weights to modify their importance in producing the response at mth head. Let $p_1, p_2, \dots, p_m$ be the outputs of m consecutive IC predictions (after cascade connections stage) for a given x (Fig. 1). The probability of the ith class in the mth additive ensemble is defined by:

$$r_m^i(x) = \frac{1}{Z_m}\left(b_m^i + \sum_{j \le m} w_m^j p_j^i(x)\right), \tag{3}$$

where $b_m^i > 0$ and $w_m^j > 0$, for $j = 1, \dots, m$, are trainable parameters, and Z_m is a normalization factor, such that $\sum_i r_m^i(x) = 1$. Observe that w_m^j can be interpreted as our prior belief in predictions of IC_j, i.e. large weight w_m^j indicates higher confidence in the predictions of IC_j. On the other hand, b_m^i represents the prior of ith class for IC_m. The m indices in w_m and b_m are needed as the weights are trained independently for each subset $\{\text{IC}_j : j \le m\}$.

Since the classifiers defined by subsequent ICs vary significantly in predictive strength (later ICs achieve better performance than early ICs) and their predictions are correlated, additive ensemble may not work well in our case. Thus, we introduce weighted geometric mean with class balancing, which allows us to reliably find a combination of pre-computed responses that maximizes the expected result. The probability of the ith class in the mth multiplicative ensemble is defined by:

$$q_m^i(x) = \frac{1}{Z_m} b_m^i \prod_{j \le m} \left(p_j^i(x)\right)^{w_m^j}, \tag{4}$$

where Z_m is again a normalization factor and $w_m^j \in \mathbb{R}, b_m^i \in \mathbb{R}$ are trainable parameters. Direct calculation of the product in (4) might lead to numerical instabilities whenever the probabilities are close to zero. To avoid this problem we note that

$$b_m^i \prod_{j \le m} \left(p_j^i(x)\right)^{w_m^j} = b_m^i \exp\left(\sum_{j \le m} w_m^j \ln p_j^i(x)\right),$$

and that log-probabilities $\ln p_j^i$ can be obtained by running the numerically stable log softmax function on the logits g_{ϕ_m} of the classifier.

Straightforward comparison between two ensemble variants suggests that a low class confidence of a single IC would reduce more the probability of that class in the multiplicative ensemble than in the additive one. In consequence, in order for the confidence of the given class to be high, we require all ICs to be confident in that class. Thus, in geometric ensembling, an incorrect although confident IC answer has less chance of ending calculations prematurely. In the additive setting, the negative impact of a single confident but incorrect IC is much higher, as we show in Section 5.4.2.

Empirically, ensembling improves primarily the performance of later ICs. This is not surprising, given that the power of the ensemble increases with the number of members, provided they are at least weak in the sense of boosting theory (Schapire, 1990). As such, the two techniques introduced above are complementary, which we also show empirically via ablation studies in Section 5.4. The training procedure for the final model is presented in Algorithm 1.

Algorithm 1 Zero Time Waste training

```
Input: pre-trained backbone model $f_\theta$, cross-entropy loss function $\mathcal{L}$, training set $\mathcal{T}$.
Initialize $M$ shallow models $g_{\phi_m}$ at selected layers $f_{\theta_m}$.
For $m = 1, \dots, M$ do in parallel      ▷ ICs with cascading
    Set $p_m$ according to (2).
    minimize $\mathbb{E}_{(x,y)\in\mathcal{T}}\left[\mathcal{L}(p_m(x), y)\right]$ wrt. $\phi_m$ by gradient descent
For $m = 1, \dots, M$ do      ▷ Ensembling
    Initialize $w_m, b_m$ and define $q_m(x)$ according to (4).
    minimize $\mathbb{E}_{(x,y)\in\mathcal{T}}\left[\mathcal{L}(q_m(x), y)\right]$ wrt. $w_m, b_m$ by gradient descent
```

3.4. Conditional inference

Once a ZTW model is trained, the following question appears: how to use the constructed system at test time? More precisely, we need to dynamically find the shortest processing path with a correct answer for a given input example. For this purpose, we use one of the standard confidence scores given by the probability of the most confident class. If the mth classifier is confident enough about its prediction, i.e. if

$$\max_i q_m^i > \tau, \text{ for a fixed } \tau > 0, \tag{5}$$

where i is the class index, then we terminate the computation and return the response given by this IC. If this condition is not satisfied, we continue processing x and go to the next IC.

Threshold τ in (5) is a manually selected value, which controls the acceleration-performance trade-off of the model. A lower threshold leads to a significant speed-up at the cost of a possible drop in accuracy. Observe that for $\tau > 1$, we recover the original model f_θ, since none of the ICs can be confident enough to answer earlier. In practice, to select its appropriate value, we advise using a held-out set to evaluate a range of possible values of τ.

4. Experiments

In this section we examine the performance of Zero Time Waste and analyze its impact on waste reduction. We do this mainly by comparing it to four well-established or recent early-exit methods: (1) Shallow-Deep Networks (SDN) (Kaya et al., 2019), (2) Patience-Based Early Exit (PBEE) (Zhou et al., 2020), (3) Global-Past-Future (GPF) (Liao et al., 2021), and (4) Learning To Weigh (L2W) (Han et al., 2022).

In our experiments, we measure how much computation we can save by re-using responses of ICs while keeping good performance, hence obeying the zero waste paradigm. To evaluate the efficiency of the model, we compute the average number of floating-point operations (FLOPs) required to perform the forward pass for a single sample.[2] We use it as a hardware-agnostic measure of inference cost and refer to it simply as the "inference time" in subsequent references. Behavior of each method is evaluated at a particular fraction of the computational cost of the base network. We select the highest threshold τ such that the average inference time for entire test set is smaller than, for example, 25% of the original time, and then calculate accuracy for that threshold. The overall performance of each method can be represented by the inference time vs. accuracy trade-off that it provides.

We examine how ZTW performs at reducing waste in both Computer Vision (CV) and Natural Language Processing (NLP) data. Moreover, to the best of our knowledge, we are the first to apply early exit methods to reinforcement learning. Finally, we explore how ZTW allows the application of early exit methods in transfer learning settings. We provide the source code for our experiments at: https://github.com/gmum/Zero-Time-Waste.

4.1. Computer vision

As this paper focuses on pre-trained models, for evaluation on computer vision data we pick the most common ILSVRC 2012 ImageNet dataset (Deng et al., 2009), which contains 1,281,167 training and 50,000 validation images of high resolution. We test each method on four recent and state-of-the-art architectures, two of which are convolutional-based: EfficientNetV2-S (Tan & Le, 2021) and ConvNeXt-T (Liu et al., 2022); and two of which are Transformer-based: ViT-B (Dosovitskiy et al., 2020) and SwinV2-S (Liu et al., 2022).

The training of each model starts by loading the pre-trained backbone model, freezing its weights, and attaching early-exit heads to predetermined locations along the depth of the network. The head architecture and placement is the same for every method, but varies with the backbone architecture. We use a relatively dense placement of small heads and attach a linear head after every block in Transformer-based architectures, and a Conv2D-Pooling-Linear head after every residual block in convolutional-based architectures (we analyze different head architecture and placement choices in Sections 5.2 and 5.3, respectively). We adopt GPF for convolutional networks by manually picking the head output dimensionality shared between all heads. The Adam optimizer (Kingma & Ba, 2014) is used for every method except L2W, for which we employ SGD with momentum similarly as the original authors did.[3] Each model is trained for 15 epochs. All other training hyperparameters and additional experiments for other architectures and datasets are available in the Appendix.

Table 1 contains the results of the experiment averaged over three seeds. ZTW achieves superior results in 9 out of 16 cases, with the advantage over the next best method reaching as high as 5.03 percentage points, and the highest difference to the leading method being only 0.30 percentage points. While not originally designed for vision, GPF is the only other method showing consistent improvements over SDN, which can be seen as the simplest early-exit approach. Although PBEE reuses information from previous layers to decide whether to stop computation or not, this is not sufficient to reduce the waste in the network. While PBEE performs well on higher inference time limits, it often fails for smaller limits $(25\%, 50\%)$. We hypothesize that this is result of the fact that PBEE has smaller flexibility with respect to τ. While for SDN and ZTW values of τ are continuous, for PBEE they represent a discrete number of ICs that must sequentially agree before returning an answer.

4.2. Natural language processing

In this section we evaluate the performance of ZTW architecture in a natural language processing (NLP) scenario. Since models based on the Transformer architecture (Vaswani et al., 2017) are an essential foundation for current state-of-the-art solutions in NLP, we take the pre-trained language model BERT-base (Devlin, Chang, Lee, & Toutanova, 2019) as our backbone network. We then fine-tune it on a given task, freeze its weights and then add a single-layer IC after each Transformer layer. Each internal classifier added to the network simply discards every token except the CLS token, and then uses a fully-connected layer to make a classification.

In Table 2, we compare the performance of different early exit methods for various classification tasks available in the GLUE benchmark (Wang et al., 2018). In every case, we report validation set accuracy averaged over three models trained with different seeds. For each task, we train each method for 5 epochs. The data for the table is generated in the same way as for CV experiments from Section 4.1. All other training hyperparameters and additional plots are available in the Appendix.

[2] Note that the cost of a head, if that head is executed for the current example, is always counted, even if the output of that head was not used for the final prediction.

[3] We implemented Adam for L2W, but due to the additional memory requirements we were not able to run L2W with Adam on Nvidia A100 GPUs.

Table 1
Evaluation results on the ILSVRC 2012 ImageNet dataset. Test accuracy (in percentages) obtained using the time budget: 25%, 50%, 75%, 100% of the base network and without any limits ("Max"). The first column shows the test accuracy of the base network. We bold multiple results when there is no meaningful statistical difference between them. ZTW either offers superior performance or remains competitive to other methods in the majority of cases.

Architecture	Method	25%	50%	75%	100%	Max
EfficientNetV2-S (84.23)	SDN	25.47 ± 0.11	45.94 ± 0.15	75.88 ± 0.04	84.12 ± 0.01	84.24 ± 0.00
	PBEE	20.38 ± 0.09	34.99 ± 0.17	70.00 ± 0.19	84.14 ± 0.01	84.24 ± 0.00
	GPF	28.26 ± 0.02	50.50 ± 0.31	**78.04 ± 0.15**	84.02 ± 0.03	84.24 ± 0.00
	L2W	25.87 ± 0.19	45.78 ± 0.26	76.35 ± 0.11	**84.17 ± 0.01**	84.23 ± 0.00
	ZTW	**28.73 ± 0.08**	**51.59 ± 0.01**	77.83 ± 0.05	84.09 ± 0.02	84.24 ± 0.00
ConvNeXt-T (82.52)	SDN	39.15 ± 0.13	62.44 ± 0.08	78.42 ± 0.05	**82.48 ± 0.01**	82.52 ± 0.00
	PBEE	34.37 ± 0.32	53.09 ± 0.23	72.32 ± 0.25	81.75 ± 0.03	82.52 ± 0.00
	GPF	38.11 ± 0.18	60.50 ± 0.20	76.70 ± 0.25	82.37 ± 0.02	82.52 ± 0.00
	L2W	32.46 ± 0.38	55.32 ± 0.02	74.61 ± 0.11	82.31 ± 0.01	82.52 ± 0.00
	ZTW	**44.19 ± 0.07**	**65.96 ± 0.16**	**79.40 ± 0.04**	82.37 ± 0.00	82.52 ± 0.00
ViT-B (81.07)	SDN	18.85 ± 0.05	55.78 ± 0.08	78.42 ± 0.08	80.45 ± 0.03	81.07 ± 0.00
	PBEE	11.65 ± 0.06	40.26 ± 0.08	65.60 ± 0.08	**81.06 ± 0.00**	81.07 ± 0.00
	GPF	**20.40 ± 0.05**	55.70 ± 0.24	78.76 ± 0.13	80.90 ± 0.03	81.07 ± 0.00
	L2W	16.46 ± 0.21	53.74 ± 0.29	78.74 ± 0.12	80.59 ± 0.07	81.07 ± 0.00
	ZTW	20.27 ± 0.02	**56.38 ± 0.08**	**79.20 ± 0.04**	80.92 ± 0.01	81.07 ± 0.00
SwinV2-S (83.71)	SDN	40.95 ± 0.07	70.64 ± 0.04	81.13 ± 0.04	81.13 ± 0.04	83.71 ± 0.00
	PBEE	25.84 ± 0.04	59.29 ± 0.13	80.53 ± 0.00	**83.71 ± 0.00**	83.71 ± 0.00
	GPF	**42.05 ± 0.36**	69.50 ± 0.22	82.55 ± 0.20	83.44 ± 0.08	83.71 ± 0.00
	L2W	26.14 ± 0.30	63.51 ± 0.10	83.09 ± 0.03	83.70 ± 0.00	83.71 ± 0.00
	ZTW	**42.06 ± 0.05**	**72.42 ± 0.10**	**83.12 ± 0.01**	83.41 ± 0.02	83.71 ± 0.00

Table 2
Training different early exit architectures with pre-trained BERT backbone on the common NLP datasets: MRPC, RTE, SST2, QNLI and QQP. ZTW achieves competitive results for every considered NLP task.

Dataset	Method	25%	50%	75%	100%	Max
MRPC (85.95 ± 1.74)	SDN	**77.53 ± 1.58**	**82.60 ± 0.25**	**85.62 ± 1.35**	**85.87 ± 1.63**	85.95 ± 1.74
	PBEE	70.18 ± 0.28	77.61 ± 0.14	83.01 ± 0.99	**86.27 ± 1.36**	85.95 ± 1.74
	GPF	**76.47 ± 1.53**	81.86 ± 0.88	**85.62 ± 1.35**	**85.87 ± 1.39**	85.95 ± 1.74
	L2W	**76.63 ± 0.62**	81.62 ± 1.30	**85.13 ± 1.44**	**85.13 ± 1.44**	85.95 ± 1.74
	ZTW	**76.72 ± 0.85**	**82.76 ± 0.37**	**85.95 ± 1.23**	**85.95 ± 1.23**	85.95 ± 1.74
RTE (68.95 ± 1.88)	SDN	60.53 ± 0.21	**64.86 ± 2.73**	**67.87 ± 1.30**	**69.68 ± 1.57**	68.95 ± 1.88
	PBEE	57.16 ± 1.27	**65.46 ± 1.71**	67.27 ± 2.54	**69.31 ± 1.57**	68.95 ± 1.88
	GPF	58.48 ± 0.72	**64.74 ± 1.27**	**67.87 ± 0.96**	**69.31 ± 1.30**	68.95 ± 1.88
	L2W	59.33 ± 0.91	**65.22 ± 1.50**	**67.51 ± 0.96**	**69.68 ± 1.57**	68.95 ± 1.88
	ZTW	**61.25 ± 0.55**	63.30 ± 1.67	**68.59 ± 1.25**	**69.07 ± 1.99**	68.95 ± 1.88
SST2 (92.58 ± 0.18)	SDN	**83.22 ± 0.37**	**91.06 ± 0.23**	**92.66 ± 0.30**	**92.66 ± 0.30**	92.58 ± 0.18
	PBEE	79.05 ± 0.40	83.22 ± 0.24	89.98 ± 0.37	**92.62 ± 0.24**	92.58 ± 0.18
	GPF	**83.14 ± 0.30**	**91.21 ± 0.35**	**92.58 ± 0.48**	**92.58 ± 0.48**	92.58 ± 0.18
	L2W	81.84 ± 0.46	90.63 ± 0.33	**92.70 ± 0.26**	**92.70 ± 0.26**	92.58 ± 0.18
	ZTW	**83.33 ± 0.48**	**91.21 ± 0.48**	**92.62 ± 0.33**	**92.62 ± 0.33**	92.58 ± 0.18
QNLI (91.43 ± 0.27)	SDN	**76.36 ± 0.33**	87.63 ± 0.17	**90.98 ± 0.02**	**91.43 ± 0.24**	91.43 ± 0.27
	PBEE	65.21 ± 0.37	83.91 ± 0.41	88.81 ± 0.42	91.17 ± 0.22	91.43 ± 0.27
	GPF	**76.94 ± 0.63**	**88.32 ± 0.12**	90.95 ± 0.17	**91.41 ± 0.29**	91.43 ± 0.27
	L2W	76.22 ± 0.15	87.39 ± 0.36	90.87 ± 0.05	**91.42 ± 0.32**	91.43 ± 0.27
	ZTW	**76.62 ± 0.23**	87.94 ± 0.07	90.90 ± 0.14	**91.44 ± 0.25**	91.43 ± 0.27
QQP (91.13 ± 0.08)	SDN	83.75 ± 0.05	90.06 ± 0.14	90.94 ± 0.08	90.94 ± 0.08	91.13 ± 0.08
	PBEE	76.15 ± 0.10	86.34 ± 0.26	90.57 ± 0.19	**91.14 ± 0.07**	91.13 ± 0.08
	GPF	**86.27 ± 0.10**	**90.73 ± 0.11**	**91.03 ± 0.07**	91.03 ± 0.07	91.13 ± 0.08
	L2W	82.05 ± 0.19	90.11 ± 0.11	**91.08 ± 0.07**	**91.08 ± 0.07**	91.13 ± 0.08
	ZTW	83.56 ± 0.06	89.93 ± 0.05	**91.04 ± 0.05**	91.04 ± 0.05	91.13 ± 0.08

On this popular benchmark the SDN, GPF, L2W and ZTW perform similarly, while PBEE consistently displays poor performance on low computational budgets. We hypothesize that the lack of advantage of ZTW over SDN is due to the low number of classes in GLUE, and a relatively small number of heads attached to the BERT-Base backbone. The number of parameters added by cascading and ensembling, the components of ZTW, depends only on these two factors. To verify this hypothesis we repeat our training procedure on the 20 Newsgroups dataset (Lang, 1995), which is a classification dataset with 20 classes. Furthermore, instead of placing an IC after every block, we place it also after the multi-head attention subblock, resulting in twice as many heads as before. The results, presented in Fig. 3, show a significant improvement of ZTW over SDN on this setup. We provide more evidence for the effect of number of classes on performance of ZTW in the Appendix.

4.3. Reinforcement learning

Although supervised learning is an important testbed for deep learning, it does not properly mirror the challenges encountered in the real world. In order to examine the impact of waste-minimization methods in a setting that reflects the sequential nature of interacting with the world, we evaluate it in a Reinforcement Learning (RL) setting. In particular, we use the environments from the suite of Atari 2600 games (Mnih et al., 2015).

Similarly as in the supervised setting, we start with a pre-trained network, which in this case represents a policy trained with the Proximal Policy Optimization (PPO) algorithm (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). We attach the ICs to the network and train it by distilling the knowledge from the core network to the ICs. We

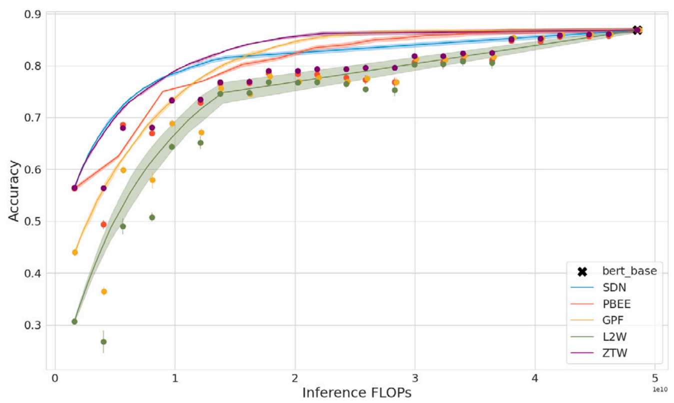


Fig. 3. Accuracy vs. computational cost trade-off obtained by five early-exit methods on the 20 Newsgroups dataset, with BERT-Base as backbone model. Each IC is marked with a point, and the score of threshold-based early-exiting is plotted for a range of τ as a line. Observe that a clear advantage of ZTW over SDN appears after passing a certain number of ICs.

use a behavioral cloning approach, where the states are sampled from the policy defined by the ICs and the labels are provided by the expert model. Since actions in Atari 2600 are discrete, we can then use the same confidence threshold-based approach to early exit inference as in the case of classification.

In order to investigate the relationship between computation waste reduction and performance, we evaluate Zero Time Waste for different values of confidence threshold τ. By setting a very high τ value, we retrieve the performance of the original model (none of the ICs respond) and by slowly decreasing its value we can reduce the computational cost (ICs begin to return answers earlier). In Fig. 4 we check values of τ in the interval $[0.1, 1.0]$ to show how ZTW is able to control the acceleration-performance balance for Q*Bert and Pong, two popular Atari 2600 environments. By setting lower τ thresholds for Q*Bert we can save around 45% of computations without score degradation. Similarly, for Pong we can get 60% reduction with minor impact on performance (note that average human score is 9.3 points). This shows that even the small four-layered convolutional architecture commonly used for Atari (Mnih et al., 2015) introduces a noticeable waste of computation which can be mitigated within a zero-waste paradigm. We highlight this fact as the field of reinforcement learning has largely focused on efficiency in terms of number of samples and training time, while paying less attention to the issue of efficient inference.

4.4. Transfer learning

Finally, we investigate the problem of training the Internal Classifiers in a transfer learning fashion. We consider a *source* dataset A, on which the base model is pre-trained, and *target* dataset B, on which we train the ICs. Our aim is to verify whether the features learnt on dataset A by the given stage of the base model will be useful for the IC trained on dataset B, and how ZTW impacts that feature transfer.

Two mutually opposing factors might influence the performance of a transfer-learned IC:

1. **Descriptiveness of the learned features:** Previous experiments indicate that the ICs attached to early layers of the network achieve worse accuracy than the deeper-placed ICs. We hypothesize this will hold in the transfer learning setting, as the features learned by shallow-placed ICs are not descriptive enough.
2. **Overfitting to the original dataset:** Prior work on transfer learning indicates that deeper network layers learn less universal features than the shallow ones, and thus recommends against freezing their parameters during fine-tuning to new tasks (Yosinski, Clune, Bengio, & Lipson, 2014). When training on dataset B the deeper-placed ICs may fail to achieve good performance, since the deeper base network layers will be overfitted to extract features specific to A and less relevant for B. Thus, the assumption than the quality of early-exit predictions will increase with larger computational budget may no longer hold.

To verify the suitability of early-exit methods for transfer learning, we pre-train ResNet-56 backbones (He et al., 2016) on CIFAR-10 and CIFAR-100 datasets. We use them as bases for training of ICs of SDN, PBEE and ZTW models on different datasets. We report the accuracy vs. inference time plots for four transfer combinations in Fig. 5.

The accuracy of standard ICs increases along with placement depth until a certain point and then decreases after that. This leads to accuracy decrease of the SDN and PBEE models when increasing the time budget. Thus, when transfer learning an SDN early exit model, one would need to devise a non-trivial early-exit heuristic that takes the decreasing accuracy into account. On the other hand, the ZTW early-exit model suffers from no such drawbacks, and due to cascading connections and ensembling of ICs its accuracy consistently increases with the growing time budget in all four cases.

5. Analysis

In this section we analyze how Zero Time Waste prevents information loss that occurs in previous models. We also present results that explain our design decisions and demonstrate the limitations of early-exiting methods by presenting its failure cases.

5.1. Information loss in early exit models

Since ICs in a given model are heavily correlated, it is not immediately obvious why reusing past predictions should improve performance. Later ICs operate on high-level features for which class separation is much easier than for early ICs, and hence get better accuracy. Thus, we ask a question — is there something that early ICs know that the later ICs do not?

For that purpose, we introduce a metric to evaluate how much a given IC could improve performance by reusing information from all previous ICs. We measure it by checking how many examples incorrectly classified by IC_m were classified correctly by any of the previous ICs. An IC which reuses predictions from the past perfectly would achieve a low score on this metric since it would remember all the correct answers of the previous ICs. On the other hand, an IC in a model which trains each classifier independently would have a higher score on this metric, since it does not use past information at all. We call this metric Hindsight Improvability (HI) since it measures how many mistakes we would be able to avoid if we used information from the past efficiently.

Let C_m denote the set of examples correctly classified by IC_m, with its complement $\overline{C}_m$ being the set of examples classified incorrectly. To measure the Hindsight Improvability of IC_m we calculate:

$$\mathrm{HI}_m = \frac{\left|\overline{C}_m \cap \left(\bigcup_{n<m} C_n\right)\right|}{\left|\overline{C}_m\right|}$$

Fig. 6 compares the values of HI for SDN – a method with independent ICs – and ZTW which explicitly recycles computations. In the case of VGG16 trained with independent ICs, over 60% of the mistakes could be avoided if we properly used information from the past, which would translate to improvement from 71.5% to 82.9% accuracy. Similarly, for ResNet-56 trained on TinyImageNet, the number of errors could be cut by around 57%.

ZTW consistently outperforms the baseline, with the largest differences visible at the later ICs, which can in principle gain the most from reusing previous predictions. Thus, Zero Time Waste is able to efficiently recycle information from the past. At the same time, there is still a room for significant improvements, which shows that future zero waste approaches could offer additional enhancements.

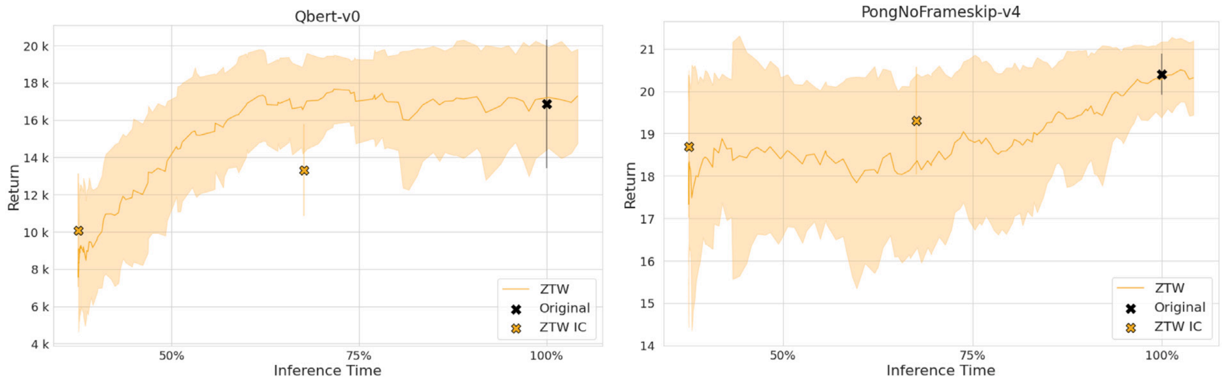


Fig. 4. Inference time vs. average return of the ZTW policy in an RL setting on Q*bert and Pong Atari 2600 environments. The plot was generated by using different values of the confidence threshold τ hyperparameter. Since the RL environments are stochastic, we plot the return with a standard deviation calculated on 10 runs. ZTW saves a significant amount of computation while preserving the original performance, showcasing that waste can be minimized also in the reinforcement learning domain.

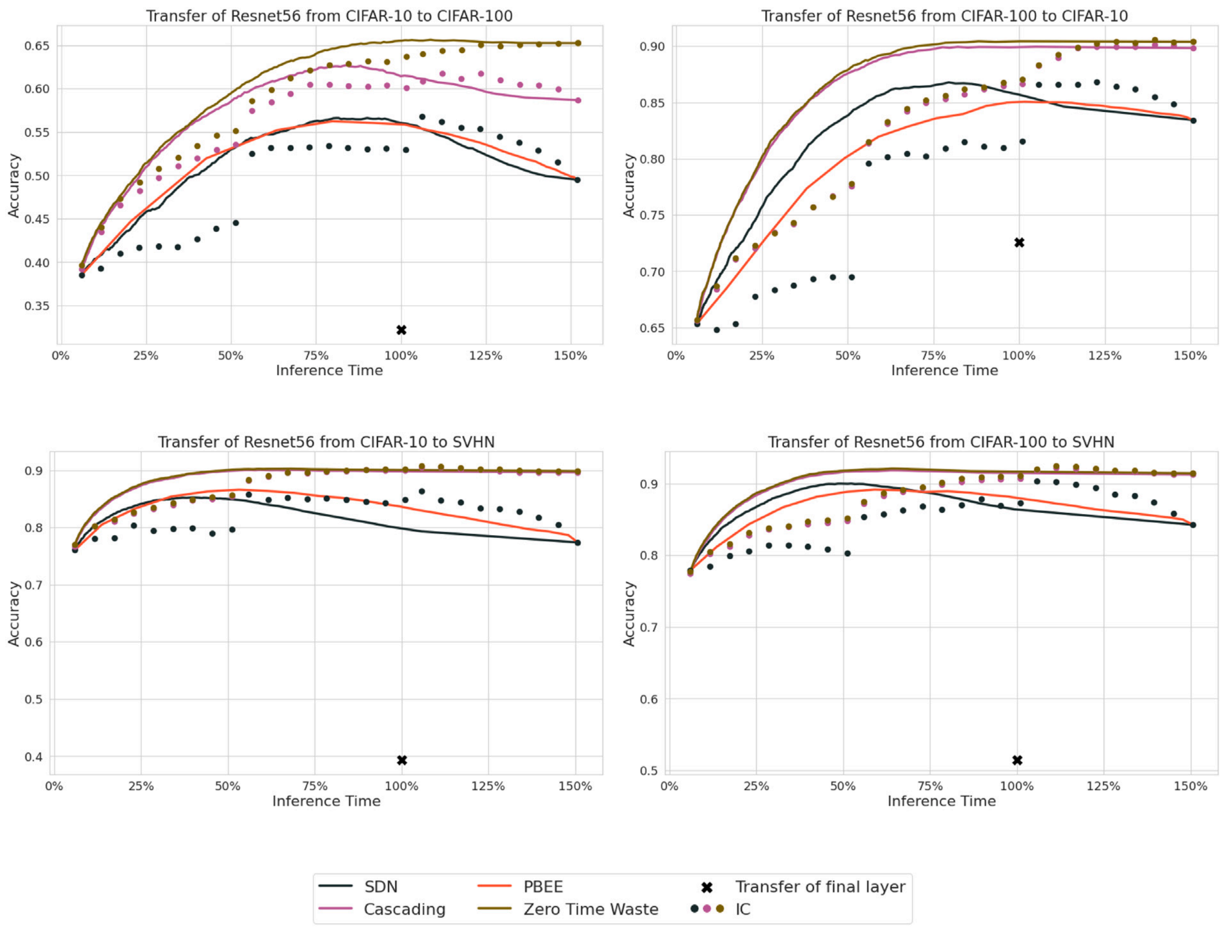


Fig. 5. Inference time vs accuracy for different transfer learning settings. In each case, the base network architecture is ResNet-56 (He et al., 2016). The dots in the plots indicate the accuracy of each consecutive IC attached to increasingly deeper parts of the base network, whereas the lines indicate the mean accuracy achieved by the set of ICs, given the inference time budget. For reference, in each plot we also mark the performance of the classifier transfer-learned by freezing the entire pre-trained network and training only the final classification layer.

5.2. Choice of head architecture

A single head l_m can be any network with the given output dimensionality and input dimensionality dictated by the dimensionality of f_{θ_m}. A natural question arises about how to pick the architecture of the heads. We assume the architecture of each head in a single model stays the same with the exception of input dimensionality, which can differ in some networks, for example in convolutional neural networks. To explore how the size of head architecture affects performance, we train multiple models on the EfficientNetV2-S backbone differing only in the size of the heads. Due to the increased size the heads are trained for thrice as long and without weight decay. We tune the hyperparameters of the convolutional layers in the heads so that the total computational cost of the ICs approaches 25% of the cost of the original model. The last layer of the head is always a fully-connected layer, and always follows a pooling layer. The IC with depth 2 is the same head architecture as was used in Section 4.1.

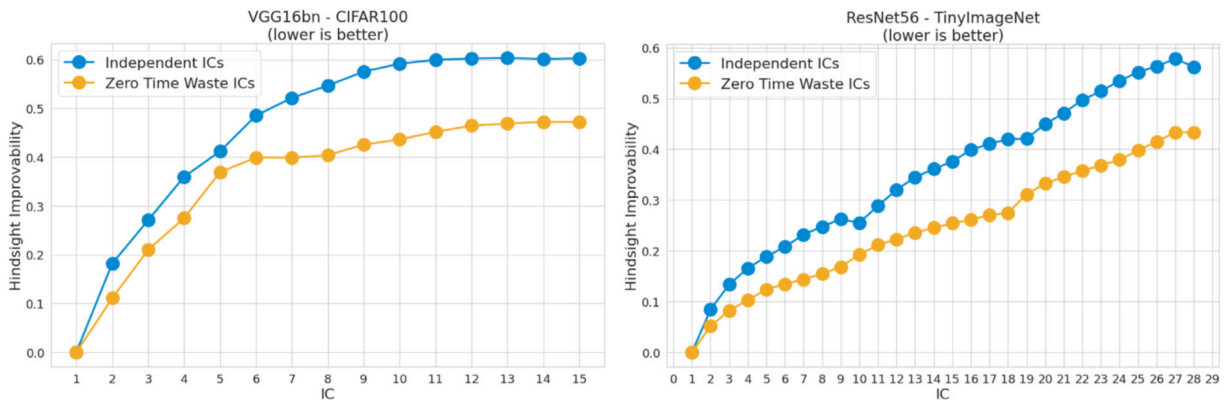


Fig. 6. Hindsight Improvability. For each IC (horizontal axis) we look at examples it misclassified and we check how many of them were classified correctly by any of the previous ICs. The lower the number, the better the IC is at reusing previous information.

Table 3
We examine different architectures of the head as the only hyperparameter that changes between runs. The head architecture depth and the total computational cost of heads, reported as a percentage of the cost of the original backbone model, are listed for each variant. Both very simple or excessively large head architectures yield suboptimal outcomes. ZTW displays consistent improvements on every variant considered in the experiment.

Method	IC depth	ICs cost ratio (%)	25%	50%	75%	100%	Max
SDN	1	0.7%	25.52	42.44	62.06	63.62	84.24
SDN	2	3.1%	26.54	44.90	70.31	79.04	84.24
SDN	3	7.8%	28.50	42.43	67.12	83.86	84.24
SDN	4	11.0%	31.78	45.58	68.27	83.72	84.24
SDN	5	17.2%	23.26	33.74	54.19	80.02	84.24
SDN	6	25.7%	28.78	38.51	55.74	76.03	84.24
ZTW	1	0.9%	28.36	47.72	74.00	81.23	84.24
ZTW	2	3.3%	29.33	50.00	**75.05**	83.48	84.24
ZTW	3	8.0%	30.47	47.74	71.77	**84.05**	84.24
ZTW	4	11.2%	**34.33**	**52.03**	73.79	**84.05**	84.24
ZTW	5	17.4%	27.06	41.69	63.62	82.13	84.24
ZTW	6	25.9%	31.51	44.73	62.70	79.14	84.24

Table 3 presents the results of this experiment along with the cost of each head variant being considered. Increasing the size of the head up to a certain point improves its final performance on low computational budgets. This effect diminishes lightly on higher computational budgets due to the accumulated cost of enlarged heads. Note that ZTW consistently demonstrates improvements over SDN across all examined head sizes, thus reinforcing the validity of the reasoning behind cascading and ensembling.

5.3. Choice of head placement density

Similarly to the choice of architecture for the heads, it is not obvious how densely to attach heads to the backbone model. Additional heads add a computational overhead that may outweigh any performance benefit that they provide. To discover the optimal approach, we train models with different head densities on the EfficientNetV2-S backbone. With 41 blocks after which a head can be placed, we consider variants that place heads after every-n blocks, and the head architecture used for this experiment is the same as for the experiments from Section 4.1.

The results are shown in Table 4. Not surprisingly, an exceedingly sparse placement is detrimental for the overall accuracy-cost trade-off. While denser placement gives slightly better results for SDN on lower and middle budgets, ZTW is able to significantly improve on this by utilizing the additional heads and reducing the waste that happens in SDN due to discardment of intermediate outputs.

Table 4
The effect of varying head placement density for early-exit model built on ImageNet pretrained EfficientNetV2-S backbone. ICs are placed after every n_{every} blocks for $n_{\text{every}} \in \{1, 2, 3, 4, 5, 10\}$. ZTW especially benefits from denser placement of heads when low computational budgets are considered, but an excessive number of ICs is detrimental for performance on higher budgets.

Method	n_{every}	25%	50%	75%	100%	Max
SDN	1	23.85	42.10	69.02	79.51	84.24
SDN	2	22.61	41.52	69.97	80.73	84.24
SDN	3	21.83	40.84	71.23	81.68	84.24
SDN	4	19.59	37.61	69.93	82.15	84.24
SDN	5	21.44	40.29	71.58	82.28	84.24
SDN	10	21.61	38.65	69.20	83.92	84.24
ZTW	1	**30.06**	**49.92**	74.50	83.70	84.24
ZTW	2	25.48	46.02	**74.86**	83.77	84.24
ZTW	3	22.68	45.01	74.77	83.85	84.24
ZTW	4	20.29	40.97	73.42	83.91	84.24
ZTW	5	22.03	42.58	74.35	83.93	84.24
ZTW	10	21.67	40.37	70.71	**84.04**	84.24

5.4. Ablation studies

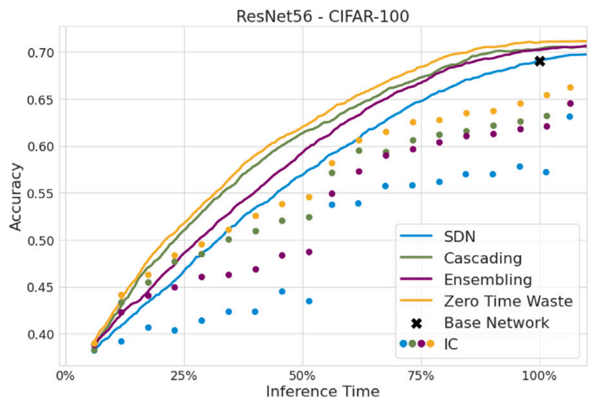
In this section we explore the following issues: (1) what is the individual impact of cascade connections and geometric ensembling, (2) how performance of additive and geometric ensembles compares in our setting, and (3) how stopping the gradient in *cascade connections* impacts learning dynamics.

5.4.1. Impact of cascading and ensembling

An important question is whether we need both *cascade connections* and ensembling in the proposed model, and what role do they play in the final performance of our model. Fig. 7 shows the results of independently applied *cascade connections* and geometric ensembling on a ResNet-56 and VGG-16 trained on CIFAR-100. We observe that depending on the threshold τ and the architecture, one of these techniques may be more important than the other. However, combining these methods consistently improves the performance that each of them achieves independently. Thus we argue that both components are required in Zero Time Waste and using only one of them will lead to significant performance deterioration.

5.4.2. Geometric vs. additive ensembles

We analyze how geometric ensembles perform in comparison to additive ensembles. The comparison between the two is presented in Fig. 8. The results show that the geometric ensemble consistently outperforms the additive ensemble, although we found out that magnitude of improvement varies across datasets and architectures. While the difference on CIFAR-10 is negligible, it becomes evident on Tiny ImageNet, especially with the later layers. The results suggest that



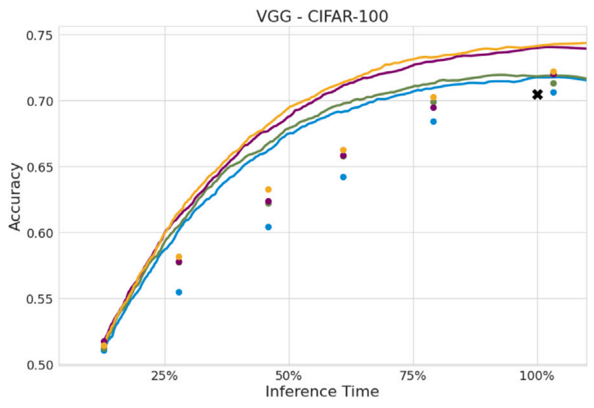


Fig. 7. Ablation studies exhibiting the importance of both techniques proposed in the paper. Although both *cascade connections* and geometric ensembling seem to help, the exact effect depends on the architecture and chosen threshold τ. For ResNet56 *cascade connections* seem to be much more helpful than ensembling, while for VGG16 the opposite is true. As such, both are required to consistently improve results.

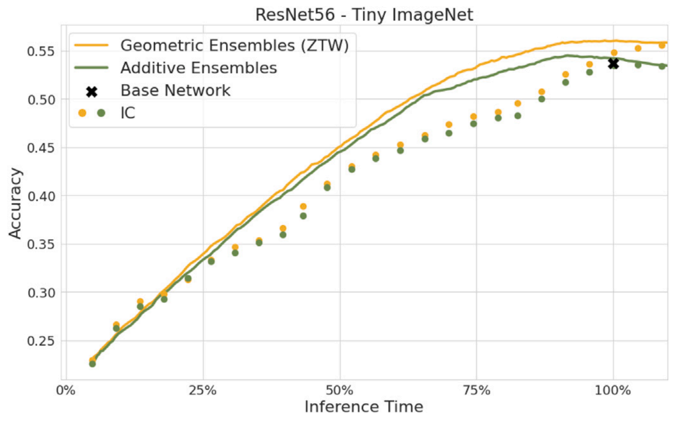


Fig. 8. Comparison of geometric and additive ensembling on ResNet-56 (ICs with *cascade connections*), conducted on TinyImageNet. Geometric ensembling is better in reusing predictions from previous ICs and thus helps in achieving better results.

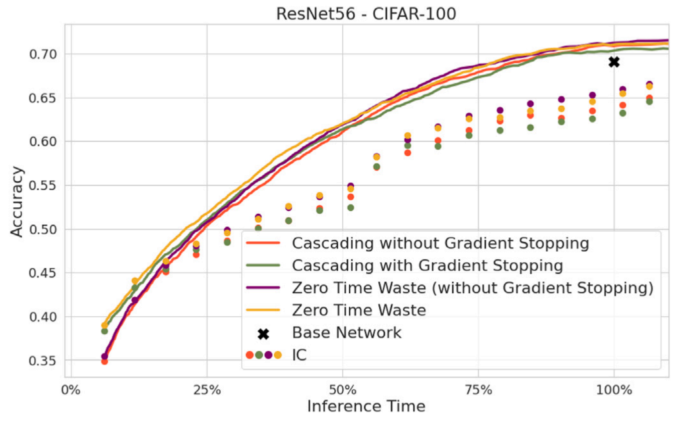


Fig. 9. Effects of stopping gradient in ResNet-56 trained on CIFAR-100.

geometric ensembling is more helpful on more complex datasets with a larger number of classes.

5.4.3. Stop gradients in cascade connections

As mentioned in Section 3 of the main paper, we decide to stop gradient from flowing through the *cascade connections*. We motivate this decision by noticing that the gradients of later layers might destroy the predictive power of the earlier layers. In order to test this hypothesis empirically, we run our experiments on ResNet-56 with and without gradient stopping. As shown in Fig. B.3, the accuracy of the early ICs is lower when not using gradient stopping. Performance of later ICs may vary, as not using stopping gradient allows greater expressivity for later ICs. Since the second component of our method, ensembling, is able to reuse information from the early ICs we find it beneficial to use gradient stopping in the final model.

We provide a more in-depth observation of the reason why the gradient of later ICs might have a detrimental effect on the performance of early ICs. Observe that in the setting without the detach the parameters of the first IC will be updated using $\sum_k g_k$, where g_k is the gradient of the loss of the kth IC wrt. parameters of the first IC. Experimental investigation showed that the cosine similarity of $\sum_k g_k$ and g_1 is approximately 0.5 at the beginning of the training, which means that these gradients point in different directions. Since the gradient g_1 represents the best direction for improving the first IC, using $\sum_k g_k$ will lead to a non-optimal update of its weights, thus reducing its predictive performance. With detach, $g_2 = g_3 = \cdots = 0$ and as such the cosine similarity is always 1. This reasoning can be extended to other ICs.

5.5. Internal classifier knowledge distillation

The efficiency of the whole model depends on the performance of individual ICs. Knowledge distillation applied to ICs is known to improve performance when the whole network is trained along with the ICs (Li et al., 2019; Phuong & Lampert, 2019; Wang & Li, 2021). However, it is not clear if ICs, which are relatively small modules compared to the whole backbone network, would benefit from knowledge distillation in the frozen-backbone setting. In this subsection we explore how SDN and ZTW combine with knowledge distillation of various forms. Assuming that we choose a set E containing indices that determine the ICs to distill from, the optimization objective for the mth IC can be now redefined as:

$$H(p_m, y) + \alpha \frac{1}{|E|} \sum_{i \in E} (H(p_m, sg(p_i))),$$

where H is the cross entropy and sg is the stop gradient operation. We consider three variants of the distillation training procedure: *next* — distill knowledge from the directly succeeding IC ($n + 1 \in E$), *last* — distill from the last head ($N \in E$), *later* — distill from every succeeding IC ($\forall_{n<i\leq N} i \in E$). Since we use cross entropy instead of Kullback–Leibler divergence for distillation, values of terms are of the same magnitude, and thus the hyperparameter α is set to 1 in all of our experiments.

Table 5 presents the results for the described variants. Distilling knowledge from the next IC brought no improvement over baseline. Using the last head for distillation resulted in a minor improvement, especially for lower and middle computational budgets. The variant with all deeper heads being used as target was beneficial for early ICs.

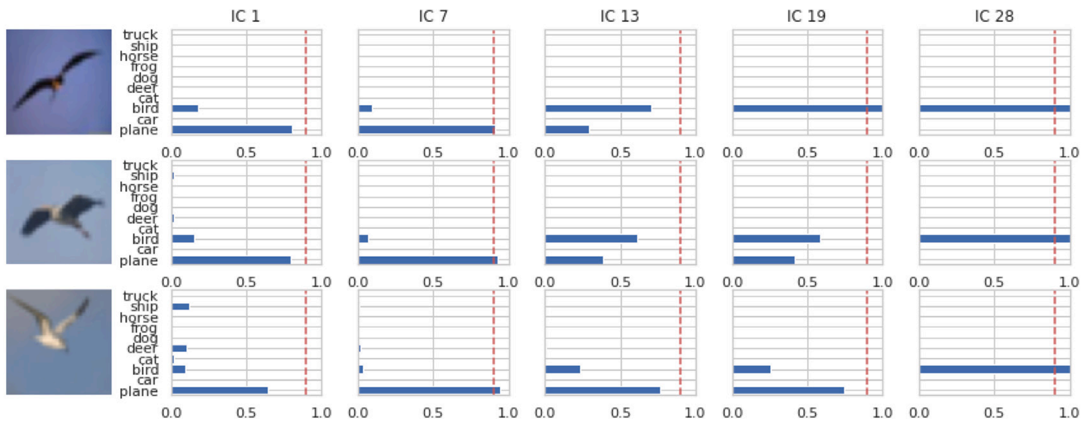


Fig. 10. Examples of bird images which were incorrectly classified as airplanes by ZTW. The early ICs are misled by the low-level features (blue sky, sharp edges, grayscale silhouette) and return a prediction before the later ICs can detect more subtle high-level features.

Table 5
Impact of the addition of distillation to the ICs training phase on the final performance of ZTW. Standard SDN and ZTW models are trained on the Tiny ImageNet dataset using the ResNet-50 base model. The same procedure is repeated for the distillation variants, but with an additional distillation loss term added in the ICs training phase. We can see that distillation improves performance on low computational budgets for both models by a similar amount, hinting that ZTW and distillation are complementary.

Method	25%	50%	75%	100%	Max
SDN	41.9	49.8	57.5	65.3	71.0
SDN - next	42.5	50.0	58.7	66.1	71.1
SDN - last	42.9	51.0	59.0	65.9	70.9
SDN - later	43.6	51.2	59.3	65.6	71.0
ZTW	42.3	50.9	59.2	66.9	71.1
ZTW - next	42.5	50.8	59.2	66.7	71.1
ZTW - last	42.9	52.0	**60.5**	67.1	71.1
ZTW - later	**44.0**	**52.2**	60.3	**67.4**	71.1

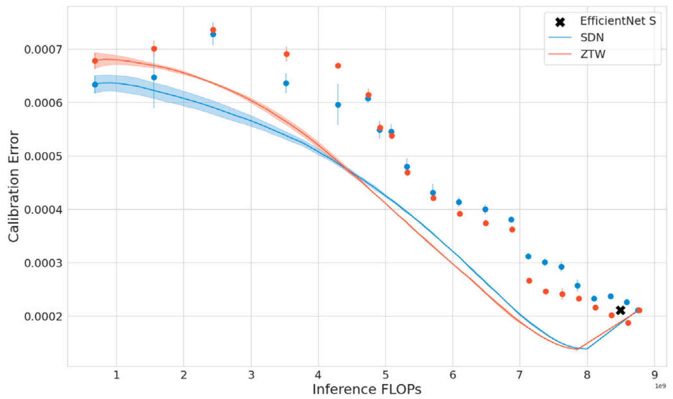


Fig. 11. KS calibration error plotted as a function of computational cost of the models trained on the ImageNet pre-trained EfficientNetV2-S backbone. Similarly as before, the heads are plotted as points, and inference by using the confidence threshold strategy is plotted for multiple thresholds as a line. We can see that early heads exhibit a considerably large calibration error as compared to the original backbone model.

We hypothesize that its superior performance is due to the significant difference in representational power between early and deep ICs.

Despite the knowledge distillation training making the answers of ICs more similar to each other, the ZTW model consistently outperforms SDN. Moreover, the gains from distillation are similar for both ZTW and SDN. These results indicate that the overthinking phenomenon (Kaya et al., 2019) in early exit models is independent from and cannot be straightforwardly fixed with knowledge distillation.

5.6. Limitations

Zero Time Waste significantly improves performance while adding only a negligible computational overhead. It may slightly complicate the implementation due to two separate phases of training. Despite the need for a separate set of hyperparameters, in our experience the optimal learning rate for the ensembling phase always remains the same and thus does not require tuning.

ZTW also inherits the limitations of other early exiting methods. We show this in Fig. 10, which contains examples of images for which low-level features in a given image consistently point at a wrong class, while high-level features would allow us to deduce the correct class. Images of birds which contain sharp lines and grayscale silhouettes are interpreted as airplanes by early ICs which operate on low-level features. If the confidence of these classifiers gets high enough, the answer might be returned before later classifiers can correct this decision.

To qualitatively confirm our findings, we measure the Kolmogorov–Smirnov (KS) calibration error (Gupta et al., 2020) of early exit models and present it in Fig. 11.

To lend qualitative validation to our observations, we assess the Kolmogorov–Smirnov (KS) calibration error of early exit models, as illustrated in Fig. 9. Evidently, In light of these insights, we emphasize the pertinence of addressing head calibration issues as a crucial avenue for future advancements in conditional computation methods. A promising direction for further exploration involves the integration of calibration techniques to meticulously calibrate the behaviors of individual heads

The calibration error is notably greater in the shallower heads compared to the deeper ones. This discrepancy diminishes progressively with increasing depth, eventually aligning with the calibration error exhibited by the original backbone model. In light of these insights, we emphasize the importance of addressing head calibration issues as a crucial avenue for future advancements in early exit methods. A promising direction for further research involves the integration of calibration techniques in order to calibrate the individual heads.

6. Conclusion

In this work, we show that discarding predictions of the previous ICs in early exit models leads to waste of computation resources and a significant loss of information, which we show by introducing the Hindsight Improvability metric. The proposed Zero Time Waste method attempts to solve these issues by incorporating outputs from the past heads by using cascade connections and ensembling of ICs. We show that ZTW outperforms other approaches on common computer vision, natural language processing, and reinforcement learning setups. ZTW is also ideally suited for transfer learning scenarios and can be combined

Table A.1
Hyperparameters used for training the early-exit models on the ImageNet dataset.

Architecture	Method	Batch size	γ	λ	Head placement	Head architecture
EfficientNetV2-S	SDN	64	0.0005	0.1	After blocks: 3, 4, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39	• 3 × 3 Conv (stride = 2, out_channels = half) • SDNPool (Kaya et al., 2019) • Linear
	PBEE	64	0.0005	0.1		
	GPF	64	0.0001	0.1		
	L2W	32	0.01	0.0001		
	ZTW (casc.)	64	0.0005	0.1		
	ZTW (ens.)	64	0.001	0.0		
ConvNeXt-T	SDN	64	0.0005	0.1	After blocks: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26	• 3 × 3 Conv (stride = 2, out_channels = half) • SDNPool (Kaya et al., 2019) • Linear
	PBEE	64	0.0005	0.1		
	GPF	64	0.0001	0.01		
	L2W	64	0.1	0.0001		
	ZTW (casc.)	64	0.0005	0.1		
	ZTW (ens.)	64	0.001	0.0		
ViT-B	SDN	64	0.005	0.0001	After blocks: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	• CLSPool • Linear
	PBEE	64	0.005	0.0001		
	GPF	64	0.0005	0.01		
	L2W	64	10.0	0.0		
	ZTW (casc.)	64	0.005	0.0001		
	ZTW (ens.)	64	0.001	0.0		
SwinV2-S	SDN	64	0.001	0.0	After blocks: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26	• SDNPool (Kaya et al., 2019) • Linear
	PBEE	64	0.001	0.0		
	GPF	64	0.0005	0.0		
	L2W	64	0.1	0.0		
	ZTW (casc.)	64	0.005	0.0001		
	ZTW (ens.)	64	0.001	0.0		

with knowledge distillation. We postulate that focusing on reducing the computational waste in a safe and stable way is an important direction for future research in deep learning.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Code publicly available. Link to the code shared in the paper. Only common and publicly available datasets have been used.

Acknowledgments

This research was funded by Foundation for Polish Science (grant no POIR.04.04.00-00-14DE/18-00 carried out within the Team-Net program co-financed by the European Union under the European Regional Development Fund) and National Science Centre, Poland grant no 2020/39/B/ST6/01511. Filip Szatkowski is supported by National Science Centre (Poland) Grant No. 2022/45/B/ST6/02817. The authors have applied a CC BY license to any Author Accepted Manuscript (AAM) version arising from this submission, in accordance with the grants' open access conditions. We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Centers: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2023/016270.

Appendix A. Training details

All experiments were performed using the PyTorch framework and NVIDIA RTX1080 Ti, RTX2080 Ti, RTX3080, V100 or A100 GPUs. We provide the source code for our experiments at: https://github.com/gmum/Zero-Time-Waste

Table A.2
Hyperparameters used for finetuning BERT-base backbone networks for the GLUE tasks.

Task	Epochs	Learning rate	Batch size
MRPC	5	0.00002	16
RTE	5	0.00002	16
SST2	3	0.00001	16
QNLI	3	0.00001	16
QQP	3	0.00005	32

A.1. Computer vision

We use the pre-trained models from the `torchvision` library[4] for the backbone networks. We consider only the "IC-only" setup with frozen backbone weights and evaluate the networks as proposed in Kaya et al. (2019). In Table A.1 we list the hyperparameters used for training each of the early-exit models.

We use AdamW for all methods except L2W, and train the models for 15 epochs with the cosine annealing scheduler. We first tune the learning rate γ by picking the one with the best results from the following search space: $\{0.0001, 0.0005, 0.001, 0.005, 0.01\}$. After this we select the optimal weight decay λ in the same way with the search space: $\{0.0, 0.0001, 0.001, 0.01, 0.1\}$. In cascading, we apply Layer Normalization (Ba, Kiros, & Hinton, 2016) to the output of the preceding IC $l_{m-1}(x)$ as part of the cascading head $l_m(x)$. We always use $\gamma \in \{0.001, 0.01, 0.1\}$ with AdamW without weight decay for ensembling in ZTW, and we always train the ensembles for 2 epochs. For L2W (Han et al., 2022), which we train with SGD with momentum, we use the following search space for learning rate: $\{0.0001, 0.001, 0.01, 0.1, 1.0, 10.0\}$. As for the other hyperparameters specific to L2W, we use the same values as the original authors did. As pooling layers we reuse the SDN pooling proposed by Kaya et al. (2019), which is defined as:

$$\text{sdn\_pool}(x) = \gamma \cdot \text{avg\_pool}(x) + (1 - \gamma) \cdot \text{max\_pool}(x),$$

where γ is a learnable scalar parameter. It reduces the size of convolutional maps to 4×4. Note that this pooling layer does not use sigmoid(γ) as proposed in Lee, Gallagher, and Tu (2016).

[4] https://pytorch.org/vision/0.14/models.html#classification.

Table A.3
Hyperparameters used for training BERT-base model on the NLP datasets.

Dataset	Method	Batch size	γ	λ	Head placement	Head architecture
MRPC	SDN	32	0.005	0.0001	After blocks: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	• CLSPool • Linear
	PBEE	32	0.005	0.0001		
	GPF	32	0.0005	0.0001		
	L2W	32	0.5	0.0001		
	ZTW (casc)	32	0.01	0.0001		
	ZTW (ens)	32	0.1	0.0		
QNLI	SDN	32	0.01	0.0001	After blocks: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	• CLSPool • Linear
	PBEE	32	0.01	0.0001		
	GPF	32	0.0001	0.0001		
	L2W	32	0.5	0.0001		
	ZTW (casc)	32	0.01	0.0001		
	ZTW (ens)	32	0.001	0.0		
QQP	SDN	32	0.05	0.0001	After blocks: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	• CLSPool • Linear
	PBEE	32	0.05	0.0001		
	GPF	32	0.0005	0.0001		
	L2W	32	2.0	0.0001		
	ZTW (casc)	32	0.01	0.0001		
	ZTW (ens)	32	0.01	0.0		
RTE	SDN	32	0.005	0.0001	After blocks: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	• CLSPool • Linear
	PBEE	32	0.005	0.0001		
	GPF	32	0.0005	0.0005		
	L2W	32	0.5	0.0001		
	ZTW (casc)	32	0.01	0.0001		
	ZTW (ens)	32	0.001	0.0		
SST2	SDN	32	0.01	0.0001	After blocks: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	• CLSPool • Linear
	PBEE	32	0.01	0.0001		
	GPF	32	0.001	0.0001		
	L2W	32	0.5	0.0001		
	ZTW (casc)	32	0.01	0.0001		
	ZTW (ens)	32	0.1	0.0		
20 Newsgroups	SDN	32	0.05	0.0001	After sub-blocks: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23	• CLSPool • Linear
	PBEE	32	0.05	0.0001		
	GPF	32	0.0005	0.0001		
	L2W	32	1.0	0.0001		
	ZTW (casc)	32	0.005	0.001		
	ZTW (ens)	32	0.01	0.0		

Additionally, in Appendix B.1 we present the results for the networks from Kaya et al. (2019) for CIFAR-10, CIFAR-100, and Tiny ImageNet, which appeared in our original work (Zhang et al., 2019). For CIFAR-10 and CIFAR-100 we train ICs for 50 epochs using the Adam optimizer with learning rate set to 0.001, but lowered by a factor of 10 after 15 epochs. When training on Tiny ImageNet, the learning rate is additionally lowered again by the same factor after epoch 40. To train the ensembling part of our method, we run SGD on the training dataset for 500 epochs. Since both the dataset and the model are very small, we use a high number of epochs to ensure convergence.

A.2. Natural language processing

For the NLP experiments we use BERT-Base from the Hugging Face Transformers library (Wolf et al., 2020). We fine-tune model on each downstream task to obtain the backbone networks. Fine-tuning hyperparameters for each task are shown in Table A.2.

The training follows the same procedure as described for computer vision in Appendix A.1. The hyperparameters used on every dataset are listed in Table A.3.

Transformer layers output hidden state for each token in the input sequence, so we use a pooling layer that simply discards all hidden states except the one corresponding to the CLS token, and then applies a simple fully-connected layer.

A.3. Reinforcement learning

We set the Atari environments as follows. Every fourth frame (frame skipping) and the one immediately before it are max-pooled. The resulting frame is then rescaled to size 84×84 and converted into grayscale. At every step the agent has a 0.1 probability of taking the previous action irrespective of the policy probabilities (sticky actions). This is added to introduce stochasticity into the environment to avoid cases when the policy converges to a simple strategy that results in the same actions taken in every run. Furthermore, the environment termination flag is set when a life is lost. Finally, the signum function of the reward is taken (reward clipping). The above setup is fairly common and we base our code on the popular Stable Baselines repository (Raffin et al., 2019).

Using that environment setup we use the PPO algorithm to train the policy, and then extract the base network by discarding the value network. We use the following PPO hyperparameters: learning rate $2.5 \cdot 10^{-4}$, 128 steps to run for each environment per update, batch size 256, 4 epochs of surrogate loss optimization, clip range (ϵ) 0.1, entropy coefficient 0.01, value function coefficient 0.5, discount factor 0.99, 0.95 as the trade-off of bias vs variance factor for Generalized Advantage Estimator (Schulman, Moritz, Levine, Jordan, & Abbeel, 2015), and the maximum value for the gradient clipping 0.5. The policy is trained for 10^7 environment time steps in total.

We use the standard 'NatureCNN' (Mnih et al., 2015) architecture with three convolutional layers and a single fully connected layer. We attach two ICs after the first and the second layer. Similarly as in the supervised setting, each IC has a single convolutional layer, an SDN pooling layer and a fully connected layer. The convolutional layer has stride set to 4 and preserves the number of channels.

To train the ICs, the early-exit policy interacts with the environment. In each step, an IC is chosen uniformly, and the action chosen by that IC is taken. However, the (o, a_p) tuple is actually saved to the replay buffer, with o and a_p being the observation and the action of the original policy, respectively. After 128 concurrent steps on 8 environments

Table B.1
Results on three computer vision datasets: Cifar-10, Cifar-100 and Tiny ImageNet. Test accuracy (in percentages) obtained using the time budget: 25%, 50%, 75%, 100% of the base network and without any limits ("Max"). The first column shows the test accuracy of the base network. We bold multiple results when there is no meaningful statistical difference between them.

Dataset	Model	25%	50%	75%	100%	Max
ResNet-56						
CIFAR-10 (92.0 ± 0.2)	SDN	77.7 ± 1.0	87.3 ± 0.5	91.1 ± 0.2	**92.0 ± 0.1**	**92.1 ± 0.2**
	PBEE	69.8 ± 1.3	81.8 ± 0.3	87.5 ± 0.1	91.0 ± 0.3	**92.1 ± 0.3**
	ZTW	**80.3 ± 1.0**	**88.7 ± 0.4**	**91.5 ± 0.2**	**92.1 ± 0.3**	**92.1 ± 0.3**
CIFAR-100 (68.4 ± 0.2)	SDN	47.1 ± 0.2	57.2 ± 0.4	64.7 ± 0.6	69.0 ± 0.2	69.7 ± 0.2
	PBEE	45.2 ± 0.5	53.5 ± 0.5	60.1 ± 0.5	67.0 ± 0.2	69.0 ± 0.2
	ZTW	**51.3 ± 0.4**	**62.1 ± 0.3**	**68.4 ± 0.4**	**70.7 ± 0.1**	**70.9 ± 0.1**
Tiny ImageNet (53.9 ± 0.3)	SDN	31.2 ± 0.2	41.2 ± 0.3	49.9 ± 0.4	54.5 ± 0.5	54.7 ± 0.4
	PBEE	29.0 ± 0.6	37.6 ± 0.3	48.2 ± 0.4	53.4 ± 0.6	54.3 ± 0.4
	ZTW	**35.2 ± 0.7**	**46.2 ± 0.4**	**53.7 ± 0.3**	**56.3 ± 0.3**	**56.4 ± 0.3**
MobileNet						
CIFAR-10 (90.6 ± 0.2)	SDN	**86.1 ± 0.5**	**90.5 ± 0.2**	90.8 ± 0.1	90.7 ± 0.2	90.9 ± 0.1
	PBEE	76.3 ± 0.9	85.9 ± 0.3	89.7 ± 0.3	90.9 ± 0.2	91.1 ± 0.1
	ZTW	**86.7 ± 0.7**	**90.9 ± 0.3**	**91.4 ± 0.2**	**91.4 ± 0.1**	**91.5 ± 0.1**
CIFAR-100 (65.1 ± 0.3)	SDN	**54.3 ± 1.4**	63.5 ± 0.8	66.8 ± 0.4	67.8 ± 0.1	67.9 ± 0.1
	PBEE	47.1 ± 2.7	61.6 ± 0.7	61.6 ± 0.7	67.0 ± 0.3	68.0 ± 0.3
	ZTW	**54.5 ± 1.1**	**65.2 ± 0.5**	**68.4 ± 0.3**	**69.0 ± 0.1**	**69.1 ± 0.1**
Tiny ImageNet (59.3 ± 0.1)	SDN	**35.6 ± 1.3**	**47.1 ± 0.6**	55.3 ± 0.3	58.9 ± 0.2	59.7 ± 0.1
	PBEE	26.7 ± 1.5	38.4 ± 2.0	50.3 ± 0.8	55.6 ± 0.3	59.7 ± 0.0
	ZTW	**37.3 ± 2.8**	**49.5 ± 1.9**	**56.7 ± 0.6**	**59.7 ± 0.4**	**60.2 ± 0.1**
WideResNet						
CIFAR-10 (94.4 ± 0.1)	SDN	83.8 ± 1.3	91.7 ± 0.5	94.1 ± 0.2	94.4 ± 0.1	94.4 ± 0.2
	PBEE	78.0 ± 1.9	84.0 ± 1.4	90.3 ± 0.5	93.8 ± 0.1	94.4 ± 0.1
	ZTW	**86.7 ± 0.7**	**92.9 ± 0.3**	**94.5 ± 0.1**	**94.7 ± 0.1**	**94.7 ± 0.1**
CIFAR-100 (75.1 ± 0.1)	SDN	55.9 ± 1.5	65.1 ± 0.9	71.6 ± 0.4	75.0 ± 0.1	75.4 ± 0.1
	PBEE	46.7 ± 2.0	57.2 ± 1.3	66.0 ± 0.6	73.2 ± 0.2	75.4 ± 0.2
	ZTW	**59.5 ± 0.6**	**69.1 ± 0.9**	**74.5 ± 0.6**	**76.2 ± 0.3**	**76.4 ± 0.2**
Tiny ImageNet (59.6 ± 0.6)	SDN	36.8 ± 0.1	46.0 ± 1.0	54.6 ± 0.7	59.4 ± 0.8	59.7 ± 0.7
	PBEE	29.9 ± 0.9	37.8 ± 0.6	52.7 ± 0.6	58.5 ± 0.9	59.7 ± 0.7
	ZTW	**40.0 ± 0.3**	**50.1 ± 0.2**	**57.5 ± 0.4**	**60.2 ± 0.1**	**60.3 ± 0.2**
VGG						
CIFAR-10 (93.0 ± 0.0)	SDN	86.0 ± 0.3	92.1 ± 0.1	**93.0 ± 0.0**	**93.0 ± 0.0**	**93.0 ± 0.0**
	PBEE	75.0 ± 0.2	86.0 ± 0.2	91.0 ± 0.3	92.9 ± 0.2	93.1 ± 0.1
	ZTW	**87.1 ± 0.1**	**92.5 ± 0.1**	**93.2 ± 0.2**	**93.2 ± 0.2**	**93.2 ± 0.2**
CIFAR-100 (70.4 ± 0.3)	SDN	58.5 ± 0.4	67.2 ± 0.1	70.6 ± 0.3	71.4 ± 0.2	71.5 ± 0.4
	PBEE	51.2 ± 0.2	65.3 ± 0.3	65.3 ± 0.3	70.9 ± 0.5	72.0 ± 0.4
	ZTW	**60.2 ± 0.2**	**69.3 ± 0.4**	**72.6 ± 0.1**	**73.5 ± 0.3**	**73.6 ± 0.4**
Tiny ImageNet (59.0 ± 0.2)	SDN	40.0 ± 1.0	50.5 ± 0.2	57.4 ± 0.5	**59.6 ± 0.3**	59.7 ± 0.3
	PBEE	31.0 ± 1.6	45.2 ± 0.6	55.2 ± 0.3	**60.1 ± 0.5**	**60.2 ± 0.5**
	ZTW	**41.4 ± 0.5**	**52.3 ± 0.4**	**59.3 ± 0.4**	**60.1 ± 0.5**	**60.5 ± 0.4**

that buffer is used to train the ICs with behavioral cloning. That is, Kullback–Leibler divergence between the PPO policy actions and the IC actions is used as the cost function. This is done for 5 epochs with batch size set to 64 and 128 for cascading stage and geometric ensembling stage, respectively. The entire process is repeated until 10^6 or more steps in total are taken.

Appendix B. Additional experimental results

This section contains additional results which were omitted in the main part of the paper.

B.1. Computer vision

For brevity, in the main part of the paper we have only shown a table summarizing the results of acceleration on multiple architectures for the ImageNet dataset. Here, we provide an alternative representation of these results. Moreover, we include the results for CIFAR-10, CIFAR-100 and TinyImageNet datasets in Table B.1. Figs. B.4, B.5, B.6, and B.7 show results of the on ImageNet, CIFAR-10, CIFAR-100 and Tiny ImageNet, respectively. Additionally, Figs. B.8, B.9, B.10 show values of Hindsight Improvability for CIFAR-10, CIFAR-100 and Tiny ImageNet, respectively.

B.2. Transfer to the OCT2017 dataset

We investigate how early exit methods work in a practical transfer learning setting by training the ICs on different dataset then the backbone network is trained on. We use ResNet-50 from the torchvision package, which is pre-trained on ImageNet. To obtain a baseline standard classifier, we remove the final linear layer of the pretrained classifier and train a new linear layer with the number of outputs corresponding to the number of classes in the target dataset. Only then we proceed to train the ICs.

We use the OCT-2017 medical dataset (Kermany et al., 2018) as the target dataset. The training dataset consists of 83 484 high-resolution retinal optical coherence tomography images categorized into four classes, with one class meaning healthy sample, and three diseases. Table B.2 shows that ZTW outperforms other methods by a significant margin, and manages to cut down the time required to obtain the accuracy of the baseline by over 75%. This suggests that leveraging

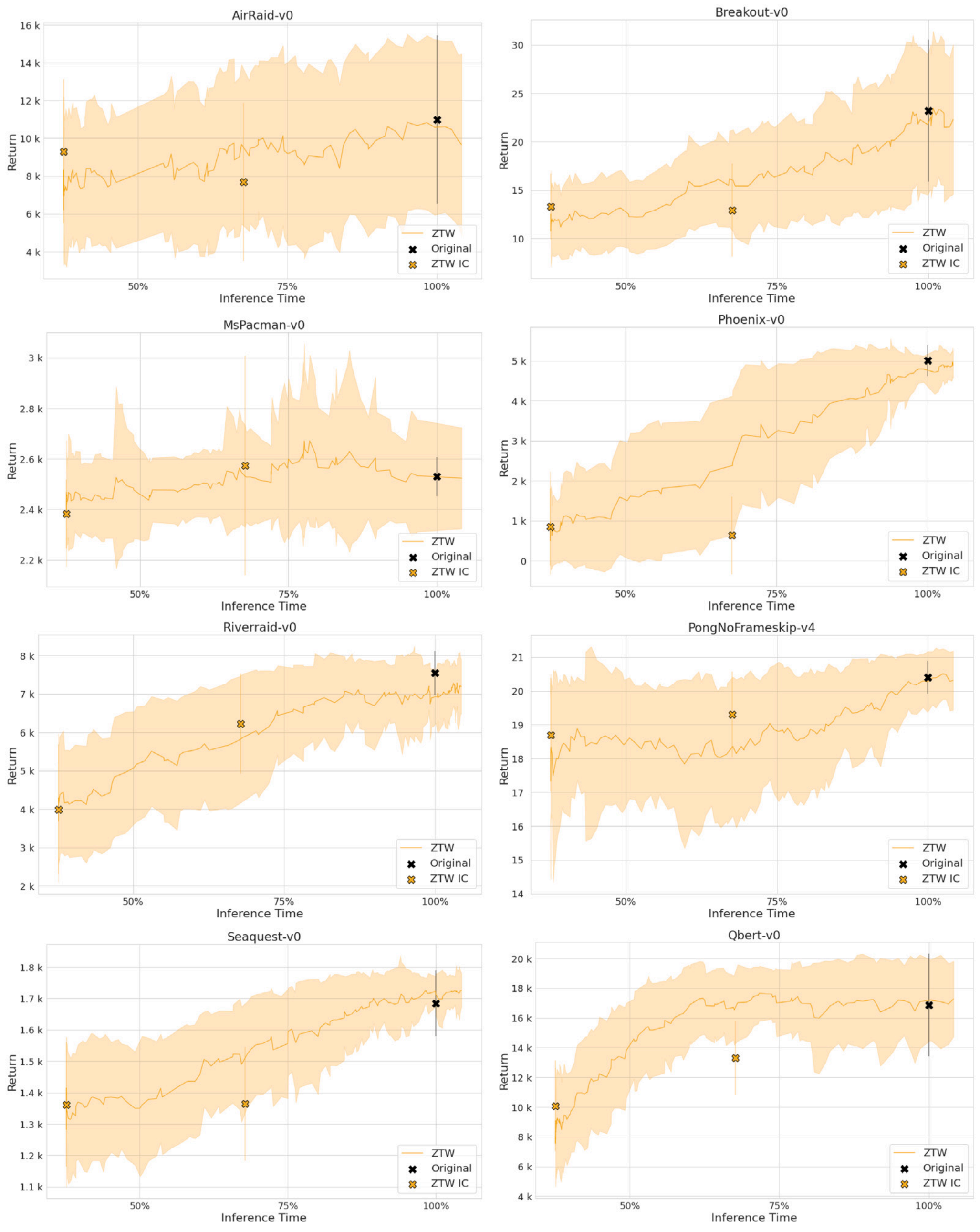


Fig. B.1. Mean and standard deviation of returns for multiple confidence thresholds on various Atari 2600 environments. Some environments allow significant computational savings with a negligible or no impact on performance.

Table B.2
Results on the OCT2017 dataset when using an ImageNet pretrained core network. Test accuracy (in percentages) obtained using the time budget: 25%, 50%, 75%, 100% of the base network and Max without any limits.

ResNet-50 (94.6)					
Model	25%	50%	75%	100%	Max
SDN	81.5	93.8	94.6	94.6	94.6
PBEE	56.5	90.3	90.3	94.5	95.2
ZTW	89.4	98.0	98.4	98.5	98.5

the power of previous ICs is especially useful when the features are not perfectly adjusted to the problem at hand, i.e. were trained for ImageNet classification and used for pathology classification data from a completely different domain.

B.3. Reinforcement learning experiments

In Fig. B.1 we show the results for all eight Reinforcement Learning environments that we ran our experiments on. Degree of time savings depends heavily on the environment. For some of the environments, such as AirRaid and Pong, the ICs obtain a similar return to that of the original policy. Because of that the resulting plot is almost flat, allowing for significant inference time reduction without any performance drop. Other environments, such as Seaquest, Phoenix and Riverraid, allow to gradually trade-off performance for inference time just as in the supervised setting.

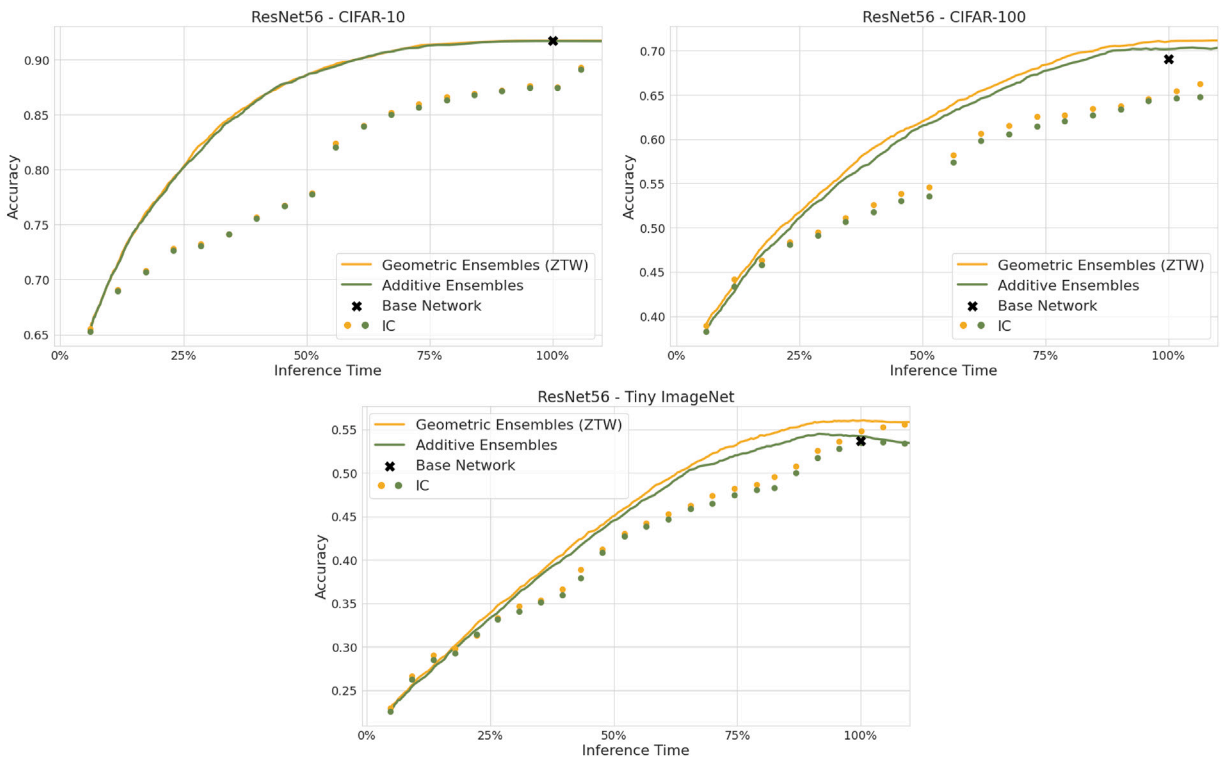


Fig. B.2. Comparison of geometric and additive ensembling on ResNet-56 with *cascade connections*, conducted on CIFAR-10, CIFAR-100, and Tiny ImageNet.

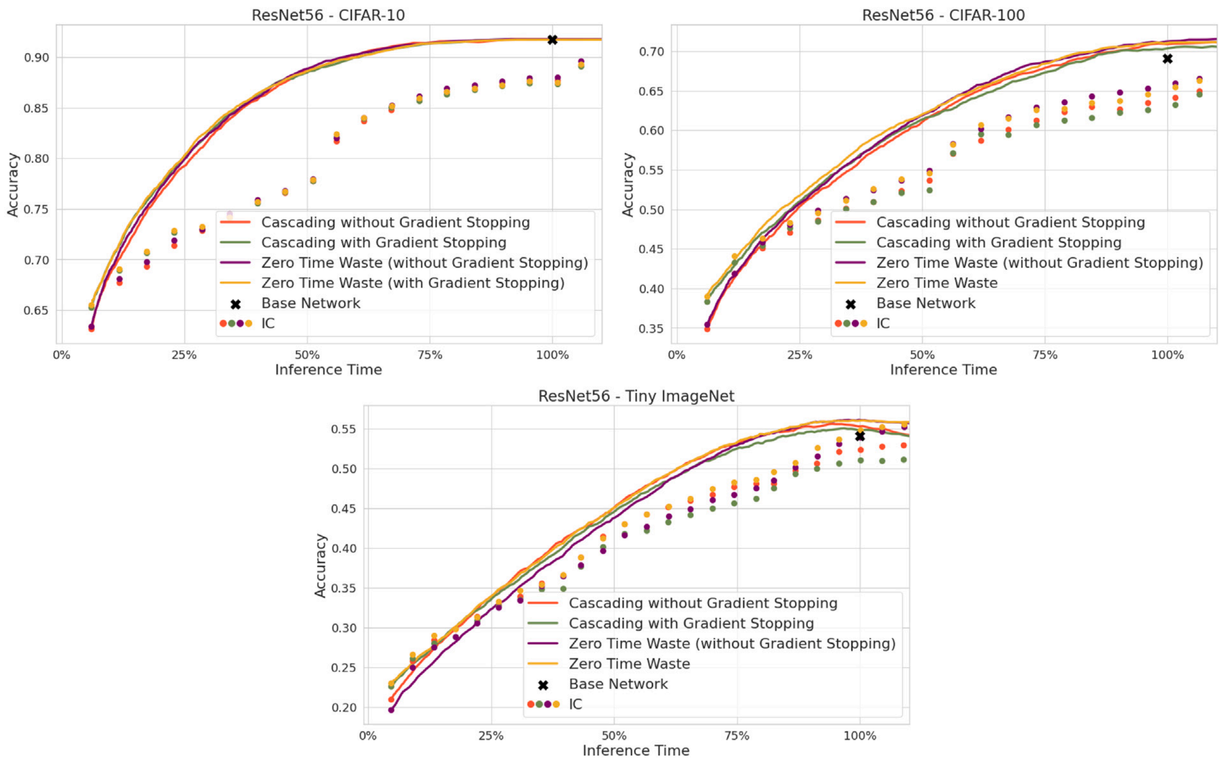


Fig. B.3. Effects of stopping gradient in ResNet-56 trained on CIFAR-10, CIFAR-100, and Tiny ImageNet.

B.4. Geometric vs. additive ensembles

In the main paper we proposed two variants of ensembling, and then subsequently analyzed differences between them. In Fig. B.2 we provide additional accuracy vs. inference time results for other datasets.

B.5. Stop gradients in cascade connections

In the main paper we demonstrated that stopping the gradient between ICs in cascade connections improves overall performance. In Fig. B.3 we provide additional results for other datasets.

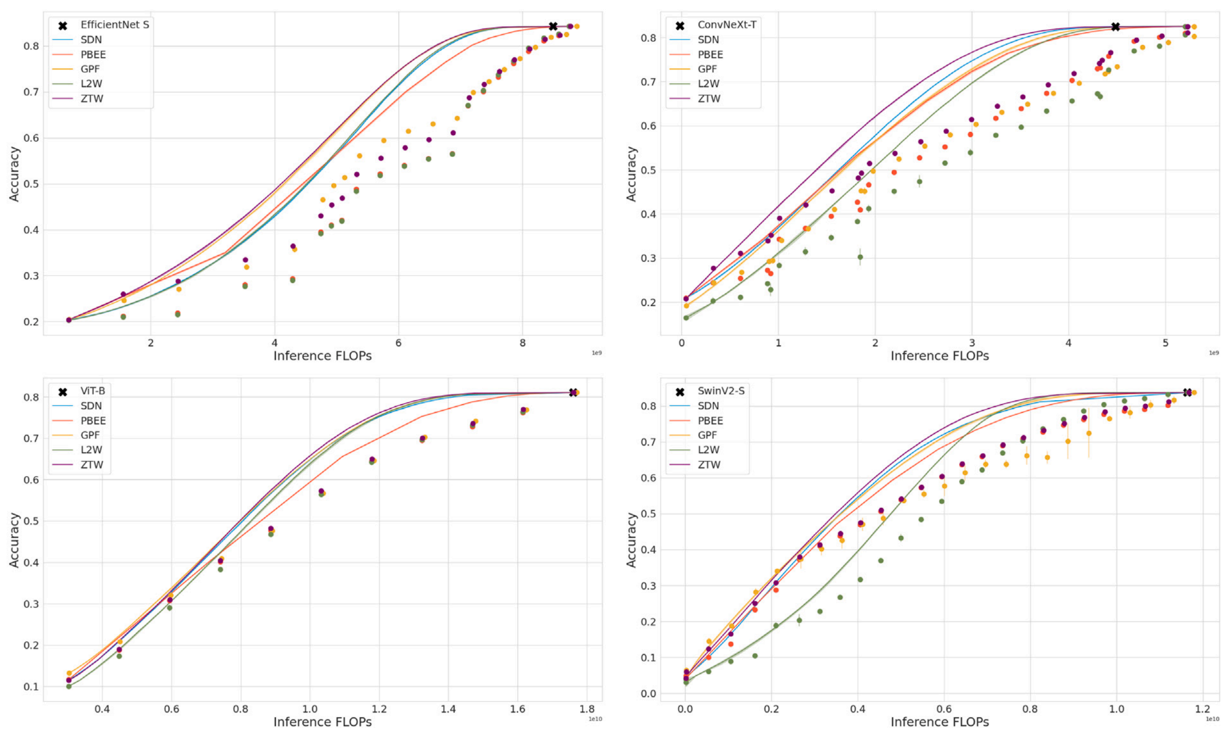


Fig. B.4. Inference time vs. accuracy obtained on various architectures trained on ImageNet.

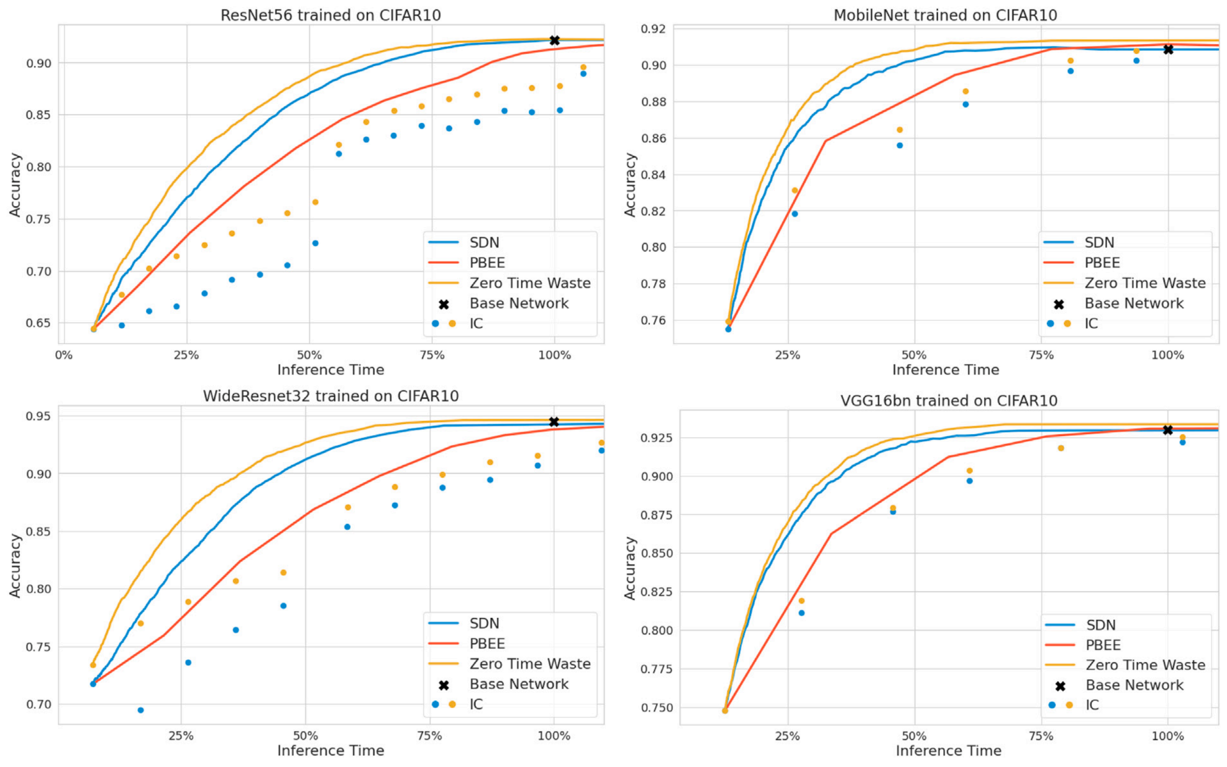


Fig. B.5. Inference time vs. accuracy obtained on various architectures trained on CIFAR-10.

B.6. Impact of the number of classes

Additionally, we check how the number of classes in the given problem impacts the results of each method. To do this, we take the CIFAR-10 dataset, which consists of 10 classes and divide the examples into two more general classes, which can be approximately described as modes of transportation (includes airplane, automobile, horse, ship, truck) and animals (bird, cat, deer, dog, frog). Thus, we obtain a dataset for binary classification which we dub CIFAR-2. We train and evaluate the proposed methods on this dataset with different backbones. Results,

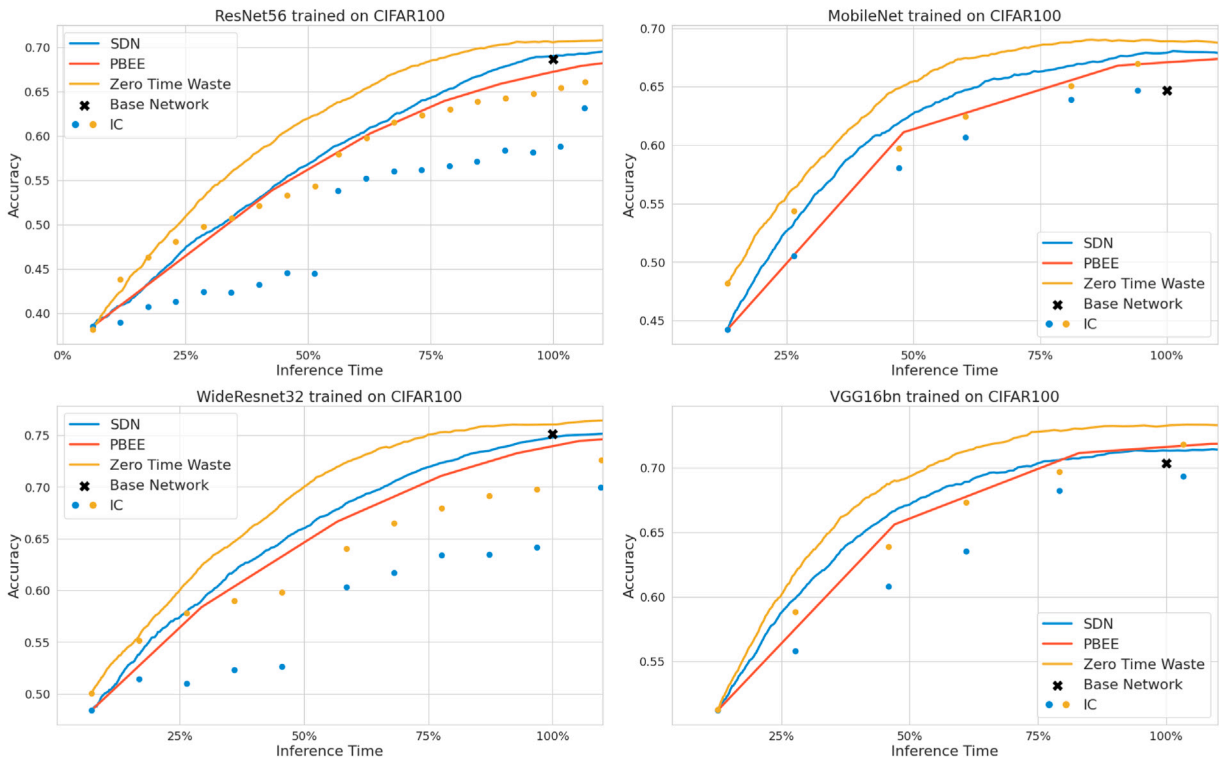


Fig. B.6. Inference time vs. accuracy obtained on various architectures trained on CIFAR-100.

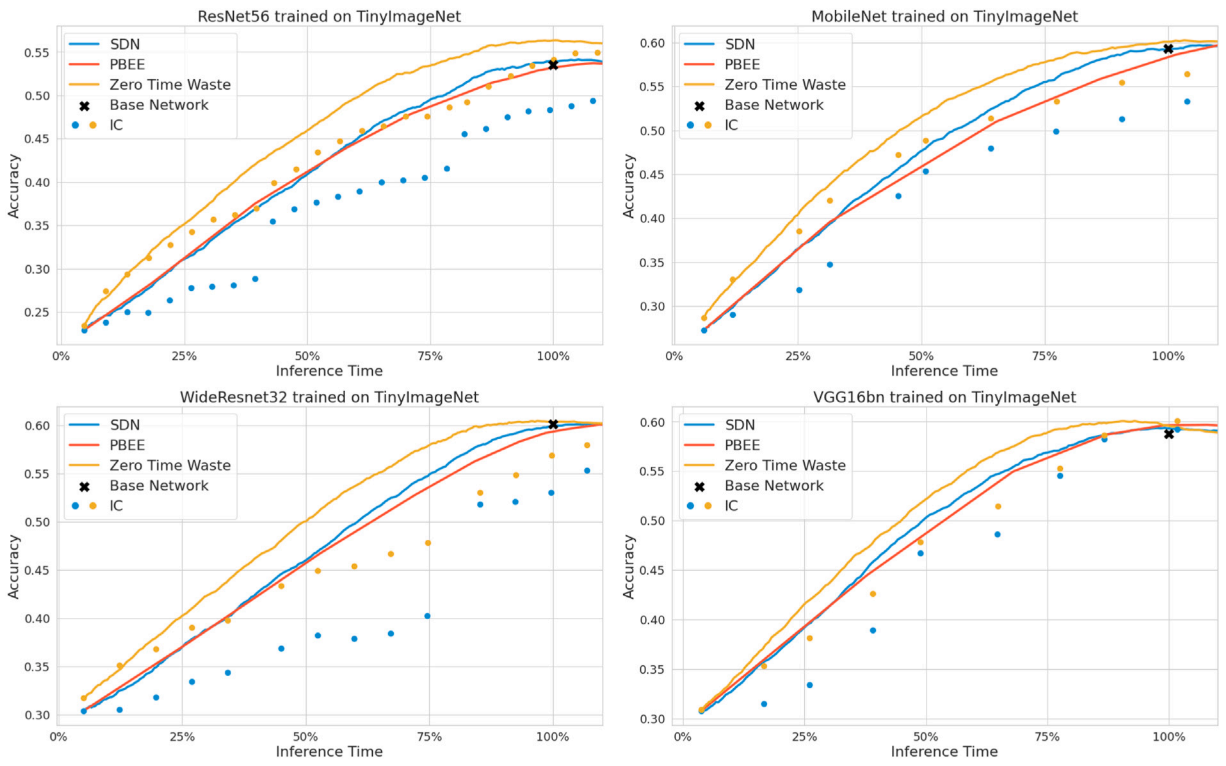


Fig. B.7. Inference time vs. accuracy obtained on various architectures trained on Tiny ImageNet.

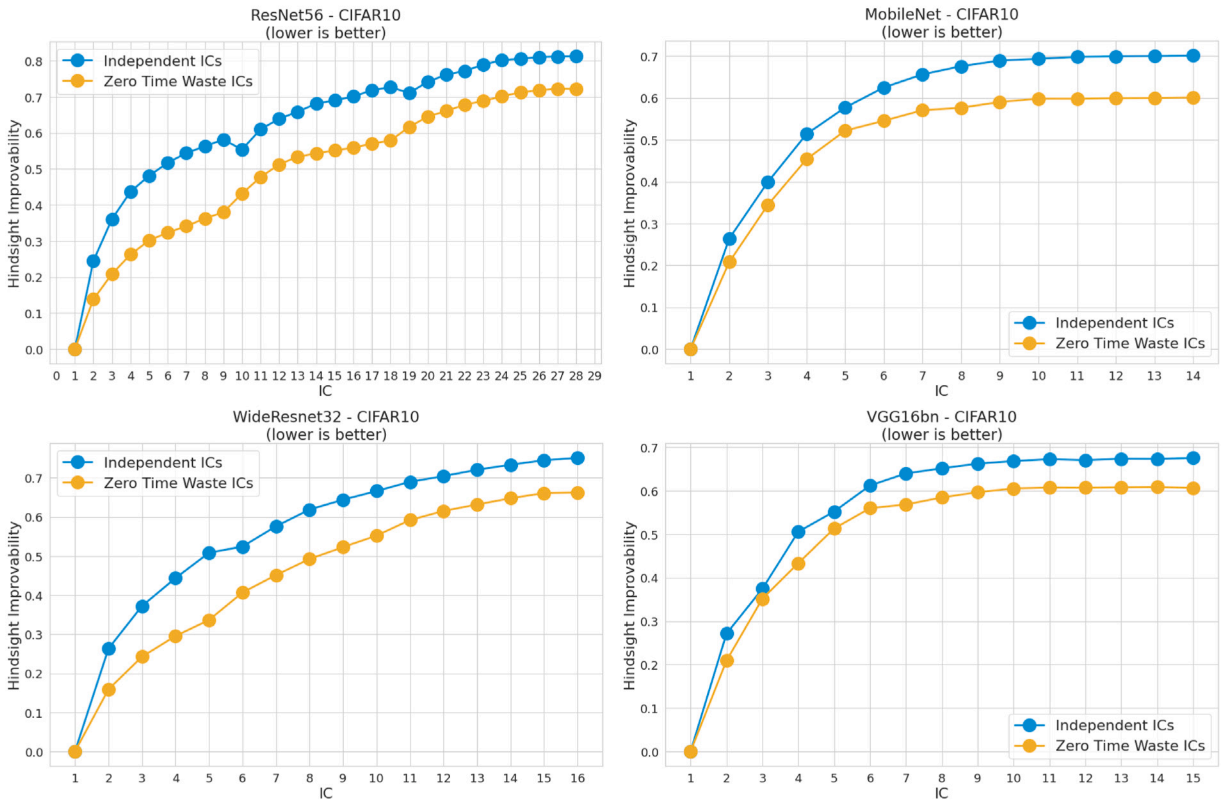


Fig. B.8. Hindsight Improvability of various architectures trained on CIFAR-10.

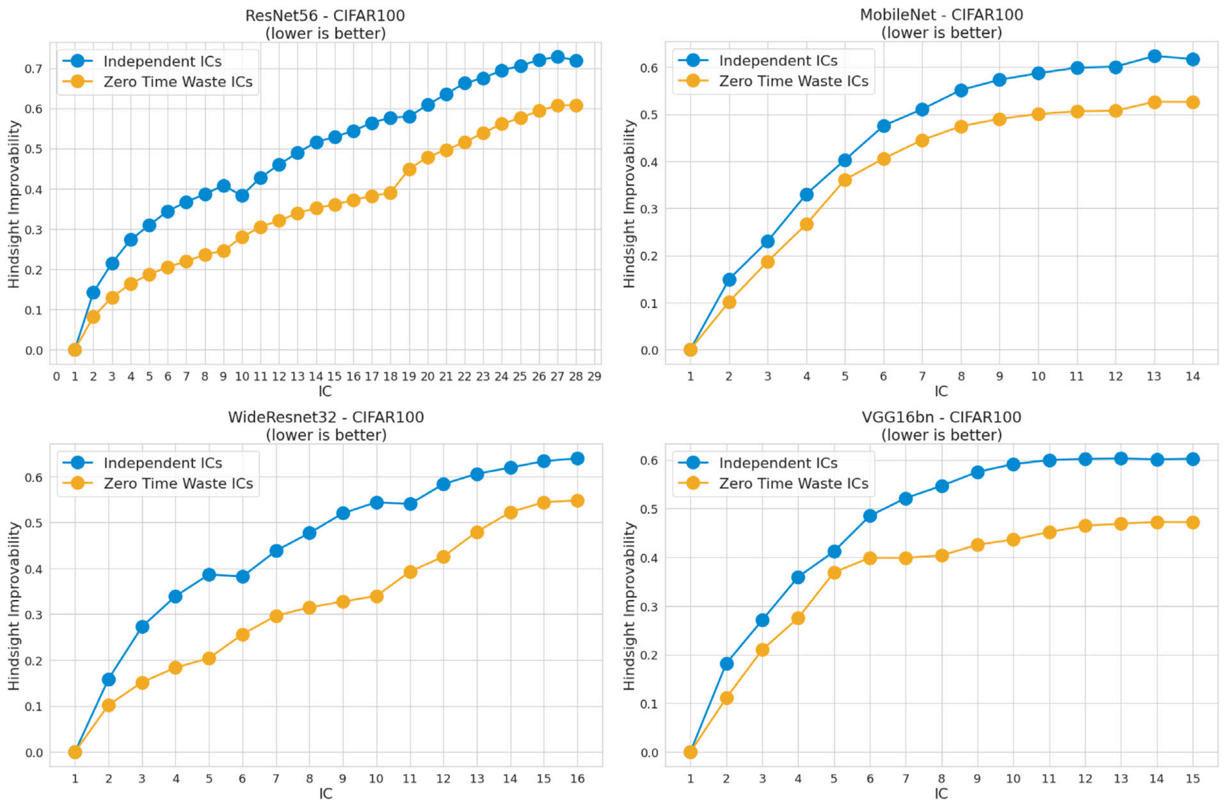


Fig. B.9. Hindsight Improvability of various architectures trained on CIFAR-100.

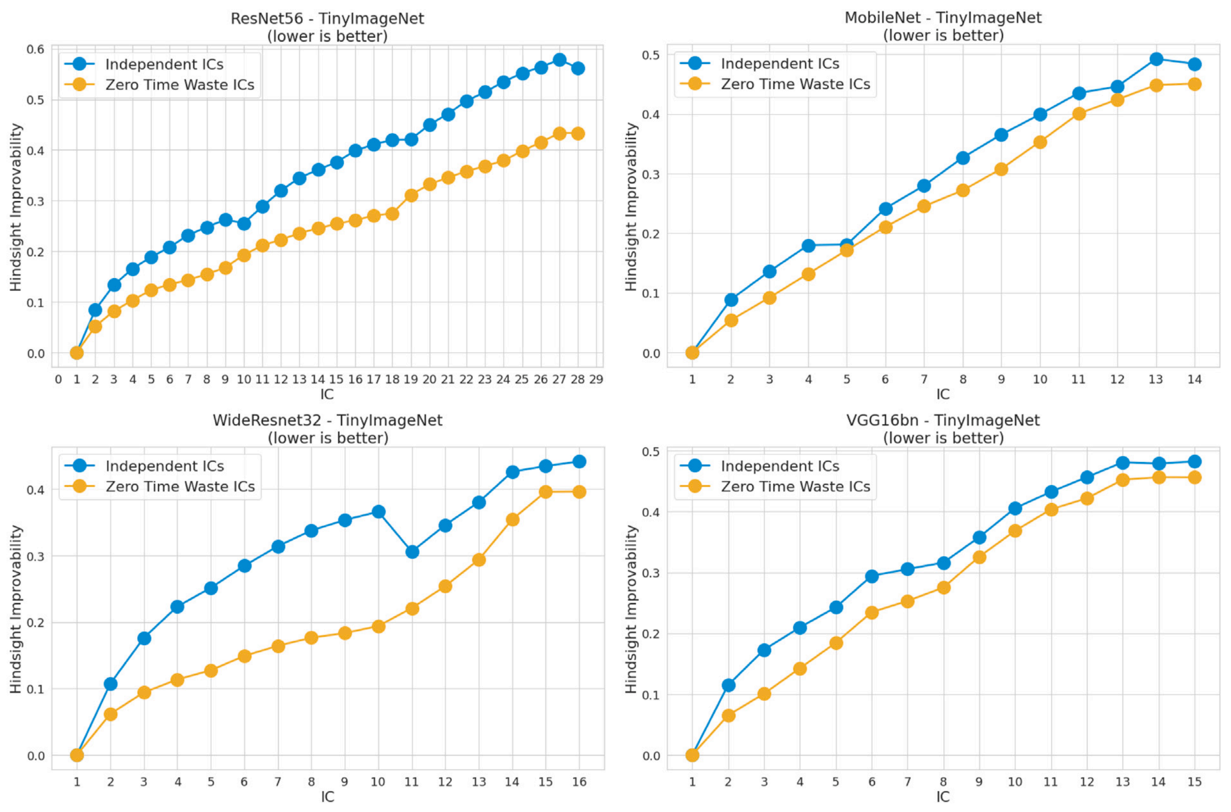


Fig. B.10. Hindsight Improvability of various architectures trained on Tiny ImageNet.

Table B.3
Results on the CIFAR-2 dataset.

ResNet-56						
Data	Model	25%	50%	75%	100%	Max
ResNet-56	SDN	95.5	96.5	96.5	96.5	96.5
	PBEE	91.2	94.1	96.3	96.5	96.6
	ZTW	95.7	96.6	96.6	96.6	96.6
VGG	SDN	96.6	97.6	97.6	97.6	97.7
	PBEE	91.2	96.4	97.2	97.4	97.6
	ZTW	96.7	97.6	97.7	97.7	97.7
WideResNet	SDN	95.2	97.0	97.3	97.3	97.4
	PBEE	89.3	93.0	95.9	97.0	97.4
	ZTW	96.3	97.4	97.6	97.6	97.6
MobileNet	SDN	95.7	96.4	96.4	96.4	96.4
	PBEE	91.9	94.3	96.2	96.4	96.4
	ZTW	96.0	96.4	96.4	96.4	96.4

summed up in Table B.3, show that although performance of ZTW is always on par or better than the baselines, the gap in performance is much smaller, with SDN achieving identical performance in some cases. Although, this might be due to the fact that CIFAR-2 is simpler than original CIFAR-10, we note that Zero Time Waste is better suited to non-binary classification problems.

References

Ariely, D., & Norton, M. I. (2011). From thinking too little to thinking too much: a continuum of decision making. *WIREs Cognitive Science*, *2*(1), 39–46, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcs.90.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv:1607.06450.

Banino, A., Balaguer, J., & Blundell, C. (2021). Pondernet: Learning to ponder. arXiv preprint arXiv:2107.05407.

Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv:1308.3432.

Bentéjac, C., Csörgő, A., & Martínez-Muñoz, G. (2020). A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 1–31.

Berestizshevsky, K., & Even, G. (2019). Dynamically sacrificing accuracy for reduced computation: cascaded inference based on softmax confidence. In *ICANN, Proceedings of the international conference on artificial neural networks* (pp. 306–320). Springer.

Davis, A., & Arel, I. (2013). Low-rank approximations for conditional feedforward computation in deep neural networks. arXiv:1312.4461.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248–255). Ieee.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, Volume 1 (Long and short papers)* (pp. 4171–4186). Minneapolis, Minnesota: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/N19-1423, URL https://aclanthology.org/N19-1423.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Proceedings of the international workshop on multiple classifier systems* (p. 15). Springer.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.

Dulac-Arnold, G., Mankowitz, D. J., & Hester, T. (2019). Challenges of real-world reinforcement learning. CoRR abs/1904.12901. arXiv:1904.12901. URL http://arxiv.org/abs/1904.12901.

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., et al. (2017). Spatially adaptive computation time for residual networks. In *CVPR, Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1039–1048).

Fort, S., Hu, H., & Lakshminarayanan, B. (2019). Deep ensembles: a loss landscape perspective. arXiv:1912.02757.

Gigerenzer, G., & Gaissmaier, W. (2011). Heuristic decision making. *Annual Review of Psychology*, *62*, 451–482.

Graves, A. (2016). Adaptive computation time for recurrent neural networks. arXiv preprint arXiv:1603.08983.

Grigorescu, S., Trasnea, B., Cocias, T., & Macesanu, G. (2020). A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, *37*(3), 362–386.

Gupta, K., Rahimi, A., Ajanthan, T., Mensink, T., Sminchisescu, C., & Hartley, R. (2020). Calibration of neural networks using splines. arXiv preprint arXiv:2006.12800.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., & Wang, Y. (2021). Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(11), 7436–7456.

Han, Y., Pu, Y., Lai, Z., Wang, C., Song, S., Cao, J., et al. (2022). Learning to weight samples for dynamic early-exiting networks. In *European conference on computer vision* (pp. 362–378). Springer.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *CVPR, Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

He, Y., Zhang, X., & Sun, J. (2017). Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 1389–1397).

Herrmann, C., Bowen, R. S., & Zabih, R. (2020). Channel selection using gumbel softmax. In *European conference on computer vision* (pp. 241–257). Springer.

Hester, T., & Stone, P. (2013). TEXPLORE: Real-time sample-efficient reinforcement learning for robots. *Machine Learning*, *90*(3), 385–429. http://dx.doi.org/10.1007/s10994-012-5322-7.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. In *Proceedings of the NIPS workshop on deep learning and representation learning*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861.

Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., & Weinberger, K. Q. (2018). Multi-scale dense networks for resource efficient image classification. In *International conference on learning representations*.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., et al. (2020). TinyBERT: Distilling BERT for natural language understanding. In *Findings of the association for computational linguistics: EMNLP 2020* (pp. 4163–4174).

Jung, S., Hwang, S., Shin, H., & Shim, D. H. (2018). Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. *IEEE Robotics and Automation Letters*, *3*(3), 2539–2544.

Kahneman, D. (2017). *Thinking, fast and slow*. Farrar, Straus and Giroux.

Kaya, Y., Hong, S., & Dumitras, T. (2019). Shallow-deep networks: Understanding and mitigating network overthinking. In *ICML, Proceedings of the international conference on machine learning* (pp. 3301–3310).

Kermany, D. S., Goldbaum, M., Cai, W., Valentim, C. C., Liang, H., Baxter, S. L., et al. (2018). Identifying medical diagnoses and treatable diseases by image-based deep learning. *Cell*, *172*(5), 1122–1131.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kouris, A., Venieris, S. I., Rizakis, M., & Bouganis, C.-S. (2019). Approximate LSTMs for time-constrained inference: Enabling fast reaction in self-driving cars. arXiv:1905.00689.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90.

Lakshminarayanan, B., Pritzel, A., & Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *NIPS, Advances in neural information processing systems* (pp. 6402–6413).

Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Proceedings of the twelfth international conference on machine learning* (pp. 331–339).

Lee, C.-Y., Gallagher, P. W., & Tu, Z. (2016). Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial intelligence and statistics* (pp. 464–472). PMLR.

Lee, J., Kim, S., Kim, S., Jo, W., & Yoo, H.-J. (2021). GST: Group-sparse training for accelerating deep reinforcement learning. arXiv:2101.09650.

Li, C., Wang, G., Wang, B., Liang, X., Li, Z., & Chang, X. (2021). Dynamic slimmable network. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 8607–8617).

Li, H., Zhang, H., Qi, X., Ruigang, Y., & Huang, G. (2019). Improved techniques for training adaptive deep networks. In *2019 IEEE/CVF international conference on computer vision (ICCV)* (pp. 1891–1900).

Liao, K., Zhang, Y., Ren, X., Su, Q., Sun, X., & He, B. (2021). A global past-future early exit method for accelerating inference of pre-trained language models. In *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: Human language technologies* (pp. 2013–2023).

Lin, M., Fu, J., & Bengio, Y. (2019). Conditional computation for continual learning. arXiv preprint arXiv:1906.06635.

Lin, J., Rao, Y., Lu, J., & Zhou, J. (2017). Runtime neural pruning. *Advances in Neural Information Processing Systems*, *30*.

Liu, Z., Hu, H., Lin, Y., Yao, Z., Xie, Z., Wei, Y., et al. (2022). Swin transformer v2: Scaling up capacity and resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 12009–12019).

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., et al. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 10012–10022).

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 11976–11986).

Liu, C., Wang, Y., Han, K., Xu, C., & Xu, C. (2019). Learning instance-wise sparsity for accelerating deep models. arXiv preprint arXiv:1907.11840.

Livne, D., & Cohen, K. (2020). PoPS: Policy pruning and shrinking for deep reinforcement learning. arXiv:2001.05012.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Nie, X., Miao, X., Cao, S., Ma, L., Liu, Q., Xue, J., et al. (2021). Evomoe: An evolutional mixture-of-experts training framework via dense-to-sparse gate. arXiv preprint arXiv:2112.14397.

Panda, P., Sengupta, A., & Roy, K. (2016). Conditional deep learning for energy-efficient and enhanced pattern recognition. In *2016 design, automation & test in europe conference & exhibition (DATE)* (pp. 475–480). IEEE.

Phuong, M., & Lampert, C. (2019). Distillation-based training for multi-exit architectures. In *2019 IEEE/CVF international conference on computer vision (ICCV)* (pp. 1355–1364).

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., & Dormann, N. (2019). Stable baselines3. https://github.com/DLR-RM/stable-baselines3.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

Scardapane, S., Comminiello, D., Scarpiniti, M., Baccarelli, E., & Uncini, A. (2020). Differentiable branching in deep networks for fast inference. In *ICASSP, Proceedings of the IEEE international conference on acoustics, speech and signal processing* (pp. 4167–4171).

Scardapane, S., Scarpiniti, M., Baccarelli, E., & Uncini, A. (2020). Why should we add early exits to neural networks? arXiv:2004.12814.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197–227.

Schuitema, E., Buşoniu, L., Babuška, R., & Jonker, P. (2010). Control delay in Reinforcement Learning for real-time dynamic systems: A memoryless approach. In *2010 IEEE/RSJ international conference on intelligent robots and systems* (pp. 3226–3231).

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. arXiv:1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., et al. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538.

Sun, Y., & Ochiai, H. (2022). Homogeneous learning: Self-attention decentralized deep learning. *IEEE Access*, *10*, 7695–7703.

Sun, T., Zhou, Y., Liu, X., Zhang, X., Jiang, H., Cao, Z., et al. (2021). Early exiting with ensemble internal classifiers. arXiv preprint arXiv:2105.13792.

Tan, M., & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning* (pp. 6105–6114). PMLR.

Tan, M., & Le, Q. (2021). Efficientnetv2: Smaller models and faster training. In *International conference on machine learning* (pp. 10096–10106). PMLR.

Teerapittayanon, S., McDanel, B., & Kung, H.-T. (2016). Branchynet: Fast inference via early exiting from deep neural networks. In *ICPR, Proceedings of the international conference on pattern recognition* (pp. 2464–2469).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. arXiv preprint arXiv:1706.03762.

Veit, A., & Belongie, S. (2018). Convolutional networks with adaptive inference graphs. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 3–18).

Verelst, T., & Tuytelaars, T. (2020). Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2320–2329).

Viola, P., & Jones, M. J. (2004). Robust real-time face detection. *International Journal of Computer Vision*, *57*(2), 137–154.

Wang, X., & Li, Y. (2021). Harmonized dense knowledge distillation training for multi-exit architectures. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 35* (pp. 10218–10226).

Wang, Y., Lv, K., Huang, R., Song, S., Yang, L., & Huang, G. (2020). Glance and focus: a dynamic approach to reducing spatial redundancy in image classification. In *Advances in neural information processing systems, Vol. 33* (pp. 2432–2444).

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP workshop blackboxNLP: Analyzing and interpreting neural networks for NLP* (pp. 353–355). Brussels, Belgium: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/W18-5446, URL https://aclanthology.org/W18-5446.

Wang, X., Yu, F., Dou, Z.-Y., Darrell, T., & Gonzalez, J. E. (2018). Skipnet: learning dynamic routing in convolutional networks. In *ECCV, Proceedings of the European conference on computer vision* (pp. 409–424).

Wołczyk, M., Wójcik, B., Bałazy, K., Podolak, I. T., Tabor, J., Śmieja, M., et al. (2021). Zero time waste: Recycling predictions in early exit neural networks. In *Advances in neural information processing systems, Vol. 34* (pp. 2516–2528). URL https://proceedings.neurips.cc/paper/2021/file/149ef6419512be56a93169cd5e6fa8fd-Paper.pdf.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., et al. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: System demonstrations* (pp. 38–45). Online: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/2020.emnlp-demos.6, URL https://aclanthology.org/2020.emnlp-demos.6.

Yang, L., Han, Y., Chen, X., Song, S., Dai, J., & Huang, G. (2020). Resolution adaptive networks for efficient inference. In *2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR)* (pp. 2369–2378).

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? CoRR abs/1411.1792. arXiv:1411.1792. URL http://arxiv.org/abs/1411.1792.

Yu, H., Li, H., Hua, G., Huang, G., & Shi, H. (2023). Boosted dynamic neural networks. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 37* (pp. 10989–10997).

Zhang, H., He, Z., & Li, J. (2019). Accelerating the deep reinforcement learning with neural network compression. In *2019 international joint conference on neural networks (IJCNN)* (pp. 1–8). http://dx.doi.org/10.1109/IJCNN.2019.8852451.

Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., & Gao, Y. (2021). A survey on federated learning. *Knowledge-Based Systems*, *216*, Article 106775.

Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., & Wei, F. (2020). BERT loses patience: fast and robust inference with early exit. arXiv:2006.04152.