Predictive Exit: Prediction of Fine-Grained Early Exits for Computation- and Energy-Efficient Inference

Xiangjie Li, Chenfei Lou, Yuchi Chen, Zhengping Zhu, Yingtao Shen, Yehan Ma, An Zou

Shanghai Jiao Tong University
{superswift, cf_lou, citrate, zhuzhengping, doctorcoal, yehanma, an.zou}@sjtu.edu.cn

Abstract

By adding exiting layers to the deep learning networks, early exit can terminate the inference earlier with accurate results. However, the passive decision-making of whether to exit or continue the next layer has to go through every pre-placed exiting layer until it exits. In addition, it is hard to adjust the configurations of the computing platforms alongside the inference proceeds. By incorporating a low-cost prediction engine, we propose a Predictive Exit framework for computation- and energy-efficient deep learning applications. Predictive Exit can forecast where the network will exit (i.e., establish the number of remaining layers to finish the inference), which effectively reduces the network computation cost by exiting on time without running every pre-placed exiting layer. Moreover, according to the number of remaining layers, proper computing configurations (i.e., frequency and voltage) are selected to execute the network to further save energy. Extensive experimental results demonstrate that Predictive Exit achieves up to 96.2% computation reduction and 72.9% energy-saving compared with classic deep learning networks; and 12.8% computation reduction and 37.6% energy-saving compared with the early exit under state-of-the-art exiting strategies, given the same inference accuracy and latency.

Introduction

Deep learning approaches, such as convolution neural networks (CNNs), have achieved tremendous success in versatile applications. However, deploying the deep learning models on resource-constrained systems is challenging because of its huge computation and energy cost. Diving into the performance of each inference case, researchers (Teerapittayanon, McDanel, and Kung 2016; Figurnov et al. 2017; Wang et al. 2018) found that the significant growth in model complexity is only helpful to classifying a handful of complicated inputs correctly, and they might become "wasteful" for simple inputs. Motivated by this observation, early exit includes additional side branch classifiers (exiting layers) to some of the network layers. As shown in Fig. 1, compared with the classic deep learning network in (a), the additional exiting layer in (b) allows inference results for a large portion of test samples to exit the network early when samples have already been inferred with high confidence.

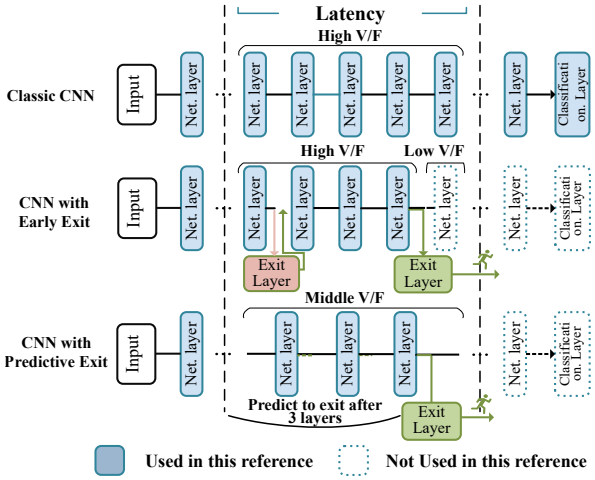


Figure 1: The architecture of classic deep learning network, early exit, and proposed Predictive Exit.

Despite being employed in burgeoning efforts to reduce computation cost and energy consumption in inference, early exits (Scardapane et al. 2020) are not capable of addressing the challenges listed below. First, there is a dilemma between fine-grained and coarse-grained placements of exiting points. Fine-grained exiting points lead to significant performance and energy overheads due to frequent execution of the exiting layers (Baccarelli et al. 2020). In contrast, coarse-grained exiting points may miss the opportunities to exit earlier. Second, the exiting layers have different topologies at different exiting points, which induce extra burdens to map the computation to computing hardware resources. Third, the energy-efficient computing platforms using early exits can only reduce the computing voltage and frequency after exiting. During the inference process, run-time computing configuration adjustment, such as dynamic voltage and frequency scaling (DVFS), can not be applied ahead of time for energy saving.

To provide an efficient early exit for resource-constrained computing platforms, we propose *Predictive Exit*: prediction of fine-grain early exits in computation- and energy-efficient inference. To our best knowledge, Predictive Exit is the first work to forecast the exiting point and adjust the computing

configuration (i.e. DVFS) along with the inference proceeds, which achieves significant computation and energy savings. The contributions of this work are three-fold.

- A novel *Predictive Exit framework* for deep learning networks is introduced, which is shown in Fig. 1 (c). Fine-grained exiting layers sharing the same topology are *potentially* placed after each network layer. A *selective* execution of the exiting layers will capture the opportunities to exit on time and reduce computation and energy costs.
- A low-cost *prediction engine* is proposed to predict the point where the network will exit, execute the exiting layer selectively at run-time based on the prediction, and optionally adjust the computing configurations for energy saving during each inference based on DVFS.
- Extensive experiments with floating-point (FP32) and fixed-point quantization (INT8) operations are conducted on both server and embedded GPUs with energy measurements to demonstrate that Predictive Exit achieves significant computation cost reduction and energy saving with the state-of-the-art early exit strategies.

Background and Related Work

Latency, Power, and Energy

In many applications, such as robots and self-driving cars, deep learning networks are released and executed periodically. The time interval between the release times of two consecutive inferences is called the inference period T. The actual execution time of each inference is called latency l, which rarely exceeds the period T. Given a computing platform and a neural network with fixed computation, the inference latency l is inversely proportional to the operating frequency of the digital logic inside the computing platform, which is called the computing frequency f. Following the basic rules of semiconductors, a higher f indicates faster computing but requires a linearly increased processor supply voltage V.

When a task is executed, the power consumption in computing, called *active power* P_{active}, is the summation of dynamic, static, and constant power. When the computing platform is idle, the power consumption, also called *idle power* P_{idle}, mainly comprises static and constant power. The energy E of a computing platform includes both active and idle power integrated with time,

$$\begin{aligned} E &= \int_0^{T_{active}} P_{active} dt + \int_0^{T_{idle}} P_{idle} dt \\ &= \int_0^{T_{active}} (P_{Dynamic}+P_{Static}+P_{const}) dt + \int_0^{T_{idle}} (P_{Static}+P_{const}) dt \end{aligned} \tag{1}$$

T_{active} is the time spent on executing the task, and T_{idle} is the time duration when the computing platform is idle, which is the time interval between finishing this inference and starting the next inference, denoted by $T_{idle} = T - T_{active}$. The *dynamic power consumption* $P_{Dynamic}$ originates from the activity of logic gates inside a processor, which can be derived by

$$P_{Dynamic} = CV^2 f, \tag{2}$$

where C is the capacitance of switched logic gates. The *static power dissipation* P_{Static} originates from transistor static current (including leakage and short circuit current) when the processor is powered on, which is described by

$$P_{Static} = V N_{tr} I_{static}. \tag{3}$$

N_{tr} is the number of logic gates, and I_{static} is the normalized static current for each logic gate. The *constant power* P_{const} is the power consumption by the auxiliary devices of a computing platform, like board fans and peripheral circuitry.

Early Exit with Dynamic Voltage Frequency Scaling

DVFS is a power management technique in computer architecture whereby f and V of a microprocessor can be automatically adjusted on the fly depending on the actual needs to conserve power and energy cost of the computing platform. Given the same inference period T, combining early exit and DVFS could effectively reduce the cost of computation resources for a deep learning network.

Originally, the deep learning networks are executed by running all network layers at the default high frequency and voltage without early exit and DVFS. We assume the inference latency (execution time) l is equal to inference period T. In this case, the energy consumption within T is $E = \int_0^T (CV_{high}^2 f_{high} + V_{high} N_{tr} I_{static} + P_{const}) dt$. As the deployment of early exists, network layers located ahead of the early exit run at the default high frequency and voltage. After the network exits at T_{exit}, computing platform can reduce the voltage and frequency to the lowest level by DVFS until time reaches T (Tambe et al. 2021). Therefore, the energy consumption can be described by $E = \int_0^{T_{exit}} (CV_{high}^2 f_{high} + V_{high} N_{tr} I_{static} + P_{const}) dt + \int_{T_{exit}}^T (V_{low} N_{tr} I_{static} + P_{const}) dt$.

Since the processor voltage-frequency pair setting is cubic to $P_{Dynamic}$ but linear to P_{Static} and latency l according to Eqs. (2) and (3), it is highly demanded to further reduce the cost of $P_{Dynamic}$. An intuition to achieve this is to adjust the voltage and frequency to proper middle-level and run the network until it exits at time T. However, how to predict where the network will exit and adjust the voltage-frequency pair during the inference is not trivial. In this work, we propose a solution to adjust the voltage and frequency to the proper "middle-level" at run-time to dynamically reduce computation and energy consumptions.

Related Work

Early exits have attracted tons of attention as an important branch for dynamic inference in the past few years. Since it was studied in (Panda, Sengupta, and Roy 2016; Teerapittayanon, McDanel, and Kung 2016), many approaches have been proposed to improve the accuracy or reduce the computation cost of exit decisions. (Wang et al. 2019) offered a dynamic loss-weight adjustment early-exit strategy for ResNets together with the hardware-friendly architecture of early-exit branches. To determine the placement of exiting layers, (Kaya, Hong, and Dumitras 2019) explored the distribution of computation cost; (Bonato and Bouganis 2021) classified the exit point for each inference object; (Panda, Sengupta, and Roy 2017) and (Baccarelli et al. 2020) calculated the benefit and additional computation and energy cost from adding this exiting layer. To further balance the accuracy, inference time, and energy tradeoffs, (Wołczyk et al. 2021) proposed a Zero Time Waste (ZTW) method approach by adding direct connections between exiting layers

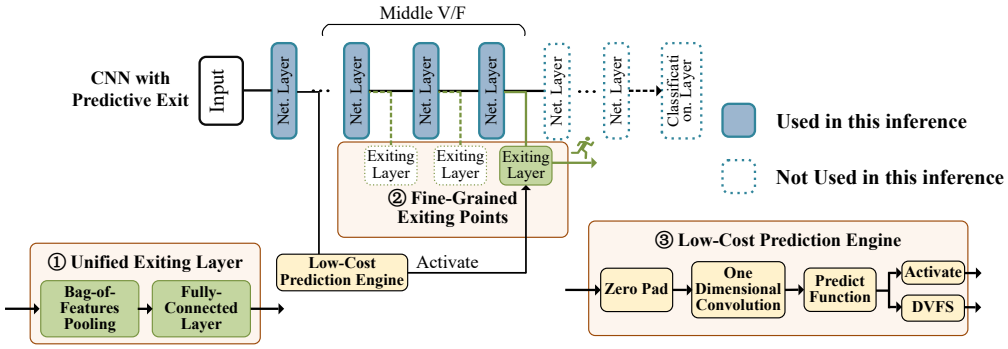


Figure 2: The framework of Predictive Exit.

and combining previous outputs in an ensemble-like manner, and (Ghodrati, Bejnordi, and Habibian 2021) proposed an on-the-fly supervision training mechanism to reach a dynamic tradeoff between accuracy and power.

Together with the proceeding of theoretical models, early exits have started being deployed in hardware and IoT systems (Odema, Rashid, and Faruque 2021; Samikwa, Di Maio, and Braun 2022). (Li et al. 2020) and (Zeng et al. 2019) adopted device-edge synergy optimized by DNN partitioning and DNN right-sizing through the early exit for on-demand DNN collaborative inference. The strategy is feasible and effective for low-latency edge intelligence. (Xu et al. 2018) presented a compressed CeNN framework optimized by five different incremental quantization methods and early exit. FPGA implementation shows that two optimizations achieve 7.8x and 8.3x speedup, respectively, while almost no performance loss. Meanwhile, (Laskaridis, Kouris, and Lane 2021) and (Scardapane et al. 2020) further provided a thorough overview of the current architecture, state-of-art methods, and future challenges of early-exit networks.

The Framework of Predictive Exit

Targeting computation- and energy-efficient inference with early exits, the Predictive Exit introduces the following schemes into one continuous spectrum, as shown in Fig. 2.

Unified exiting layer: The neural networks run on the graphics processing units (GPU), tensor processing units (TPU), or application-specific integrated circuits (ASIC) on local or edge devices. To release the burden of executing versatile computation, we make the exiting layers used in the network share the same topology.

Fine-grained exiting points: Exiting layers are potentially placed after each convolution layer to fully utilize the early exit opportunities. To avoid the computation overhead from running every exiting layer, only the layer at the predicted exiting point will be executed.

Low-cost prediction engine: The core of the framework is the low-cost prediction engine, which forecasts the possible exiting point and activates the early exiting layer. Based on the remaining computation workload before the expected exiting point, the proper computing frequency and voltage will be selected to run the inference and to further save energy.

Unified Exiting Layer

In this work, the design of exiting layer is based on the existing work (Passalis et al. 2020). All the exiting layers share the same topology. It contains a Bag-of-Features (BoF) pooling layer and a Fully-Connected (FC) layer. Let y_i be the intermediate results at the i^{th} layer and N_c be the number of object classes in the dataset. The BoF pooling layer functions as a feature aggregation to extract from y_i. In the BoF pooling, a set of feature vectors called codebook are used to describe y_i. The weight of each codebook is generated by measuring the similarity between the codebook and y_i. Since the size of codebook weight is usually larger than N_c, the FC layer further works as a classifier that adjusts the result of BoF pooling to $\mathbb{R}^{N_C}$, which estimates the final output of the exiting layer. Details can be found in (Passalis et al. 2020). The function of the exiting layer is denoted as

$$g^{(i)}_{\mathbf{W}_i}(y_i) = g^{(i)}_{\mathbf{W}_i}(f_{\mathbf{W}}(\mathbf{x}, i)) \in \mathbb{R}^{N_C}, \tag{4}$$

where $f_{\mathbf{W}}(\mathbf{x}, i)$ stands for the intermediate result after the calculation of 1^{st} to i^{th} layer.

After we've trained the early exiting layers, the average feature weight is first calculated according to Eq. (5), which are parameters for the exit decision.

$$\mu_i = \frac{1}{N}\frac{1}{N_C}\sum_{k=1}^{N_c}\sum_{j=1}^{N}\left[g^{(i)}_{\mathbf{W}_i}(y_i)\right]_k, \tag{5}$$

where N is the size of the training dataset. During the inference, at each early exiting layer, the weight ratio α_W is calculated as the maximum feature weight over the μ_i multiplied by a hyperparameter β specified by the user. Once α_W in Eq.(6) is larger than 1, the inference is terminated and the result at this early exiting layer is used as the final result.

$$\alpha_W = \frac{\left[g^{(i)}_{\mathbf{W}_i}(y_j)\right]_k}{\beta\mu_i}, k = \arg\max g^{(i)}_{\mathbf{W}_i}(y_j) \tag{6}$$

The key idea behind such implementation is that the larger the maximum feature weight is, the more confident the classification is regarded. The parameter β plays the role of striking a balance between the accuracy and acceleration. If a more accurate result is expected, β should be higher and vice versa.

Fine-Grained Exiting Points

In the classic early exit design, a limited number of exiting layers are settled in a fixed location and distance, usually $\frac{1}{3}$

to $\frac{1}{4}$, of the neural networks. For example, in the inference model proposed in (Teerapittayanon, McDanel, and Kung 2017), the two early exits are placed at roughly $\frac{1}{3}$ and $\frac{2}{3}$ of the network. If the first exiting layer fails, the inference must run through the following network layers until it reaches the next exiting layer to check whether it can exit. A more sensible way proposed by (Kaya, Hong, and Dumitras 2019) is to settle the exiting layers based on the percentage of the total cost after a rough estimation of the computational costs of each layer. However, their "fixed-place" design ignores the possibility of exiting the layers between the two early exits. It is nontrivial to choose where and how to place early exits to satisfy both computation time and inference accuracy.

In this work, we proposed a fine-grained exiting points design, that is, placing potential exiting layers on every network layer since the starting layer L_0. L_0 is a hyperparameter specified by the user, working as an adjuster of computation time and inference accuracy. During the inference process, the trial of early exit begins on L_0. If the trial succeeds, the early exit is triggered. If fails, instead of running through every exiting layer from L_0+1, the position of the next trial is $L_1 = L_0 + \textsc{Predict}\left(g^{(L_0)}_{\mathbf{W}_{L_0}}\left(y_{L_0}\right), \beta\right)$, where PREDICT is the function of low-cost prediction engine to forecast the next location to exit, which will be introduced in the following subsection. Under the circumstances that early exit does not succeed on L_i, we will start another prediction based on the exiting layer result of L_i. That is

$$L_{i+1} = L_i + \textsc{Predict}\left(g^{(L_i)}_{\mathbf{W}_{L_i}}\left(y_{L_i}\right), \beta\right). \tag{7}$$

Therefore, the location expectation of exit L_e can be expressed as

$$L_e = L_0 + \sum_{j=1}^{\xi} p_{f_j} \times \textsc{Predict}\left(g^{(L_j)}_{\mathbf{W}_{L_j}}\left(y_{L_j}\right), \beta\right), \tag{8}$$

where L_0 is the location of first trial, ξ stands for the times of predictions, and p_{f_j} is possibility of prediction failure.

Low-Cost Prediction Engine

Prediction of Exiting Points The PREDICT function starts to forecast the exiting point since the L_0 layer. To fully use the information from the intermediate result during the inference process, the key function of PREDICT is realized by one dimensional convolution on the intermediate result y_{L_0} and the exiting layer result based on it $g^{(L_0)}_{\mathbf{W}_{L_0}}\left(y_{L_0}\right)$ at L_0, which is summarized in Algorithm 1. To perform the convolution, we first extend the intermediate result $g^{(L_0)}_{\mathbf{W}_{L_0}}\left(y_{L_0}\right)$ from $\mathbb{R}^{N_c}$ to $\mathbb{R}^{K+N_c-1}$ by zero padding, which allow for more space for the filter to cover the intermediate result,

$$\begin{aligned} J_{L_0} &= \textsc{ZeroPad}_{(K+N_c-1)}\left(g^{(L_0)}_{\mathbf{W}_{L_0}}\left(y_{L_0}\right)\right) \\ &= \begin{cases} g^{(L_0)}_{\mathbf{W}_{L_0}}\left(y_{L_0}\right)[i], & \frac{K-1}{2} \le i \le \frac{K-3}{2} + N_c \\ 0, & \text{otherwise} \end{cases}. \end{aligned} \tag{9}$$

Afterwards, a vector $h \in \mathbb{R}^K$ is generated (with all of 1 in our work) as the filter of the convolution. Based on J_{L_0} and h, a new set of feature weight $G^{L_0+1}_{L_0} \in \mathbb{R}^{N_c}$ is generated through one dimensional convolution, which estimates the results of exiting layer placed at L_0+1,

$$G^{L_0+1}_{L_0}[i] = \sum_{k=0}^{K-1} h[k] \times J_{L_0}[i - \frac{k-1}{2}]. \tag{10}$$

By recursively replacing the intermediate result $g^{(L_0)}_{\mathbf{W}_{L_0}}\left(y_{L_0}\right)$ in Eq. (9) with $G^{L_0+1}_{L_0}[i]$ and repeating the computation of Eq. (9) and Eq. (10), the predicted results of exiting layer placed at L_0+2 can be obtained and noted as $G^{L_0+2}_{L_0+1}$. Following above steps, the predicted results of any exiting layer placed after L_0 can be calculated.

With the predicted results of exiting layer placed at any layer after L_0, we will have PREDICT function as follows:

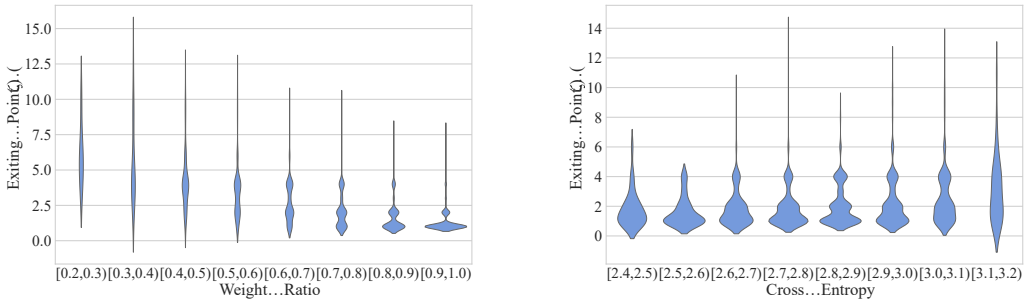
$$\textsc{Predict}\left(g^{(L_0)}_{\mathbf{W}_{L_0}}\left(y_{L_0}\right), \beta\right) = \zeta, \tag{11}$$

where $\zeta \in \mathbb{Z}$ represents the predicted exiting point after L_0, which is the smallest number that satisfies

$$\frac{\left[G^{L_0+\zeta}_{L_0+\zeta-1}\right]_k}{\beta \mu_{L_0+\zeta}} > 1, k = \arg\max G^{L_0+\zeta}_{L_0}, \tag{12}$$

where $\frac{\left[G^{L_0+\zeta}_{L_0+\zeta-1}\right]_k}{\beta \mu_{L_0+\zeta}}$ indicates the predicted exiting confidence at layer $L_0+\zeta$. Therefore, $L_0+\zeta$ is the predicted exiting point from PREDICT. In case that ζ cannot be found, a hyperparameter τ is further introduced, that is $\zeta \in [1, \tau]$. τ should be no more than $L_{\text{total}} - L_0$ where L_{total} stands for the number of layers of the model, meaning that prediction result beyond the last layer is forbidden. If no integer in $[1, \tau]$ satisfies Eq. (12), we assume that $\zeta = \tau$. The prediction engine is simple enough to avoid adding notable computation costs to the network. The time complexity of Algorithm 1 is $O(N_c \times K)$. Therefore, the computation cost of the prediction engine is 10^{-7} to 10^{-9} of the entire networks' computation cost. Meanwhile, hyperparameters are introduced in the prediction. L_0, β, and τ can be tuned by advanced users, balancing the prediction accuracy and computation cost for different scenarios.

Discussion of Weight Ratio- and Cross Entropy-Based Prediction We discuss the prediction for early exits using α_W and cross entropy $\mathcal{J}(\mathrm{p}, \mathrm{q})$, which will be defined in Eq. (15), since these two parameters are the indicators of confidence in early exits. More precisely, a higher weight ratio or a lower cross entropy indicates more confidence in exiting. As shown in Fig. 3, we record the values of weight ratio and cross entropy (x axis) at the prediction starting point L_0 and the actual exiting layer (y axis) in the inference. The test case results of the Resnet-34 model and the SVHN dataset are presented in the figure. However, on the weight ratios ranging from [0.2 0.6), the exiting points are distributed between the 1st and 12th layers. These spreading trends are also observed when the weight ratio lies in [0.6 0.8) and [0.8 1.0). Similarly, the exiting points are also widely distributed on the same cross entropy. Therefore, it is hard to predict the exiting point purely based on weight ratio and cross-entropy, even they are the indicators of confidence.



(a) Weight ratio α_W (b) Cross entropy $\mathcal{J}(\mathrm{p},\mathrm{q})$

Figure 3: Exit prediction with weight ratio α_W and cross entropy $\mathcal{J}(\mathrm{p},\mathrm{q})$.

Algorithm 1: : Low-cost Prediction Engine

Input: Hyperparameter starting layer L_0, β and τ;
Exiting layer result at L_0; $g^{(L_0)}_{\mathbf{W}_{L_0}}(y_{L_0})$;
Output: Predicted exiting point: $L_0+\zeta$;
//The prediction procedure:
for $L_0+\zeta = L_0+1, ..., L_{total}$ **do**
//Zero padding from $\mathbb{R}^{N_c}$ *into* $\mathbb{R}^{K+N_c-1}$
if $L_0+\zeta = L_0+1$ **then**
$J_{L_0+\zeta} = \textsc{ZeroPad}\left(g^{(L_0)}_{\mathbf{W}_{L_0}}(y_{L_0})\right)$;
else
$J_{L_0+\zeta} = \textsc{ZeroPad}\left(G^{L_0+\zeta}_{L_0+\zeta-1}(y_{L_0+\zeta-1})\right)$;
//One dimensional convolution
$G^{L_0+\zeta}_{L_0+\zeta-1}[i] = \sum_{k=0}^{K-1} h[k] \times J_{L_0+\zeta}[i-\frac{k-1}{2}]$;
//Check prediction confidence
if $\frac{\left[G^{L_0+\zeta}_{L_0+\zeta-1}\right]_k}{\beta\mu_{L_0+\zeta}} > 1, k = \arg\max G^{L_0+\zeta}_{L_0}$ **then**
return ζ;
$\zeta = \tau$, where τ is a predefined hyperparameter.
return ζ;

DVFS for Predictive Exit

Based on the predicted exiting point from the prediction engine, the voltage-frequency pair is adjusted to proper "middle levels" and run the network until it exits at time T. Given a network with L_{total} layers and a prediction engine that starts the prediction at L_0 and predicts the exiting point is $L_0+\zeta$, the prediction engine will reduce the computing frequency (and its corresponding voltage) to

$$f_{middle} = \frac{L_0+\zeta-L_0}{L_{total}-L_0} f_{high}, \tag{13}$$

where f_{middle} is the lowest frequency that can finish the inference by time T and f_{high} is the default high frequency of the computing platform. Therefore, the energy consumption used in this inference would be

$$\begin{aligned} E = &\int_0^{T_{L_0}} (CV^2_{high} f_{high} + V_{high} N_{tr} I_{static} + P_{const}) dt \\ + &\int_{T_{L_0}}^{T} (CV^2_{middle} f_{middle} + V_{middle} N_{tr} I_{static} + P_{const}) dt, \end{aligned} \tag{14}$$

where T_{L_0} is the time of finishing the prediction starting layer L_0 and its previous network layers. As the selection of computing frequency is based on the predicted remaining workloads in the inference, the prediction engine can make the inference finish by time T based on correct predictions. Since the processor voltage and frequency are linear scales to the computing performance (inversely proportional to inference latency) but the cubic scale and linear scale to dynamic power and static power, the predictive exit will effectively reduce energy consumption.

Training the Network with Predictive Exit

The learning objective of Predictive Exit is to keep the network inference accuracy given the predictive exit functions. Therefore, the objective function is the cross entropy loss

$$\mathcal{J}(\mathrm{p},\mathrm{q}) = -\sum_{i=1}^{N_C} \mathrm{p}(i) \log\left(\mathrm{q}(i)\right), \tag{15}$$

where $\mathrm{p}(i)$ and $\mathrm{q}(i)$ are the true class distribution and the predicted class distribution for each object.

Our model is trained through batch gradient descent. The data are fed into the model to optimize the parameter. Let N_{train} be the size of the training set $\mathcal{X}$, and a target vector $r \in \mathbb{R}^{N_c}$ be the correct result. At first, the model is trained without any exiting layer by the following equation

$$W' = W - \eta \cdot \nabla_W \mathcal{J}\left(f_w\left(\mathcal{X}, L_{total}\right), r\right), \tag{16}$$

where η is the learning rate. In the circumstance that the accuracy fails to meet the requirement, η can be adjusted. Afterward, with W of the original model fixed and armed with exiting layers, the model is trained again to optimize the parameters in the BoF pooling and FC layer,

$$W' = W - \eta \cdot \nabla_W \mathcal{J}\left(g^{(i)}_{\mathbf{W}_i}(y_i), r\right), i \in [1, ..., L_{\text{total}}]. \tag{17}$$

Although the training procedure requires two steps, the advantage brought by the inference process far outweighs this disadvantage, especially when only the inference is deployed on the resource-constrained platforms.

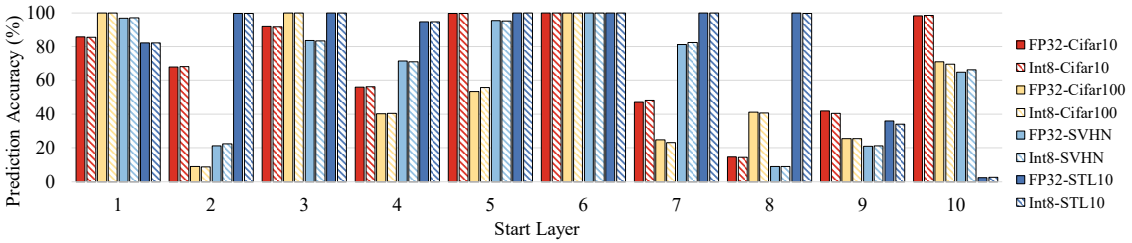
Evaluation

Experimental Setup

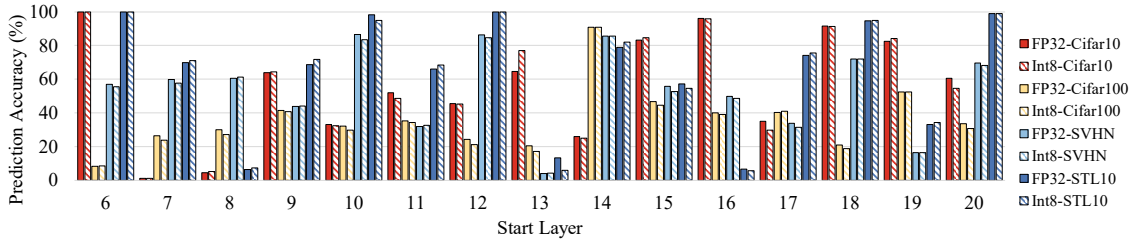
We evaluate the Predictive Exit using **VGG-19** and **ResNets-34** as the backbone models on the commonly used CIFAR-10,

Model	VGG-19				ResNet-34			
Approach	Classic CNN	Hierarchical	Placement	Predictive Exit	Classic CNN	Hierarchical	Placement	Predictive Exit
FP32 Cifar10	89% | 100%	88% | 59.8%	88% | 46.9%	**88% | 46.2%**	90% | 100%	89% | 28.7%	89% | 12.1%	**89% | 7.7%**
FP32 Cifar100	64% | 100%	62% | 86.6%	62% | 80.6%	**62% | 76.2%**	66% | 100%	63% | 36.6%	63% | 21.2%	**63% | 16.1%**
FP32 SVHN	89% | 100%	87% | 55.6%	87% | 56.5%	**87% | 46.8%**	91% | 100%	89% | 6.9%	89% | 5.4%	**89% | 3.8%**
FP32 STL10	75% | 100%	74% | 67.6%	74% | 64.2%	**74% | 51.5%**	76% | 100%	73% | 61.8%	73% | 17.8%	**73% | 5.7%**
Int8 Cifar10	88% | 100%	87% | 62.3%	87% | 46.4%	**87% | 46.1%**	85% | 100%	82% | 29.6%	82% | 14.9%	**82% | 8.4%**
Int8 Cifar100	64% | 100%	62% | 95.0%	62% | 79.5%	**62% | 76.4%**	62% | 100%	60% | 44.4%	60% | 22.3%	**60% | 18.5%**
Int8 SVHN	89% | 100%	86% | 57.2%	86% | 47.4%	**86% | 46.7%**	88% | 100%	85% | 7.1%	85% | 5.6%	**85% | 4.0%**
Int8 STL10	74% | 100%	74% | 66.9%	74% | 64.3%	**74% | 51.5%**	71% | 100%	70% | 78.1%	70% | 17.8%	**70% | 5.8%**

Table 1: Inference accuracy and computation cost in the VGG-19 and ResNet-34 network



(a) VGG-19 network



(b) ResNet-34 network

Figure 4: Prediction accuracy with hyperparameter L_0.

CIFAR-100 (Krizhevsky, Hinton et al. 2009), SVHN (Netzer et al. 2011), and STL10 datasets (Coates, Ng, and Lee 2011). For the consideration of low computation and energy applications, both floating-point (FP32) and fixed-point quantization (Int8) operations are tested. The training follows the procedure in Section 4. For the classic network and exiting layer, we set the learning rates as 0.00025/0.001 and train them with 100 iterations. We compare the proposed **Predictive Exit (Predictive.)** with different early exit approaches:

- **Classic CNN (Classic.)** adopts VGG-19 (Simonyan and Zisserman 2014) or ResNets-34 (He et al. 2016);
- **Hierarchical (Hier.)** Early Exit, proposed in (Passalis et al. 2020), which is specifically designed for CNNs by placing exits at roughly $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$ of the network;
- **Placement (Place.)** Early Exit, proposed in (Panda, Sengupta, and Roy 2017) and improved in (Baccarelli et al. 2020) with tunable hyperparameters, which identifies the best place to attach an early exit by considering the benefit and extra computation and energy cost.

The computation and energy cost are evaluated on both server GPU Quadro GV100 and embedded GPU Jetson TX2 with both offline and online measured power consumption. The overhead from executing the prediction engine and adjusting the processor voltage and frequency are included in the tests.

Inference Accuracy and Computation Cost

The inference accuracy and computation cost (the number of floating-point or integer operations normalized by that of classic CNN) of different inference approaches and datasets are shown in Table 1. On VGG-19 model, Predictive Exit achieves the same inference accuracy as other early exit approaches (which is 1% - 3% lower than Classic CNN). Because Predictive Exit will continue for the next prediction instead of forcing the inference to exit even if the prediction is wrong, no extra inference accuracy loss is brought. Compared with Hierarchical and Placement, Predictive Exit further reduces up to 12.8% of computation cost by leveraging the opportunities to exit earlier and avoiding frequent execution of the exiting layers. On the ResNet model, compared with Hierarchical and Placement, Predictive Exit reduces up to 12.1% of computation cost without extra accuracy loss.

For the training process, Hierarchical structure demands

Frequency (GHz)	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95	1.00	1.05	1.10	1.15	1.20	1.25	1.30	1.35	1.40	1.45
Active Power (W)	59.2	67.4	73.5	81.1	85.3	90.4	97.5	104.9	112.8	119.5	130.1	139.5	148.9	161.1	170.2	180.6	199.1	218.5
Idle Power (W)	35.1	35.9	37.0	37.8	38.9	39.8	41.1	41.3	43.8	44.2	45.0	45.8	46.5	47.8	49.3	50.5	52.3	55.1

Table 2: Measured power consumption of the NVIDIA Quadro GV100 GPU system

Model	VGG-19				ResNet-34			
Approach	Classic CNN	Hierarchical	Placement	Predictive Exit	Classic CNN	Hierarchical	Placement	Predictive Exit
FP32 Cifar10	100%	62.2%	47.9%	**27.1%**	100%	58.6%	59.3%	**30.3%**
FP32 Cifar100	100%	85.1%	72.8%	**44.1%**	100%	69.9%	67.3%	**39.9%**
FP32 SVHN	100%	58.9%	54.0%	**27.1%**	100%	46.8%	49.1%	**27.1%**
FP32 STL10	100%	65.7%	64.7%	**27.1%**	100%	73.6%	64.5%	**27.2%**
Int8 Cifar10	100%	76.8%	47.0%	**27.1%**	100%	58.8%	60.7%	**30.6%**
Int8 Cifar100	100%	85.2%	73.1%	**45.1%**	100%	75.0%	68.1%	**41.9%**
Int8 SVHN	100%	54.0%	47.7%	**27.1%**	100%	46.9%	49.1%	**27.1%**
Int8 STL10	100%	66.0%	64.7%	**27.1%**	100%	85.3%	64.5%	**27.2%**

Table 3: Normalized energy consumption

additional training of the exiting layers placed in specified positions of the network. While both Placement and Predictive design need to train all exiting layers that settle after each layer of the network, Placement further needs one more step to determine placement indexes by an exhaustive search.

Where to Start Prediction: Hyperparameter L_0

The prediction engine will start the exiting prediction since the starting layer L_0. This section quantitatively compares the prediction accuracy of different L_0 during the training process. To let the prediction engine cover a wide range of network layers, L_0 should be the first few layers in the networks. Therefore, in the VGG-19 network, we test L_0 placed at the 1st-10th layers, and in the ResNet-34 network, we test L_0 placed at the 6th-20th layers. Fig. 4(a) and Fig. 4(b) present the prediction accuracy under FP32 and INT8 operations across different datasets. The prediction accuracy indicates the percentage of successful exiting at the first predicted exiting point. In the VGG-19 network, if L_0 is the first network layer, the prediction can achieve over 81% accuracy across all tested datasets and operations. Most notably, if L_0 is the 6th network layer, the prediction can achieve over 99.8% accuracy. Therefore, L_0 for VGG-19 network and test datasets will be the 6th network layer. Similarly, for the ResNet-34 network, the desired L_0 is the 6th, 14th, 10th, and 6th layers for CIFAR-10, CIFAR-100, SVHN, and STL 10.

Energy Benefit by DVFS

To illustrate the potential energy benefit of Predictive Exit, we first calculate the energy consumption with offline measured active and idle power of NVIDIA Quadro GV100 GPU at different frequency-voltage pairs (Kandiah et al. 2021) (shown in Table 2). In a network with L_{total} layers, when the Predictive Exit predicts the network will exit at $L_0 + \zeta$ layers, the processor will select the lowest candidate frequency in Table 2 that is higher than or equal to $\frac{L_0+\zeta-L_0}{L_{total}-L_0} * 1.45GHz$ to execute this network. We count the inference workload (including the exiting layer) under each frequency-voltage pair.

Approach	Classic.	Hier.	Place.	Predictive.
Cifar10	1195.6	451.9	500.2	**384.8**
Cifar100	1198.2	914.8	732.2	**496.6**
SVHN	1209.4	873.9	521.6	**492.2**

Table 4: Energy consumption on Jetson TX2 (Joule)

Based on the power consumption in each frequency-voltage pair, we compare the energy consumption and normalize it to the energy consumption used by the Classic CNN model in Table 3. Unsurprisingly, early exit achieves tremendous energy savings compared with Classic CNN. Predictive Exit further reduces the energy consumption (compared with the best cases of Hierarchical and Placement) by 19.9%-37.6% on the VGG-19 network and 19.7%-37.3% on the ResNet-34 network. Meanwhile, the proposed method is tested on the embedded Nvidia Jetson TX2, whose power consumption is measured at run time with Tektronix mdo32 oscilloscope and TCP2020 current probe. Since its memory limits loading **ResNets-34** models with early exits, power consumption on **VGG-19** are tested and its results are summarized in Table 4. The Predictive Exit reduces up to 45.7% and 32.2% of energy compared with Hierarchical and Placement early exits.

Conclusion

The proposed Predictive Exit can accurately predict where the network will exit as a computation- and energy-efficient inference technique. By activating the exiting layer at the expected exiting point, the Predictive Exit reduces the network computation costs by exiting on time without running every pre-placed exiting layer. The Predictive Exit significantly reduces the energy consumption used in inference by selecting proper computing configurations in the CNN inference process. Since the Predictive Exit is a plug-in for existing networks without model modification, it has the potential to be applied to general learning networks.

Acknowledgements

Published in AAAI-23. This research project is supported by NSFC 62202287, NSFC 62103268, and Shanghai Chenguang Program 21CGA11. An Zou is the corresponding author.

References

Baccarelli, E.; Scardapane, S.; Scarpiniti, M.; Momenzadeh, A.; and Uncini, A. 2020. Optimized training and scalable implementation of Conditional Deep Neural Networks with early exits for Fog-supported IoT applications. *Information Sciences*, 521: 107–143.

Bonato, V.; and Bouganis, C.-S. 2021. Class-specific early exit design methodology for convolutional neural networks. *Applied Soft Computing*, 107.

Coates, A.; Ng, A.; and Lee, H. 2011. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 215–223. JMLR Workshop and Conference Proceedings.

Figurnov, M.; Collins, M. D.; Zhu, Y.; Zhang, L.; Huang, J.; Vetrov, D.; and Salakhutdinov, R. 2017. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1039–1048.

Ghodrati, A.; Bejnordi, B. E.; and Habibian, A. 2021. FrameExit: Conditional Early Exiting for Efficient Video Recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 15608–15618.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

Kandiah, V.; Peverelle, S.; Khairy, M.; Pan, J.; Manjunath, A.; Rogers, T. G.; Aamodt, T. M.; and Hardavellas, N. 2021. AccelWattch: A Power Modeling Framework for Modern GPUs. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 738–753.

Kaya, Y.; Hong, S.; and Dumitras, T. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*. PMLR.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*.

Laskaridis, S.; Kouris, A.; and Lane, N. D. 2021. Adaptive Inference through Early-Exit Networks: Design, Challenges and Directions. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, EMDL'21, 1–6. New York, NY, USA: Association for Computing Machinery. ISBN 9781450385978.

Li, E.; Zeng, L.; Zhou, Z.; and Chen, X. 2020. Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. *IEEE Transactions on Wireless Communications*, 19(1): 447–457.

Netzer, Y.; Wang, T.; Coates, A.; Bissacco, A.; Wu, B.; and Ng, A. Y. 2011. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

Odema, M.; Rashid, N.; and Faruque, M. A. A. 2021. EExNAS: Early-Exit Neural Architecture Search Solutions for Low-Power Wearable Devices. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design*.

Panda, P.; Sengupta, A.; and Roy, K. 2016. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 475–480. IEEE.

Panda, P.; Sengupta, A.; and Roy, K. 2017. Energy-efficient and improved image recognition with conditional deep learning. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3): 1–21.

Passalis, N.; Raitoharju, J.; Tefas, A.; and Gabbouj, M. 2020. Efficient adaptive inference for deep convolutional neural networks using hierarchical early exits. *Pattern Recognition*, 105: 107346.

Samikwa, E.; Di Maio, A.; and Braun, T. 2022. Adaptive Early Exit of Computation for Energy-Efficient and Low-Latency Machine Learning over IoT Networks. In *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, 200–206.

Scardapane, S.; Scarpiniti, M.; Baccarelli, E.; and Uncini, A. 2020. Why should we add early exits to neural networks? *Cognitive Computation*, 12(5): 954–966.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Tambe, T.; Hooper, C.; Pentecost, L.; Jia, T.; Yang, E.-Y.; Donato, M.; Sanh, V.; Whatmough, P.; Rush, A. M.; Brooks, D.; et al. 2021. Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 830–844.

Teerapittayanon, S.; McDanel, B.; and Kung, H. 2017. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 328–339.

Teerapittayanon, S.; McDanel, B.; and Kung, H.-T. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE.

Wang, M.; Mo, J.; Lin, J.; Wang, Z.; and Du, L. 2019. DynExit: A Dynamic Early-Exit Strategy for Deep Residual Networks. In *IEEE International Workshop on Signal Processing Systems (SiPS)*, 178–183.

Wang, X.; Yu, F.; Dou, Z.-Y.; Darrell, T.; and Gonzalez, J. E. 2018. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Woł czyk, M.; Wójcik, B.; Bał azy, K.; Podolak, I. T.; Tabor, J.; Śmieja, M.; and Trzcinski, T. 2021. Zero Time Waste: Recycling Predictions in Early Exit Neural Networks. In *Advances in Neural Information Processing Systems*, volume 34, 2516–2528.

Xu, X.; Lu, Q.; Wang, T.; Hu, Y.; Zhuo, C.; Liu, J.; and Shi, Y. 2018. Efficient Hardware Implementation of Cellular Neural

Networks with Incremental Quantization and Early Exit. *J. Emerg. Technol. Comput. Syst.*, 14(4).

Zeng, L.; Li, E.; Zhou, Z.; and Chen, X. 2019. Boomerang: On-Demand Cooperative Deep Neural Network Inference for Edge Intelligence on the Industrial Internet of Things. *IEEE Network*, 33(5): 96–103.