Early-Exit Deep Neural Network - A Comprehensive Survey

HASEENA RAHMATH P, Bennett University, Noida, India
VISHAL SRIVASTAVA, Computer Science and Engineering, Motilal Nehru National Institute of Technology, Allahabad, India
KULDEEP CHAURASIA, Bennett University, Noida, India
ROBERTO G. PACHECO, Universidade Federal Fluminense, Rio das Ostras, Brazil
RODRIGO S. COUTO, GTA/PEE-COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil

Deep neural networks (DNNs) typically have a single exit point that makes predictions by running the entire stack of neural layers. Since not all inputs require the same amount of computation to reach a confident prediction, recent research has focused on incorporating multiple "exits" into the conventional DNN architecture. Early-exit DNNs are multi-exit neural networks that attach many side branches to the conventional DNN, enabling inference to stop early at intermediate points. This approach offers several advantages, including speeding up the inference process, mitigating the vanishing gradients problems, reducing overfitting and overthinking tendencies. It also supports DNN partitioning across devices and is ideal for multi-tier computation platforms such as edge computing. This article decomposes the early-exit DNN architecture and reviews the recent advances in the field. The study explores its benefits, designs, training strategies, and adaptive inference mechanisms. Various design challenges, application scenarios, and future directions are also extensively discussed.

CCS Concepts: • **Computing methodologies** → **Neural networks**;

Additional Key Words and Phrases: Deep neural networks, early-exit neural network, multi-exit neural network, edge intelligence, inference acceleration.

ACM Reference Format:
Haseena Rahmath P, Vishal Srivastava, Kuldeep Chaurasia, Roberto G. Pacheco, and Rodrigo S. Couto. 2024. Early-exit Deep Neural Network—A Comprehensive Survey. *ACM Comput. Surv.* 57, 3, Article 75 (November 2024), 37 pages. https://doi.org/10.1145/3698767

Authors' Contact Information: Haseena Rahmath P, Bennett University, Noida, Uttar Pradesh, India; e-mail: haseenarahmath@gmail.com; Vishal Srivastava, Computer Science and Engineering, Motilal Nehru National Institute of Technology, Allahabad, Uttar Pradesh, India; e-mail: vishalsri@mnnit.ac.in; Kuldeep Chaurasia, Bennett University, Noida, Uttar Pradesh, India; e-mail: kuldeep@bennett.edu.in; Roberto G. Pacheco, Universidade Federal Fluminense, Rio das Ostras, RJ, Brazil; e-mail: robertopacheco@id.uff.br; Rodrigo S. Couto, GTA/PEE-COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brazil; e-mail: rodrigo@gta.ufrj.br.

ACM 0360-0300/2024/11-ART75
https://doi.org/10.1145/3698767

1 Introduction

Deep neural networks (**DNNs**) have monopolized the world of Artificial Intelligence, due to their ability to extract features and produce high-quality decisions from complex data. They have been extensively applied across various domains, including **natural language processing** (**NLP**) [63, 67, 88], **computer vision** (**CV**) [43, 65, 116, 129], and wireless networking [56, 109]. The great success of DNNs in image classification, image recognition, machine translation, video gaming, and many others can be attributed to their complex and deep non-linear layer architectures. Advances in automatic learning techniques coupled with hardware [42] and optimization-level techniques are also playing an important role in the success of DNN. These models are typically designed as a sequence of interconnected neural layers stacked on top of one another. Achieving high accuracy often requires deeper models. Popular models such as AlexNet [65], VGGNet [116], ResNet [43], and DenseNet [51] achieve high performance with deep architectures that have millions of parameters. However, these models demand significant processing power and energy, introducing latency [64]. Deploying these models on resource-constrained devices for low-latency applications can be challenging [100]. For example, DNN-based speech or face recognition applications [18, 47] and real-time navigation assistance [102] provide precise predictive accuracy but with an extremely high requirement of computational resources that can't be affordable by most mobile or IoT devices available on the market today.

Researchers have proposed various innovative techniques to reduce DNN complexity and enable deployment in low-resource environments, such as mobile devices. Model compression techniques, such as model pruning [36], quantization [55], and **knowledge distillation** (**KD**) [48], aim at compressing large models. Additionally, hardware and software optimization methods are introduced to reduce execution time. Lightweight models, such as MnasNet [120] and MobileNetV2 [158], require fewer computational resources but suffer from significant accuracy loss. For instance, compared to ResNet-152 [43], MobileNetV2 [158] reports an accuracy loss of up to 6.4% for image classification on the ImageNet dataset [93]. The tradeoff between accuracy and computational resource requirements can be mitigated by offloading data to powerful remote cloud servers [90]. In this approach, a mobile device gathers input (e.g., an image) and sends it to the cloud server for processing by the DNN model. Cloud servers, equipped with the necessary computational resources like GPUs, can run complex DNN models efficiently [33]. However, this cloud-based approach introduces latency in data transfer and raises privacy concerns, which are unacceptable for most latency-sensitive mobile applications.

A promising new computing paradigm, called edge computing [13], allocates computational resources closer to end devices. This approach has been successfully applied in various domains, such as smart home systems [154], autonomous vehicles [80], and healthcare applications [144]. In edge computing, most DNN computations are performed at edge devices near the end devices, rather than at centralized cloud servers. Edge computing involves offloading data partially or completely to an edge device situated close to the end device or mobile device. While this method addresses many limitations of cloud-based solutions, the performance of edge-based approaches depends heavily on the bandwidth availability between the end device and the edge device or server [151]. Deploying DNNs on edge devices with variable bandwidth and offloading constraints, while maintaining accuracy, is challenging [34]. Research has extensively investigated training and executing resource-demanding DNNs on resource-limited edge devices. Recent contributions have proposed various DNN model partitioning techniques that split the training and execution of the DNN model among devices, edge servers, and cloud servers based on context parameters.

Belilovsky et al. [6], Marquez et al. [91], Lee et al. [74], Wang et al. [131], and Huang et al. [50] demonstrated that small and lightweight neural network architectures can recognize and classify most input samples from complex datasets with acceptable accuracy. For instance, a DNN model

with a single convolutional (conv) layer can accurately classify 30% of the images in the complex ImageNet dataset [76]. The overthinking problem occurs when a DNN makes correct predictions before reaching its final layer, but by the final layer, those correct predictions become misclassifications. This issue can be mitigated using a smaller DNN architecture [2, 62]. Additionally, the deep and sequential nature of neural networks, combined with the gradient locking problem [129], makes the DNN structure challenging to parallelize.

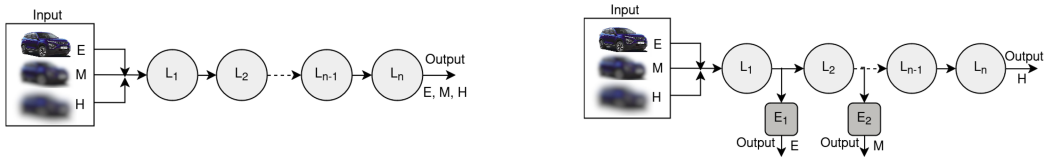
To address the previous problems, the early-exit DNN was introduced. This approach modifies the conventional DNN architecture by adding multiple side branches into its intermediate layers, accelerating inference since many inputs can be classified at these branches, as shown in [6, 74, 91]. The novel concept of a early-exit DNN model, first proposed by Teerapittayanon et al. [123] and named BranchyNet, modifies the conventional DNN structure by adding multiple side branches. In BranchyNet, inference results can be deduced from any side branch based on confidence criteria, unlike conventional DNN models where input samples propagate through all layers to the output layer regardless of the input sample characteristics. The early-exit DNN is ideal for edge computing as it enhances performance and reduces computational costs by allowing early-exits at intermediate layers. It also offers flexibility in partitioning the DNN model via side branches and supports multi-tier DNN execution.

This article explores recent advances in early-exit DNNs and their applications. Systematic state-of-the-art surveys of early-exit DNNs are very few. Scardapane et al. [112] introduced multi-exit neural networks and discussed various techniques and ideas for designing, training, and deploying them in time-constrained environments. Matsubara et al. [93] provided a short overview of early-exit DNNs and their applications. Laskaridis et al. [69] briefly described the architecture and execution of early-exit networks and their applicability in adaptive inferences. Han et al. [37] reviewed dynamic neural networks and provided abstract details on early-exit DNNs. The previous surveys either mentioned early-exiting in the context of dynamic inference networks [37], combined it with split computing and offloading [93], or provided a very brief description of early-exit DNN, its design, training, and deployment [69]. [112] provided a coherent introduction to early-exit DNN but only addressed a few training and deployment strategies. Furthermore, many studies have been conducted since the publication of that survey. This survey provides a comprehensive introduction to early-exit DNNs and a systematic analysis of recent techniques and strategies for designing, training, and deploying early-exit DNNs. This article focuses on the design constraints of early-exit DNN and divides the architecture into different parts, reviewing recent developments in each. It also discusses the key benefits and open challenges of early-exit DNN and its applicability across various domains.

1.1 The Main Contributions of This Work

Early-exit DNNs have recently sparked significant research interest, yet only a few surveys have reviewed recent advances in this area. Researchers have used various terms to describe similar concepts, such as cascaded networks, multi-classifiers, dynamic networks, adaptive networks, early termination neural networks, and multi-exit neural networks. Despite the progress, several research issues still need to be resolved to advance early-exit DNNs. A structured introduction and systematic survey of the state-of-the-art can increase research interest, provide insights, and attract new researchers to the field. In this context, this article makes the following contributions:

(1) **Unified Literature:** This article unifies the fragmented literature under different names and provides a coherent, structured introduction to early-exit DNNs.
(2) **Benefits Comparison:** It comprehensively compares the advantages of early-exit DNNs over conventional DNNs.



(a) Conventional DNN (b) Early-exit DNN

Fig. 1. Inference process—(a) Conventional DNN: always delivers results from the output layer, irrespective of input sample characteristics; (b) Early-exit DNN: delivers results via intermediate exit branches based on input complexity (E, M, and H represent samples with easy, medium, and hard complexity, respectively; L1 to Ln are backbone DNN layers, E1 and E2 are early-exit branches).

(3) **Architecture Description:** It describes the early-exit DNN architecture and discusses major design constraints.
(4) **Implementation Review:** It reviews several early-exit DNN implementations in-depth and discusses various training and deployment strategies.
(5) **Research Challenges and Applications:** It emphasizes critical research challenges and explores various application possibilities.

The remainder of this article is organized as follows: Section 2 discusses the motivation and benefits of early-exit DNNs. Section 3 analyses the literature that conveys similar concepts of an early-exit DNN. Section 4 describes the architecture of the early-exit DNN, while Section 5 explains the various training strategies used for training the early-exit DNN. Section 6 covers the deployment and inference processes of the early-exit DNN. Sections 7 and 8 explore various applications and research challenges of early-exit DNN, respectively. Finally, Section 9 concludes this work.

2 Motivation and Benefits of Early-exit DNNs

This section discusses the motivation and benefits of early-exit DNNs over conventional DNNs. Early-exit DNNs address several challenges faced by conventional DNNs, such as vanishing gradient problems [1], overfitting issues [130], and difficulties in parallelized implementation [118]. These advantages are detailed below.

2.1 Speed Up the DNN Inference

Designing and deploying a DNN on resource-constrained devices such as mobile and IoT devices to enable faster inference with acceptable accuracy is a critical task. Many studies have attempted to reduce model complexity by employing techniques like model compression [36], and quantization [55] to remove redundant parameters from the model. However, even with these techniques, conventional DNNs require each input to be processed by all neural layers. Figure 1(a) illustrates a conventional DNN model inference process, where each input is processed layer-by-layer, from the input layer to the output layer. Not all inputs are equally difficult to classify, and samples may require different amounts of computation to achieve a confident prediction [123]. Early-exit DNNs provide a solution by allowing certain inputs to exit the model earlier, speeding up the inference process while maintaining accuracy.

Early-exit neural networks insert multiple side branches into the conventional DNN architecture. The inference process in such DNNs can be stopped early at a side branch if it meets predetermined confidence criteria. Figure 1(b) illustrates the inference process of early-exit DNNs, where each side branch estimates the prediction confidence for each input. This confidence is related to the difficulty of classifying the input. Easier inputs can be classified at earlier side branches using

features extracted by the initial neural layers, while more complex inputs require processing until the output layer. BranchyNet [123] evaluated the performance of early-exit DNNs using popular architectures like LeNet [72], AlexNet [65], and ResNet [43], revealing a 2× to 6× acceleration in inference speed on both CPU and GPU. Additionally, Pacheco et al. [98] and Rahmath et al. [40] analyzed the performance of early-exit DNNs on MobileNetV2 [158], demonstrating that a significant portion of images were classified early through side branches. Experimental results confirmed the superior performance of early-exit DNNs over conventional DNNs, resulting in substantial reductions in inference time.

2.2 An Alternate Solution to the Overfitting Challenge of DNNs

Overfitting is a common issue in deep learning, occurring when a DNN model performs well on training data but fails to generalize to unseen data [4]. This happens because the model memorizes the training data parameters instead of learning to generalize. Various solutions have been proposed to mitigate overfitting, such as data augmentation [1], different regularization schemes [4, 130], dropouts [118], and early stopping [11]. Despite these efforts, models often still overfit. The early-exit DNN model offers a solution by treating the entire DNN structure as a single optimization problem and jointly optimizing the weighted losses of all side branches [123]. Each side branch acts as a regularizer for the other branches, which helps prevent overfitting. This regularization effect has been highlighted in studies such as [123], where the interaction between side branches aids in improving generalization and model robustness. By sharing features and learning representations at different levels of abstraction, side branches enhance overall model regularization.

2.3 Rectify Overthinking Issue of DNNs

While DNNs can perform prediction from raw input data, most real-world inputs are not inherently complex and do not require the execution of the entire depth of the DNN architecture. When a DNN can predict such inputs before its final layer, full-stack execution becomes computationally wasteful. Sometimes, a DNN makes correct predictions before its final layer, but misclassifies them by the final layer (overthinking issue) [62]. Studies by Kaya et al. [62] and Srivastava et al. [134] showed that smaller DNNs correctly predicted certain inputs, while more complex DNNs delivered incorrect predictions. This destructive effect of overthinking could be eliminated if a conventional DNN could deliver output from intermediate layers. Early-exit DNNs are a natural choice in this case, as they can deliver results from any of the intermediate layers through the added exit points. By terminating early for simpler inputs, early-exit DNNs mitigate the wasteful and destructive effects of overthinking, ensuring computational efficiency and improved accuracy.

2.4 Mitigate the Vanishing Gradients Problem of DNNs

The vanishing gradient problem is an unstable behavior exhibited by DNNs during training, where the model fails to propagate back the gradient information effectively, hindering weight optimization. This issue arises because, during each iteration, the gradient becomes progressively smaller until it vanishes, causing the training to halt completely. Various solutions have been proposed to address this problem, including normalized network initialization [29, 73], highway network [119], intermediate normalization layers [54], and residual network [43]. These techniques often involve skip connections between network layers. Early-exit DNNs provide an additional gradient update signal through their exit points during backpropagation, helping prevent gradient shrinking issues. Furthermore, early-exit DNNs add a discriminative effect to the lower layers [123], enhancing the network's ability to learn effectively. In this manner, early-exit DNNs mitigate the vanishing gradient problem, ensuring more stable and efficient training.

Table 1. Categorization of Related Literature Conveying Similar Concepts of Early-exit DNN

Category	References
Cascade networks	[8, 9, 32, 91, 117, 134]
Multi-classifiers/Internal classifiers	[62, 77, 111]
Dynamic/Adaptive inference	[16, 22, 23, 31, 84, 115, 127, 136, 145]
Adaptive networks	[14, 77, 78]
Layer skipping networks	[94, 127, 135, 138]
Dynamic wide-networks/DNN	[22, 89, 95, 97]
Deeply-supervised models	[74, 131]
Neural Trees	[122, 148]
Conditional deep learning network	[103, 104]
Multi-branch networks	[16, 78, 95, 111]
Multi-exit networks	[15, 21, 35, 52, 53, 57, 60, 75, 106, 125, 133, 137, 150]
Early-exit/termination neural networks	[8, 9, 17, 23, 24, 28, 30, 39, 41, 58, 59, 61, 70, 71, 75, 79, 82, 87, 98, 100, 101, 103, 110, 114, 121, 123, 124, 132, 139, 140, 141, 142, 143, 146, 149, 155, 156]

2.5 Provide a Multi-tiered Platform for Training and Deploying DNNs

An important motivation for early-exit DNNs is their suitability for distributed environments. The sequential execution of conventional DNNs, due to their interconnected layers, makes parallelization challenging. Early-exit DNNs, however, can be easily partitioned via their side branches, facilitating distributed training and deployment. This makes them ideal for distributed computing environments, such as edge computing applications, where computations are offloaded to nearby edge or cloud servers [88]. By enabling parts of the model to be processed on different devices or servers, early-exit DNNs enhance computational efficiency and reduce latency, making them well-suited for real-time applications and resource-constrained environments.

3 Related Works

Recent research has seen a proliferation of studies focusing on early-exit DNNs, often presented under diverse names. As illustrated in Table 1, these studies can be categorized into distinct groups: cascaded networks, multi-classifiers or internal classifiers, dynamic or adaptive inferences, dynamic or adaptive networks, layer skipping networks, deeply supervised models, neural trees, conditional deep learning networks, multi-branch or multi-exit networks, and early-exit or early termination neural networks. This section provides a brief overview of each category.

Cascaded networks consist of multiple DNNs, referred to as component DNNs, arranged sequentially for classification. These networks operate in a cascade fashion, stopping computation when prediction confidence exceeds a specified threshold [91]. Similar to early-exit DNNs, cascaded networks employ thresholds and softmax outputs to determine termination points [8]. As input complexity rises, additional component DNNs are incorporated into the cascade. Bolukbasi et al. [9] introduced an alternative cascading architecture where computation can pause after each conv layer. This design includes conv layers branching to classifiers, with a decision function guiding the sequence of DNN components in execution. Guan et al. [32] utilized a cascade of classifiers with a policy module to select the optimal classifier for each input instance. The policy module integrates three **fully connected** (**fc**) layers, two ReLU units, and a softmax function to compute the termination probability. Wang et al. [134] introduced the "**I Don't Know**" (**IDK**) cascade framework, enhancing base models with additional classifiers that generate an IDK class for uncertain predictions. Upon predicting an IDK class, the subsequent model in the cascade is activated. Inference concludes upon predicting a real class or reaching the end of the cascade. Notably,

IDK classifiers operate independently of base models, evaluating their confidence scores to assess prediction uncertainty. Soldaini et al. [117] proposed a **cascade transformer** (**CT**) for document ranking, leveraging classifiers at various encoding stages to prune candidates in batches.

Dynamic or adaptive inference is a contemporary strategy to reduce DNN computational demands for resource-constrained devices like IoT and mobile devices. These approaches utilize multi-branch neural networks to dynamically select computational paths for each input instance [16, 22, 23, 31, 84, 115, 127, 136, 145]. Similar to early-exit DNNs, they employ multi-classifier and internal classifier networks attached to intermediate layers of conventional DNNs to facilitate early prediction [62, 77]. Multi-branch networks [16, 78, 95, 111] use multiple kernels to extract features from input samples and combine the results to produce the final outcome. Dynamic DNNs, or dynamic wide networks, employ semantic specialization techniques to enhance computational efficiency during execution [22, 89, 95, 97]. They partition classes into visually similar groups and assign them to specific branches, enabling these branches to discriminate among the classes. This reduces computational costs by focusing on a subset of the architecture for inference on each input. HydraNet [95] employs a gating mechanism to specialize branches in feature extraction for visually similar classes, improving component selection during execution. Li et al. [77] and Shuang Li et al. [78] developed adaptive neural networks and **dynamic domain adaptation** (**DDA**) frameworks, respectively, optimizing performance and enabling knowledge transfer across domains. Fang et al. [22] introduced FlexDNN, adapting to varying model complexities in on-device video analytics to improve deployment efficiency.

In layer-skipping networks, the inference process stops early by selectively skipping intermediate layer computations. A gating system analyzes input characteristics to decide if executing a layer is necessary. Veit et al. [127] proposed a convolutional network with an adaptive inference graph to dynamically determine the next computing layer. SkipNet [135] is a modified residual network featuring a gating mechanism to selectively skip layers. BlockDrop [138] dynamically drops and selects layers in ResNet for each instance. **Conditional Deep Learning** (**CDL**) activates layers based on input complexity [103]. CDL networks consist of sequential stages with a linear classifier and an activation module. Each stage predicts class labels and confidence scores, with the activation module deciding whether to proceed to the next layer or terminate the process with the predicted label [104]. Neural trees integrate DNN parameter optimization with decision tree conditional computation. Tanno et al. [122] introduced **Adaptive Neural Trees** (**ANTs**), combining architecture learning, hierarchical representation, and conditional computation. ANTs use edges, leaf nodes, and rooting functions from DNN representation learning, selecting execution layers conditionally via stochastic routing. ANT's growth depends on data complexity and availability, performing conditional computation by activating a subset of model parameters based on selected root-to-leaf paths. Yu et al. [148] employed a Soft Conditional Gate, which uses pooling and an MLP layer for input-conditioned routing to select the optimal path during inference. The activation probabilities of dynamic nodes are compared with a predefined threshold to decide whether to terminate the inference process. **Deeply supervised nets** (**DSN**) attach multiple classifiers to intermediate layers, using objective functions to enhance interpretability and reduce prediction error [74]. MSDNet [50] is a DSN with intermediate exit classifiers in ResNet and DenseNet backbones, improving efficiency and performance in resource-aware environments. Multi-exit or early-exit networks generate predictions using intermediate results from attached side exit branches.

4 Early-Exit DNN—Architecture

This section provides an overview of the early-exit DNN architecture. Section 4.1 outlines the structure of a conventional DNN, while Section 4.2 introduces the early-exit DNN architecture,

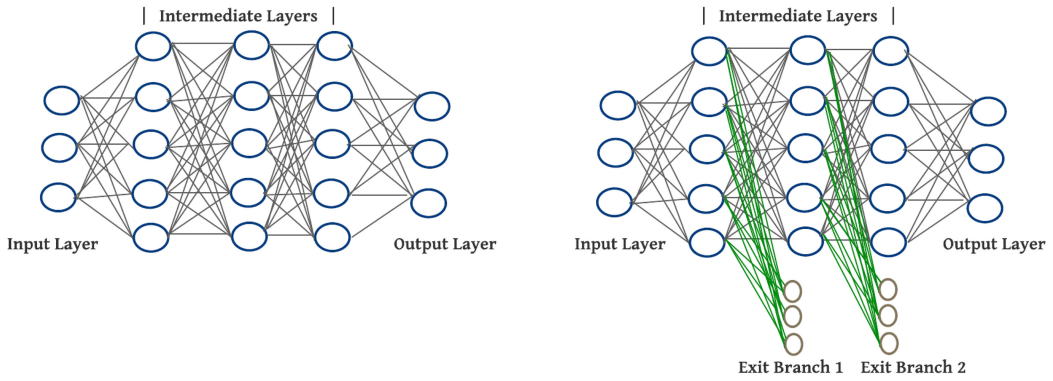
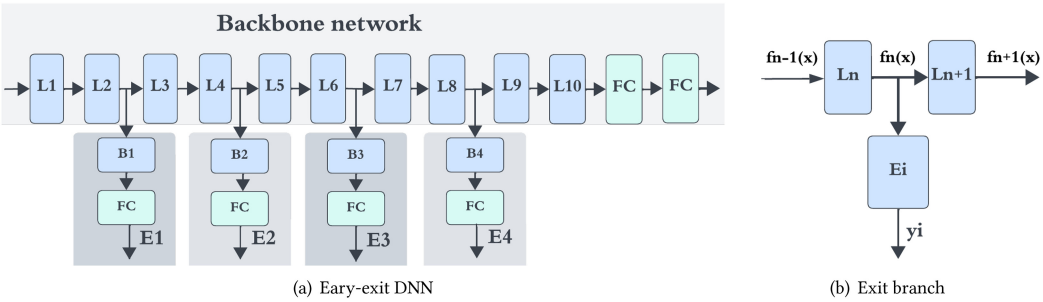


Fig. 2. Illustration of conventional DNN (left) and early-exit DNN (right) with two side branches.



(a) Eary-exit DNN (b) Exit branch

Fig. 3. (a) Early-exit DNN with four intermediate branches. (b) Illustration of a side branch.

which is the central focus of this article. Figure 2 provides a visual comparison of both models. Additionally, Section 4.3 elaborates on the design constraints specific to early-exit DNN models.

4.1 Conventional DNN Architecture

The fundamental components of a neural network are neurons, which receive inputs $(x_1, x_2, \ldots, x_n)$ along with a bias (b). These inputs are processed $(\sum)$, passed through a non-linear activation function (σ), and yield an output (y) using Equation (1) [44]:

$$y = \sigma\left(\sum_{i=1}^{N} w_i x_i + b\right) = \sigma(W^T X + b), \tag{1}$$

where, $\mathbf{X}$ represents the input vector, $[x_1, x_2, \ldots, x_n]$, and $\mathbf{W}^T$ denotes the transpose of the weight vector $\mathbf{W}$, $[w_1, w_2, \ldots, w_n]$.

A neural network comprises neurons organized in layers, interconnected either fully or convolutionally. These layers include input, hidden (intermediate), and output layers. The number of hidden layers determines the network's depth and complexity. Figure 3 illustrates the layer-wise architecture of a neural network. Input layers receive multi-dimensional input samples, which propagate through hidden layers where various nonlinear and activation functions are applied. The final output or prediction, y, is obtained from the output layer through Equation (2):

$$y = F_n(x) \odot F_{n-1}(x) \odot \cdots \odot F_j(x) \odot F_i(x) \odot \cdots \odot F_2(x) \odot F_1(x), \tag{2}$$

where $F_j(x)$ is the output from the j^{th} layer after applying the operation, F_i, to the output produced by layer i. The F_i may be a single operation or a sequence of operations. The composite operator $\odot$ in $(F_j(x) \odot F_i(x))$ indicates the composition of operations such as convolution, batch normalization, dropout, self-attention, and so on, applied to the output of the i^{th} layer before being processed by the layer j [44, 112].

A DNN can be realized by connecting every neuron in one layer to every neuron in the next layer. This type of neural network is called a feed-forward neural network. **Convolutional neural networks** (**CNNs**) are another realization of DNNs, where each neuron connects to nearby neurons and shares weights across groups of neuron connections. **Recurrent neural networks** (**RNNs**) are used for sequential or time-series data, where the results of the previous layer are passed as input to the current layer. To obtain a prediction, the conventional DNN model necessitates sequential execution across all layers—from the input layer through all hidden layers to the output layer. However, as discussed in Section 1, this exhaustive end-to-end processing may be unnecessary for many input samples in a dataset, thereby increasing computational overhead. The early-exit DNN model addresses this issue by incorporating side exit branches that allow inference to terminate at intermediate layers based on confidence levels. The following section provides an in-depth exploration of the early-exit DNN model, covering its foundational architecture and key design considerations essential to early-exit DNN modeling.

4.2 Early-exit DNN Architecture

A typical early-exit DNN architecture comprises a backbone DNN model, which is a conventional DNN such as AlexNet [65], ResNet [116], MobileNet [49], BERT [63], or DenseNet [51], to which side branches are attached. These side branches can be created using conv layers, fc layers, or a combination of both. Figure 3(a) illustrates an early-exit DNN with four side branches (E1 to E4) attached to intermediate layers of the conventional DNN. During inference, input is processed layer-by-layer until reaching a side branch, where a prediction is made and confidence is assessed. If the confidence is sufficient, the inference process stops and returns the prediction; otherwise, it continues until reaching the next side branch. If any side branch can provide a sufficiently confident prediction, the input is processed until the final output layer, which always returns a prediction. Figure 3(b) shows the graphical representation of a side branch E_i in which $f_{n-1}(x)$ is the output generated by (n-1)$^{\text{th}}$ layer and $f_n(x)$ is the output generated by the n$^{\text{th}}$ layer. The output from the n$^{\text{th}}$ layer can propagate in two different paths: one to the side branch E_i and the other to the layer (n+1)$^{\text{th}}$ of the backbone DNN. If the branch, E_i, provides a sufficiently confident prediction, the inference terminates and this side branch E_i classifies the input $\mathbf{X}$. The structure of E_i could be a smaller neural network or a complex structure consisting of various neural layers such as conv layers, fc layers, softmax layers, and is explained in detail in Section 4.3.1. Early termination via these exit points reduces computational cost and inference time by processing input data and making predictions sooner.

4.3 Design Constraints of Early-exit DNNs

Designing an effective early-exit DNN involves several key decisions: selecting the DNN backbone architecture, structuring the side branches, determining their placement within the backbone, and formulating an optimal early-exit policy. As a recent and evolving research area, early-exit DNNs lack standardized strategies for these aspects. The choice of early-exit policy significantly influences the number of early predictions at side branches, impacting overall performance. Various studies have employed different methods to determine prediction confidence. This section delves deeply into previous works, exploring how side branches were constructed, the strategies adopted

Table 2. Branching Structure Employed to Construct Early-exit DNN

Branching Structure	References
One fc layer	[14, 16, 17, 21, 41, 60, 66, 70, 77, 84, 89, 104, 106, 121, 125, 139, 140, 141, 145, 156]
Multiple fc layers	[155]
One conv layer and one fc layer	[45, 57, 82, 114, 136, 149]
Multiple conv layers with optional fc layers	[35, 53, 59, 61, 123, 146]
Pooling layer with an fc layer	[8, 17, 62, 71, 79, 98, 132, 136]
Convolutional, pooling, and fc layers	[58, 66, 75, 78, 81, 95, 111, 133, 135, 137, 143]
MLP followed by ReLu	[52]
Capsule networks	[87]
Learning the best branching structure	[150]

for placing them on the backbone model, and the policies formulated to determine confidence criteria at each exit point.

4.3.1 The Structure of the Branch. The neural components used to construct side branches in early-exit DNNs are summarized in Table 2. Most studies [14, 16, 17, 21, 66, 70, 77, 84, 89, 104, 106, 125, 139, 140, 141, 145, 156] [41, 60, 121] employed a single fc layer. Additionally, pooling layers were frequently utilized [8, 17, 62, 71, 79, 98, 132, 136]. For instance, DDI [136] used max-pooling, while average pooling was employed in [8, 17, 71, 79, 98]. SDN [62] adopted a mixed max-average pooling approach, dynamically adjusting the mixing ratio from data. Beretizshevsky and Even [8] constructed side branches with an average pooling layer, an fc layer, a batch normalization layer, a ReLU activation layer, and another fc layer followed by a softmax layer. Other studies such as [45, 57, 82, 114, 136, 149] used a single conv layer followed by an fc layer, while [35, 53, 59, 61, 123, 146] employed multiple conv layers. MSDNets [50] used two conv layers, an average pooling layer, and one linear layer for side branches. DEE [59] used five conv layers and three fc layers. Ilhan et al.'s EENet [52] employed a two-layer MLP with ReLU activation, while Li et al.'s EENet [79] used a pooling layer followed by an fc layer EENets [17] featured three types of side branches: plain (an fc layer), pool (an average pooling layer followed by an fc layer), and bnpool (batch normalization, ReLU activation, an average pooling layer, and an fc layer). LoCoExNet [58] employed pooling, followed by conv layers, batch normalization, and ReLU activation, then proceeded with an fc layer while OdeBERT[87] employed Capsule networks. Leco [150] applied DARTS [83] techniques to learn the optimal branching structure. While most studies maintained uniform structures across all side branches, studies such as BranchyNet [123], EdgeML [155], and DynamicSleepNet [137] varied their structures. BranchyNet employed one conv layer followed by one fc layer for the first branch and two conv layers followed by an fc layer for the subsequent branches. EdgeML used three fc layers for the first branch and two for the subsequent branches. DynamicSleepNet utilized two conv layers followed by global average pooling for the first exit and a transformer encoder block with layer normalization, two fc layers, and another layer normalization for another exit.

In summary, early-exit DNNs typically augment simple classifiers or small neural networks with one or two hidden layers to the backbone model. However, the high dimensionality of shallow conv layers in the backbone often requires more complex classifiers [62]. To streamline the architecture before classification, techniques such as average pooling, larger pooling windows, and bottleneck injection [93] should be considered.

4.3.2 Location and the Number of Branches. Table 3 details the number of side branches added to construct early-exit DNNs, while Table 4 outlines strategies for their placement on the backbone DNN. Studies such as [16, 45, 111, 145], placed branches after every conv layer. Conversely,

Table 3. Backbone DNN and Number of Side Branches Attached to Construct Early-exit DNN

Backbone	#Br.	References	Backbone	#Br.	References	Backbone	#Br.	References
VGG	5	[45]	AlexNet	4	[104]	AlexNet	5	[82, 111, 149]
VGG	9	[111]	AlexNet	2	[123, 125]	BERT	12	[84, 139, 140, 141, 156]
VGG	2	[60]	AlexNet	6	[59]	MSDNet	5	[52, 53, 57, 77, 78, 106]
LeNet	1	[123]	MSDNet	2	[146]	MSDNet	6	[77, 146]
LeNet	2	[104]	Faster-RCNN	4	[110]	ResNet	3	[8, 58, 125]
UNet	4	[142]	DenseNet	4	[52]	ResNet	5	[32, 57, 79]
LSTM	3	[71]	ResNet	11	[16, 132]	ResNet	6	[62, 75, 138]
BERT	4	[52]	ALBERT	12	[141, 156]	ResNet	9	[16, 59, 111]
ResNet	12	[16]	MobileNet	2	[125]	VGG	3	[58, 60, 125, 155]
ResNet	2	[123]	MobiletNet	5	[98]	VGG	6	[62, 79]
ResNet	4	[52]	MobiletNet	6	[75]	VGG	14	[14, 35]
MSDNet	11	[32]	MobileNet	3	[58, 100]	RoBERTa	12	[140, 141]

Table 4. Side Branch Placement Strategy Adopted by Previous Studies to Construct Early-exit DNN

Branch Placement Strategy	References
After every layer. (conv/res/transformer/bottleneck)	[15, 16, 21, 41, 61, 71, 84, 87, 103, 106, 110, 111, 114, 139, 140, 141, 142, 145, 156]
After specific layers. (conv/res/transformer/bottleneck)	[8, 14, 32, 45, 59, 60, 66, 77, 78, 82, 98, 99, 104, 123, 125, 132, 137, 146, 149, 155]
Based on some metrics	[22, 35, 58, 62, 70, 103, 104, 133]
Based on some gating function	[79, 94, 95]
Based on some user-defined pattern	[17, 52, 53, 136, 143]

other studies [14, 32, 59, 82, 104, 106, 123, 125, 155] positioned them after specific conv layers. DEE [59] placed branches after each residual (res) block, whereas [98] and [132] placed them after specific res blocks. The res blocks are neural blocks composed of several neural layers. SDN [62] and SPINN [70] calculated the relative computational cost of each layer, determining branch placement based on percentile scores relative to the total DNN cost. Studies [103] and [104] added side branches by evaluating the benefits of branch insertion at specific layers versus the penalties of incorrect classification. EENets [17] suggested the number of side branches should be proportional to the depth of the backbone model, as their computational cost is negligible. They recommended placement strategies based on the dataset and network capacity, distributing branches using quadratic, linear, golden ratio, and fine distributions. FlexDNN [22] placed branches based on the computation saved by early exits relative to the computation consumed, while HydraNets [95] employed a gating function to determine side branch placement. EPNet [16] attached branching classifiers and exit controllers after every conv layer except the last. In summary, early-exit DNNs utilize both uniform and non-uniform branch placement strategies. Uniform policies include Pareto, Golden Ratio, Fine, Linear, and Quadratic distributions. Proper branch placement is crucial for optimal performance, yet there is no consensus on the ideal strategy for determining the number and placement of side branches on the backbone model, making it a topic of ongoing research.

4.3.3 Early-exit Policies. After training an early-exit DNN, selecting appropriate early-exit policies is crucial for effectively utilizing the exit branches and processing each input instance efficiently. These policies determine whether the inference process terminates at an exit branch or proceeds to the next for further processing. Early-exit policies in the literature are broadly

Table 5. An Overview of the Training Strategies Adopted by the Existing Studies

Training Strategy	References
Joint	[8, 21, 35, 38, 53, 62, 70, 79, 82, 87, 89, 98, 99, 111, 117, 121, 123, 124, 132, 136, 139, 142, 145, 146, 149, 150, 155, 156]
Branch-wise	[5, 6, 7, 46, 68, 131, 142]
Separate	[15, 20, 71, 74, 128, 132]
Two-stage	[8, 9, 45, 66, 75, 76, 79, 103, 104, 110, 133, 140, 143]
KD-based	[41, 48, 77, 84, 92, 106, 137, 159]
Hybrid	[10, 32, 52, 100, 136, 141]

categorized into rule-based or static exit policies and learning-based or dynamic exit policies [69]. A detailed discussion of exit policies used across various studies can be found in Section 6.

5 Training Strategies for Early-exit DNN

Early-exit DNNs modify conventional DNN architectures by incorporating additional side branches, necessitating effective training strategies. As illustrated in Table 5, early-exit DNN training can be categorized into six main strategies. The first strategy is joint training, where the entire network, including the backbone DNN and all side branches, is trained as a unified optimization problem [123]. The second strategy is branch-wise training, which involves iteratively training each side branch alongside the preceding backbone layers [46]. In the third strategy, separate training, side branches are treated as standalone classifiers and trained independently [110]. The fourth strategy is two-stage training, where the backbone DNN is initially trained, and its parameters are frozen, followed by separate training of the side branches using these frozen weights [128]. The fifth strategy is KD-based training, where side branches act as student models learning from the final outputs of the backbone DNN, which serves as the teacher [48]. Finally, the sixth strategy is a hybrid approach that combines multiple training techniques to comprehensively train the early-exit DNN model [100]. The following section discusses these training strategies in detail.

5.1 Joint Training Strategy

As shown in Table 5, the joint training strategy is the most commonly used method for training early-exit DNNs. It is utilized in BranchyNet [123], SDN [62], DDI [136], DynExit [132], Boomerang [149], Edgent [82], SPINN [70], PABEE [156], EdgeML [155], AdaDet [146], HiDEC [53], DeeDiff [121], OdeBERT [87], and Leco [150]. Figure 4 provides a pictorial representation of this strategy, and the training procedure is summarized in Algorithm 1. In the figure, L_1 to L_n and FE are neural layers of the Backbone DNN, and E_1 to E_j are side branches. The losses generated by the corresponding exit branches are denoted as l_1 to l_j, while the loss generated at FE is l_n. The combined loss is represented as l. In this method, training the entire DNN, including additional side branches, is treated as a single optimization problem. It can be implemented in two ways: by combining the losses of each side branch [123], or by calculating the overall loss from the combined outputs of each side branch [111]. The training process consists of two passes: a forward pass, where training data is propagated through the backbone DNN and side branches, during which training parameters and errors are recorded; and a backward pass, where the recorded error is propagated back through the backbone model and weight parameters are adjusted using a gradient descent algorithm. BranchyNet [123] defined a softmax cross-entropy loss function for each side branch, as shown in Equation (3) and formulated a single optimization problem by combining all the losses, as shown in Equation (4).

$$\mathcal{L}(\hat{y}_i, y; \theta) = \frac{1}{|C|} \sum_{c \in C} y_c \log \hat{y}_{i,c}, \tag{3}$$

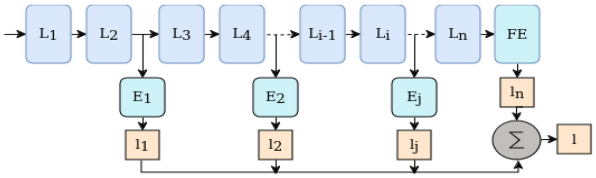


Fig. 4. A visual representation of the joint training strategy.

ALGORITHM 1: Joint Training

1. Make a feed-forward pass.
2. Train the entire model (backbone+side branches).
3. Track output at each side branch.
4. Calculate error rate.
5. Perform a backward pass.
6. Propagate error through the model.
7. Update weights using gradient descent.

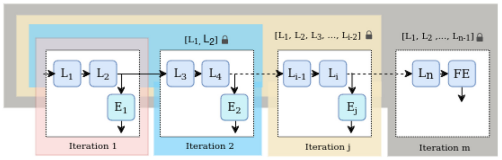


Fig. 5. A visual representation of the branch-wise training strategy.

ALGORITHM 2: Branch-wise Training

1. Train the first branch and freeze its parameters.
2. For each subsequent branch:
   a. Reuse the frozen parameters from the previous branch.
   b. Train the branch.
   c. Freeze its parameters after training.

where C is the set of all possible labels, y is the ground-truth label vector, $\hat{y}_{i,c} = softmax(z) = \frac{exp(z)}{\sum_{c \in C} exp(z_c)}$, where $z = f_i(x;\theta)$ is the output of the i^{th} exit branch and θ denotes the weight parameters of all neural layers between an entry point and an exit point.

$$\mathcal{L}_{\text{combined}}(\hat{y}, y; \theta) = \sum_{i=1}^{N} w_i \mathcal{L}(\hat{y}_i, y; \theta), \tag{4}$$

where N is the total number of exit points, including the main network's exit, and w_i is the weighting factor chosen across all side branches.

5.2 Branch-wise Training Strategy

In the branch-wise training strategy, each side branch is trained separately along with the preceding backbone DNN layers. This method follows the concept of forward-thinking, introduced by Hettinger et al. [46], which trains conventional DNNs in a layer-wise fashion without backpropagation. MSDNet [50] applied this strategy, where each iteration freezes the parameters of auxiliary classifiers and the preceding backbone DNN layers, as depicted in Figure 5. Initially, L_1, L_2, and E_1 are trained and their parameters are then frozen. Next, L_1, L_2, L_3, L_4, and E_2 are trained while leveraging the frozen parameters of L_1 and L_2. Similarly, subsequent branches are trained, leveraging previously frozen parameters. Algorithm 2 summarizes this training operation. Belilovsky et al. [5, 6] also investigated this training strategy. Larochelle et al. [68] explored an unsupervised layer-wise training strategy, while Bengio et al. [7] employed a supervised layer-wise training approach. Wang et al. [131] adopted a semi-supervised layer-wise training method. The primary advantage of this training strategy is its ability to mitigate issues like vanishing and exploding gradients, given that each branch is relatively lightweight.

5.3 Separate Training Strategy

In the separate training strategy, individual models are created for each side branch, starting from the input layer and encompassing all subsequent layers connected to the respective branch. Each side branch undergoes independent training [15, 20, 71, 74, 132]. Figure 6 illustrates this approach, where L_1, L_2, and E_1 form one independent model, and similarly, L_1, L_2, L_3, L_4, and E_2 form another,

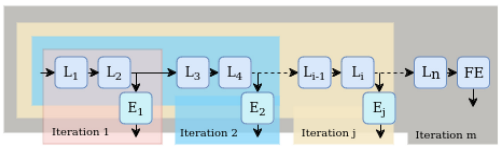


Fig. 6. A visual representation of the separate training strategy.

ALGORITHM 3: Separate Training

1. Construct exit models, each spanning from the input layer to its corresponding side branch.
2. For each exit model:
   a. Perform training.

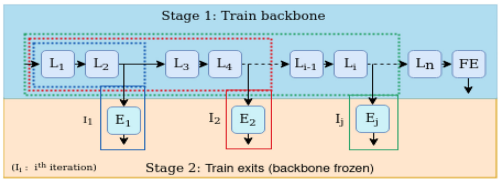


Fig. 7. A visual representation of the two-stage training strategy.

ALGORITHM 4: Two-stage Training

1. Train the backbone and freeze its parameters.
2. For each side branch:
   a. Reuse the frozen parameters of the backbone.
   b. Train the branch.

with each trained separately. This strategy may involve varying training objectives for each exit branch, typically focusing on minimizing standard loss functions like cross-entropy. Algorithm 3 provides a broad overview of this approach. Separate training offers substantial benefits when different exit points capture distinct features or levels of abstraction, enabling each branch to make a unique contribution to the overall architecture [128]. In this training, each side branch can be associated with different classifiers; for instance, DSN [74] uses SVM and softmax classifiers on its side branches.

5.4 Two-stage Training Strategy

In this training approach, the backbone DNN is initially trained without considering the side branches, and its parameters are then frozen. These frozen parameters are subsequently used to train each side branch separately. As shown in Figure 7, the backbone layers (L_1 to L_n and FE) are trained first. Their parameters are then used to train each branch independently: side exit branch E_1 is trained using the frozen parameters of L_1 and L_2. Similarly, side exit branch E_2 is trained using the frozen parameters of L_1, L_2, L_3, and L_4, and this process is repeated for all branches. The training process is summarized in Algorithm 4. This method is utilized in EPNet [16], CDL [104], PersEPhonEE [75], DeeBERT [140], PTEENet [66], TEEM [110], ENet [79], TLEE [133], and LGViT [143]. This approach is particularly useful when the backbone DNN is already in place and needs to be converted into a multi-exit architecture. Additionally, it enables rapid experimentation with quickly trainable and detachable side branches. Furthermore, this strategy allows for the incorporation of non-neural classifiers, such as decision trees or support vector machines, as auxiliary classifiers for intermediate predictions.

5.5 KD-based Training Strategy

The core principle of KD-based training involves transferring information asymmetrically from high-accuracy exit points to lower-accuracy ones. This approach assumes that the final output layer of the backbone DNN achieves the highest accuracy, with additional branches learning from it. Hence, the backbone DNN acts as the "teacher" model, guiding the augmented exit branches, which serve as "student" models. To facilitate this knowledge transfer, KD-based methods employ a distillation loss that measures the similarity between the predictions of the teacher and student models. This encourages the student models to replicate the soft targets or probability distributions generated by the teacher model. The distillation loss is typically formulated using

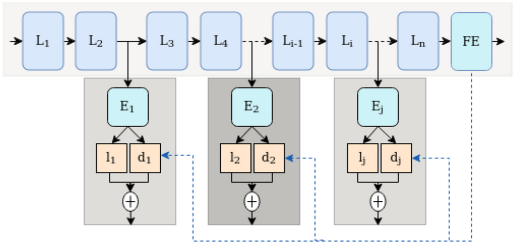


Fig. 8. A visual representation of the KD-based training strategy.

ALGORITHM 5: KD-based Training

1. Train the backbone (teacher) and track the output, P^t.
2. Train each side branch (Students):
   a. Make a feed forward pass.
   b. Calculate loss from student output, P_i^s.
   c. Calculate distillation loss using P^t and P_i^s.
   d. Calculate total loss, $\mathcal{L}i$, using steps b and c.
   e. Perform a backward pass and propagate $\mathcal{L}_i$.

the **Kullback-Leibler (KL)** divergence, as described in Equation (5):

$$d_i^{t,s} = \mathrm{KL}(P^t || P_i^s) = \sum_c P_c^t \log\left(\frac{P_c^t}{P_{i,c}^s}\right), \tag{5}$$

where i and c are indices representing branch and class, and the superscripts t and s denote the teacher and student models, respectively. $d_i^{t,s}$ denotes the distillation loss for branch i with respect to P^t, the soft targets provided by the teacher, and P_i^s, the soft targets generated by student i. Temperature scaling is applied to both P^t and P_i^s to soften their probability distributions. P_c^t signifies the predicted probability distribution by the teacher for class c, and $P_{i,c}^s$ is the corresponding distribution generated by student i for class c. The KL divergence penalizes the student model for deviating from the teacher's soft targets. One key advantage of the distillation loss is its applicability in scenarios lacking ground truth labels, which enables semi-supervised training of early-exit DNNs. In this method, the teacher model undergoes training first, and subsequently, its outputs are employed to train the student models. The overall loss at each branch i integrates both the cross-entropy loss l_i^s and the distillation loss $d_i^{t,s}$ weighted by a hyperparameter α, as shown in Equation (6):

$$\mathcal{L}_i = \alpha l_i^s + (1-\alpha) d_i^{t,s}. \tag{6}$$

This formulation ensures a balanced training approach, leveraging the complementary benefits of both loss components. Phuong and Lampert [106] and Li et al. [77] utilized a KD-based approach to train early-exit DNNs. The pictorial representation of this training strategy is illustrated in Figure 8 and the training procedure is summarized in Algorithm 5. GAML-BERT [159], FastBERT [84], and DE^3-BERT [41] introduced a mutual learning BERT framework that improves BERT's early-exit performance by using mutual KD techniques among exit points. DynamicSleepNet [137] also employed this training approach for training a transformer-based early-exit model for sleep stage classification.

5.6 Hybrid Training Strategy

This approach combines multiple training strategies to train early-exit DNNs. FreezeOut [10] integrates joint and branch-wise training strategies, while EENet [52] combines joint training with KD-based methods. Initially, the entire model is trained using joint training strategy then the side branches are fined further by using branch-wise or KD-based methods. Pacheco et al. [100] merge joint training with branch-wise training to create specialized expert side branches. They first train the entire DNN with side branches using joint training, followed by branch-wise training of each side branch with specialized datasets. BERxiT [141] fine-tunes its parameters by combining joint and separate training strategies. DDI [136] combines two-stage and joint training strategies. First, the backbone model is trained and its parameters are frozen. Only the gating networks are then

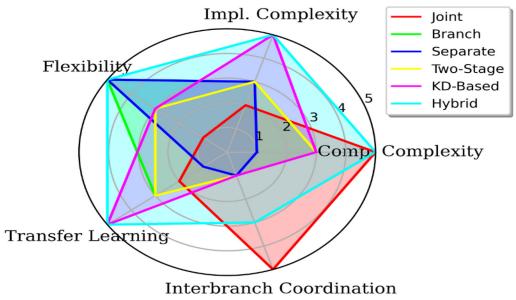


Fig. 9. Comparison of various training strategies.

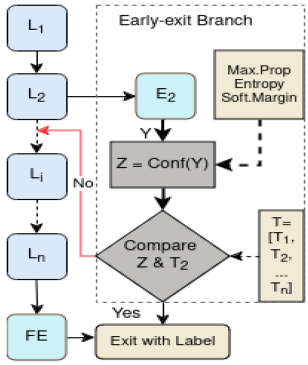


Fig. 10. Inference with static exit policy.

Table 6. Strengths and Weaknesses of Various Training Strategies

Strategy	Strengths	Weaknesses
Joint	Promotes holistic optimization, fosters implicit inter-branch learning, and simplifies implementation by training all branches simultaneously.	Imposes uniform task constraints across all branches, increases computational demands and training complexity, and struggles with scalability in large, deep networks.
Branch	Offers flexibility to independently optimize each branch, facilitates fine-tuning of specific branches, and reduces computational burden by focusing on targeted areas.	Time-consuming iterative training, limited adaptability due to frozen parameters, may not fully exploit inter-branch interactions.
Seprate	Enables targeted optimization and fine-tuning for each branch, simplifies debugging, and offers a flexible approach to training.	Lacks inter-branch learning, may result in inconsistent performance across branches, and increases complexity in integrating independently trained branches.
Two-stage	Enables efficient use of pre-trained models, reduces overall training time, and allows focused optimization of the backbone and early-exit branches.	Sequential training phases can be complex, limited interaction due to frozen parameters, potential suboptimal performance compared to joint optimization.
KD based	Facilitates knowledge transfer, improving side branch performance by emulating the teacher model, and allowing semi-supervised training	Relies on teacher model quality, Involves complex implementation, and increased computational overhead.
Hybrid	Combines strategies to maximize their strengths, optimizing both backbone and side branches effectively, while allowing for customization and leveraging synergies.	Complex implementation, requires careful balancing of strategies, and may increase computational resource and time requirements.

trained to achieve a zero skipping ratio. With this initialization, the backbone DNN's parameters are unfrozen and it is jointly trained with the gating networks to reach the targeted skipping ratio.

5.7 Comparison of Early-exit Training Strategies

Figure 9 evaluates early-exit training strategies based on computational complexity, implementation complexity, interbranch coordination, flexibility, and transfer learning potential. Table 6 outlines the strengths and weaknesses of each strategy, illustrating the tradeoffs involved in choosing a training method. Computational complexity refers to the amount of computational resources required to execute a training strategy, while implementation complexity assesses the difficulty involved in implementing the strategy. Interbranch coordination measures the effectiveness of communication and collaboration between branches during training. Flexibility indicates how well the training process can adapt to different needs and transfer learning potential evaluates the model's ability to transfer knowledge to new tasks or domains.

The joint training strategy excels in holistic optimization and inter-branch learning but requires high computational resources and training complexity. The branch-wise training strategy, which

Table 7. Early-exit Policies Employed in Various Literature

Exit Policy	Policy Type	References
Entropy Comparison	Static	[10, 17, 20, 22, 66, 71, 81, 84, 87, 89, 103, 110, 111, 121, 123, 124, 132, 137, 140, 146]
Exit Selection Controllers	Dynamic	[16, 32, 61, 141]
Softmax Probability Comparison	Static	[8, 15, 21, 27, 35, 41, 50, 57, 58, 62, 70, 75, 77, 97, 98, 100, 104, 113, 117, 128, 139, 143, 145, 148]
Multi-Armed Bandits	Dynamic	[59, 101]
Scoring Function Comparison	Static	[21, 23, 30, 31, 45, 52, 53, 94, 136, 142]
Objective Function	Dynamic	[9, 14]
Pseudo Labels and Prediction Mean	Dynamic	[78]
Soft Gating Mechanism	Dynamic	[95, 133]
Reinforcement Learning Technique	Dynamic	[32, 135, 138]
Intermediate Prediction Consistency	Dynamic	[141, 150, 156, 159]
Optimizer Algorithm	Dynamic	[149, 155]

trains branches sequentially, balances training complexity and inter-branch coordination, allowing each branch to build upon the previous one. This strategy's medium complexity and flexibility make it effective for optimal results. The separate training strategy offers maximum flexibility by training each branch independently but suffers from low inter-branch coordination, leading to potential inconsistencies. The two-stage training strategy enables rapid experimentation with quickly trainable and detachable side branches and allows for the incorporation of non-neural classifiers. KD-based training leverages pre-trained models to transfer knowledge to early-exit branches, enhancing performance through standard and distillation losses. However, it relies heavily on the backbone DNN's quality and involves medium complexity and moderate flexibility. The hybrid training strategy combines strengths from multiple strategies, offering flexibility and customization options but entails high complexity in design and implementation. This comparison highlights each early-exit training strategy's strengths and weaknesses, providing insights into the tradeoffs between complexity, coordination, and flexibility, crucial for selecting the most suitable approach for specific applications.

6 Inferences in Early-exit DNN

Once trained, an early-exit DNN can be deployed to perform inferences on target systems. The process begins with the DNN receiving an input and processing it layer-by-layer until it reaches the first side branch. At this point, the model employs an early-exit policy to determine whether the input should exit the model early or continue propagating through subsequent layers. These early-exit policies can be broadly classified into static or rule-based exit policies and dynamic or learnable exit policies. Table 7 summarizes the exit policies used in various studies, with detailed discussions in Sections 6.1 and 6.2.

6.1 Static Early-exit Policies

Static early-exit policies in early-exit DNNs rely on predefined rules to determine when processing can terminate early. If a side branch meets the exit criteria, the model produces output from that branch; otherwise, it advances to the next branch until reaching the final layer. Each input's progression is guided by confidence scores generated at the side branches. This strategy enables inference to stop early upon achieving a confident prediction or continue processing until reaching the final output layer, depending on the input's characteristics. Existing studies have developed

various rules and criteria for early termination, as illustrated in Table 7. Static exit policies measure prediction confidence using metrics like entropy, maximum softmax probability (max_prob), or custom scoring functions. A predefined threshold vector $\mathbf{T} = [T_1, T_2, \ldots, T_i, \ldots, T_{n-1}]$ is employed for comparison, where T_i represents the threshold value for the i^{th} side branch, and n is the total number of exit points. The input sample $\mathbf{x}$ is initially evaluated at the lowest exit point and propagates to subsequent higher side branches until reaching the final output layer, which always returns a result. Figure 10 illustrates the operation of static exit policies. At each side branch E_i, the softmax output Y_i is computed, and prediction confidence measures such as entropy or max_prob are derived from Y_i and compared against the predefined threshold value T_i. If the condition is met, processing terminates and the prediction is returned; otherwise, the input sample propagates to the next side branch, repeating the process.

In entropy-based confidence criteria, predictions are considered confident if the entropy value falls below a predefined threshold, prompting the side branch to classify the input. Several models, including BranchyNet [123], FlexDNN [22], FastBERT [84], EENets [17], DynExit [132], DeeBERT [140], AdaDet [146], DynamicSleepNet [137], DeeDiff [121], and OdeBERT [87], apply this criterion. Other studies utilizing entropy-based criteria for early-exit policies include [20, 66, 71, 81, 89, 103, 110, 111, 124]. Threshold values vary across studies, determined empirically or chosen via heuristics. For example, EENets [17] used a threshold of 0.5 for all side branches, while Scardapane et al. [111] employed 0.7 for the first exit point and 0.5 for subsequent points. FlexDNN [22] set a uniform threshold of 0.8, whereas Panda et al. [103] opted for 0.5. Several studies utilize softmax-based metrics to assess prediction confidence and determine classification readiness, including MSDNet [16], SDN [62], CT [117], RANet [145], PersEPhonEE [75], SPINN [70], CDL [104], Large [113], T-RECX [27], LoCoExNet [58], DE^3-BERT [41], LGViT[143]. This metric applies a softmax layer to the output vector from each branch's fc layer, producing a probability vector where each element represents the likelihood of the sample belonging to each class. The highest probability indicates prediction confidence, compared against a predefined threshold. If the confidence meets or exceeds the threshold, the branch classifies the sample, terminating inference; otherwise, processing continues to subsequent layers. Pacheco et al. [100] and Kaya et al. [62] used a threshold of 0.8 for comparison. Halting score-based exit policies are employed in [21, 23, 31] where a halting function acts as a confidence criterion compared to a specific threshold. Figurnov et al. [23] and Graves et al. [31] introduced a halting function for each residual unit within blocks, with cumulative halting scores compared against a threshold of 1.0 to potentially skip the remaining units in the block.

E^{2}CM [30] introduced an early-exit policy that compares each neural layer's output features with a representative feature, averaged from all dataset samples. The Euclidean distance between the layer's output and the representative features is computed and normalized into a probability, which is then compared to predefined thresholds to assess prediction confidence. Unlike entropy-based or softmax-based exit policies, this method does not depend on gradient-based training of side branches, making it suitable for energy-constrained devices [30]. DE^3-BERT [41] argues that accurate early predictions require high estimation results from both local and global information. It integrates distance-based global information into the classic entropy-based early-exiting method, calculating the harmonic mean of entropy and distance ratio to improve exit decision-making. Experimental results show that this integration enhances prediction correctness, offering better tradeoffs between performance and efficiency, particularly in high-acceleration scenarios. While static or rule-based exit policies are easy to implement, they require expert knowledge or heuristics to set threshold values. These non-learnable confidence scores are less adaptable to changing environments. Since early-exit policies significantly impact system accuracy and efficiency [98], poorly designed policies can degrade performance and lead to overconfidence in side branches, causing incorrect classifications.

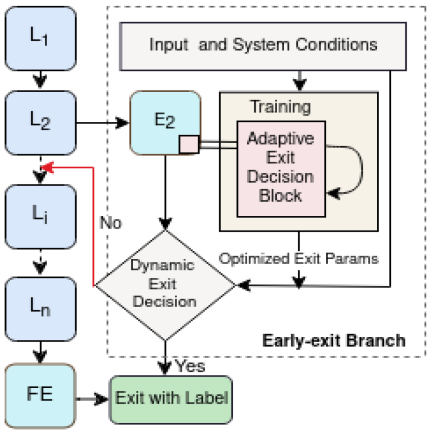


Fig. 11. Inference with dynamic exit policy.

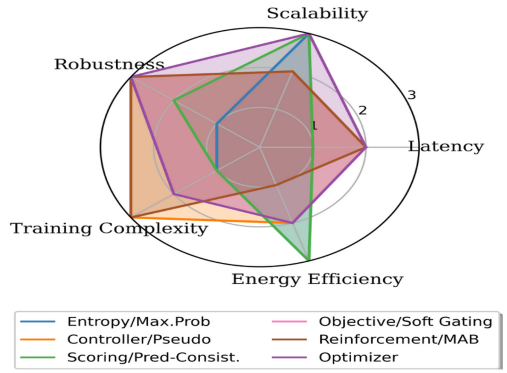


Fig. 12. Comparison of various exit policies.

6.2 Dynamic or Learnable Exit Policy

Dynamic or learnable exit policies offer an innovative approach in early-exit DNNs by automating the decision-making process for terminating processing at specific exit points. Instead of relying on manually designed rules or fixed thresholds, these policies leverage features and parameters from side branches to dynamically decide whether to exit or continue processing. By treating exit decisions as learnable parameters, a DNN can dynamically adjust its processing based on input features, system conditions, and contextual factors. Figure 11 illustrates this mechanism. During training, these parameters are optimized alongside the model using backpropagation and optimization techniques. Various studies have developed different methodologies for implementing learnable exit policies, such as exit selection controllers, reinforcement learning, and soft gating mechanisms, as described in Table 7. Integrating these methods into the model architecture enables early-exit DNNs to make informed decisions about processing termination, maximizing efficiency without compromising prediction quality. This dynamic approach optimizes the balance between prediction accuracy and computational efficiency, allowing for contextually appropriate early termination decisions.

EpNet [16] proposed a learning-based policy attaching exit controllers to each side branch, formulating the exit decision as a Markov decision process. This process learns early-exit policies from gradient parameters and environmental factors like input complexities and system resources. DQN [125] introduced a deep reinforcement learning-based approach for dynamic exit point selection, considering inference latency, battery state, and accuracy. DEE [59] and AdaEE [101] employed **Multi-Armed Bandits** (**MAB**)-based real-time online learning algorithm that adapts to varying input samples by evaluating each exit point in context. NLP-based applications like PABEE [156], BERxiT [141], and GAML-BERT [159] use a patience-score-based policy, stopping inference when intermediate predictions remain unchanged for several iterations. Bolukbasi et al. [9] transformed early-exit policy learning into a global objective function, solving it as a layer-wise weighted binary classification problem with a bottom-up training approach. LISTA-stop [14] developed a variational Bayes-based framework for modeling early-exit stopping policies, learning on-the-fly whether to terminate at the n^{th} layer for a given input. DDA [78] viewed early-exit DNNs as sequentially predicting sub-networks, assigning pseudo-class labels based on modeling certainty and using cosine similarity to measure confidence. If a sub-network's prediction closely matched the model's average prediction, execution could terminate. SkipNet [135] combined supervised and reinforcement learning to determine which layers to skip. BlockDrop [138] used associative

Table 8. Comparison of Early-exit Policies

Exit Policy	Strengths	Weaknesses
Entropy Comparison	Simple, effective for certain data distributions	Poor adaptation to diverse/changing inputs
Exit Selection Controllers	Adaptable, learns optimal exit points	Extra training, high computational needs
Softmax Probability Comparison	Simple, efficient, interpretable	Fixed thresholds may not suit all scenarios
Multi-Armed Bandits	Informed decisions, high adaptability	Complex, sensitivity to input distribution:
Scoring Function Comparison	Flexible, easy to understand	Limited adaptability in dynamic environments
Objective Function	Objective-specific optimization, customizable	Requires careful design, computationally heavy
Pseudo Labels and Prediction Mean	Regularizes classifiers, semi-supervised potential	Relies on pseudo labels, complex training.
Soft Gating Mechanism	Selective processing, saves resources	Gating needed, complex to implement.
Reinforcement Learning Technique	Learns optimal policies, highly adaptable	Computationally intensive, extensive training
Prediction Consistency	Simple, prevents unnecessary computations	Poor generalization to varied inputs
Optimizer Algorithm	Hardware-specific optimization, efficient	Complex algorithms, hardware tuning needed

reinforcement learning and a reward system to minimize block usage while maximizing accuracy. Guan et al. [32] defined early-exit policies as a Markov decision process, using reinforcement learning to train the exit policy based on the output probabilities of each classifier. In HydraNet [95], a gating mechanism selects branches for each input instance. These branches extract unique features rather than predict the final outcome, with the final prediction produced by combining the results of all selected branches.

6.3 Comparison of Early-exit Policies

Figure 12 evaluates early-exit policies based on training complexity, inference latency, scalability, and robustness to input variability. Table 8 outlines the strengths and weaknesses of each policy, highlighting tradeoffs in selecting an appropriate strategy. Training complexity refers to the resources required to train with a specific exit policy, while inference latency measures processing time. Scalability assesses efficiency across data scales and model sizes, and robustness gauges adaptation to input variability, while energy efficiency pertains to minimizing energy use. Static methods like entropy comparison and softmax probability comparison exhibit low training complexity and inference latency due to their reliance on predefined thresholds, making them simple to implement and quick to execute. However, these methods show lower robustness to diverse input characteristics because they lack adaptability. Conversely, dynamic methods such as exit selection controllers and reinforcement learning techniques have high training complexity due to sophisticated learning components but excel in robustness by learning optimal exit points and adapting to varying inputs and conditions. While more computationally intensive, these dynamic methods offer greater flexibility and contextually informed decisions, ensuring optimal performance across different scenarios. Context-based inference history and objective function-based inference methods balance medium inference latency with high robustness by leveraging historical data and learned objectives. Most methods maintain high scalability, adaptable and deployable across various scales, yet dynamic approaches may demand increased computational resources. This comprehensive evaluation underscores the tradeoffs between simplicity and adaptability, guiding the selection of an early-exit strategy that best fits the specific needs of a given application.

7 Applications of Early-exit Neural Networks

DNNs with early-exit strategies have been applied to CV and NLP applications. Section 7.1 discusses CV applications, while Section 7.2 covers NLP applications. Section 7.3 summarizes early-exit DNN applications in distributed computing.

Table 9. Computer Vision (Image Classification) Studies Using Early-exit DNN

Reference	Backbone Models	Datasets	Achievements
BranchyNet [123]	LeNet, AlexNet, ResNet	MNIST, CIFAR-10/100	2× −6× speedup on both CPU and GPU.
EpNet [16]	ResNet	MNIST, CIFAR-10	3% costs reduction and accuracy improvement within budget.
MSDNet [50]	MSDNet	CIFAR-10/100, ImageNet	6% higher accuracy with 2–3 times less computation time.
HydraNet [95]	ResSep	CIFAR-100, ImageNet	Cost reduction by 2–4× and accuracy improvement by 2.5%.
DynExit [132]	ResNet	CIFAR-10/100	Save up to 43.6% FLOPS.
EENets [17]	ResNet	MNIST, CIFAR-10, SVHN	Better performance with 20% of the total computational cost.
Neshatp. et al. [97]	ICNN	ImageNet	38% reduction in execution time.
RANet [145]	DenseNet, ResNet	CIFAR-10/100, ImageNet	Reduced spatial redundancy and optimized resource use.
Li et al. [77]	MSDNet	CIFAR-10/100, ImageNet	>30% cost reduction and >6% accuracy improvement.
DDI[136]	ResNet, DenseNet	CIFAR-100, ImageNet	Reduces computational costs by up to 4× .
PTEENET [66]	ResNet, DenseNet, VGG	SVHN, CIFAR-10	Significant reduction in computational cost.
DQN [125]	AlexNet, MobileNet, ResNet, VGG-16	CIFAR-10	Reduction in execution time by 63.5% and cost by 33%.
E^2CM [30]	MobileNetV3, ResNet, EfficientNet	CIFAR-100, ImageNet, MNIST	Higher accuracy under training time and computational cost budget.
PersEPhonEE [75]	ResNet, MobileNetV2	ImageNet, FEMNIST	3.2× throughput increase, 3.1× lower computational cost, 25.1× fewer parameters, and training speed up to 23× .
LGViT[143]	ViT, DeiT, Swin	CIFAR-100, Food-101, ImageNet	Speed-up of 1.8× with less than 2% accuracy sacrifice.
T-RECX [27]	ResNet, MobileNetV1, DS-CNN	CIFAR-10	Improved accuracy with a 31.58% reduction in FLOPS for 1% accuracy tradeoff.
LoCoExNet [58]	VGG-16, ResNet, MobileNetV2	CIFAR-10/100, ImageNet	91% energy savings and 4.46× speedups; used 28.7% of total model parameters while maintaining accuracy.
EENet [79]	VGG, ResNets	CIFAR-10/100, SVHN, CINIC	Upto 63.8% energy savings compared to traditional networks and 21.5% savings over existing early-exit strategies.
Han et al. [35]	MsdNet	CIFAR-100, ImageNet	Improved early-exit performance while preserving later exits.
Bolukbasi et al. [9]	Alex/Res Net, GoogLeNet	ImageNet	2.8× speedup with less than 1% loss of top-5 accuracy.
BlockDrop [138]	ResNet	CIFAR-10/100, ImageNet	20% average speedup with ImageNet top-1 accuracy at 76.4%.
LISTA-stop [14]	VGG-16	Tyni ImageNet	Boosted accuracy from 77.78% to 83.26% after Stage 1.
EENet [52]	Res/Dense Net, MSDNet	CIFAR-10/100, ImageNet	0.23% to 1.95% accuracy gain under budget constraints.

7.1 Early-exit DNN in Computer Vision Applications

Early-exit DNNs have been widely applied in CV. This section categorizes them into "image classification" and "other CV applications". Table 9 details studies on early-exit in image classification, while Table 10 covers applications in areas such as object detection, image synthesis, and video analytics.

7.1.1 Image Classification. Early-exit DNNs, as summarized in Table 9, have been extensively studied for image classification across diverse datasets and hardware platforms, demonstrating their practical utility in real-world applications. BranchyNet [123] optimizes inference speed and efficiency through early-exit strategies, leveraging LeNet-5, AlexNet, and ResNet backbones on datasets like MNIST, CIFAR-10, and CIFAR-100. It achieves speedups of 2× −6× on both CPU and GPU while maintaining high accuracy. EpNet [16] dynamically selects the optimal exit branch based on inference budgets, yielding up to 3% higher accuracy on MNIST and CIFAR-10 compared to state-of-the-art methods. MSDNet [50] employs a multi-scale network structure and dense connectivity to preserve coarse-level features, achieving 6% higher accuracy with 2–3 times less computation time than baseline models, suitable for high-precision tasks on mobile devices. HydraNet

Table 10. Other Computer Vision-based Studies Using Early-exit DNN

Reference	Application	Backbone Models	Datasets
FlexDNN [22]	Video Analytics	Inception-V3, VGG-16	UCF101
DEE [59]	Video Analytics	AlexNet, ResNet	Two real-world video streams
Sabet et al. [110]	Object Detection	Faster-RCNN	CDNET
FIANCEE [61]	Image Synthesis	OASIS	Cityscapes
DeeDiff [121]	Image Synthesis	Diffusion Models	CIFAR-10, Celeb-A, MS-COCO
ASE [94]	Content Generation	DiT, U-ViT	ImageNet, CelebA
ESNO [45]	Object Detection	VGG-16	Pascal-VOC 2007/2012
AdaDet [146]	Object Detection	MSDNet	MS COCO
TLEE [133]	Video Recognition	ResNet	UCF101, HMDB51
EENet [52]	Image Segmentation	MSDNet, HRNet	Cityscapes
BranDeepLabV3 [28]	Image Segmentation	DeepLabV3	Pascal-VOC 2012
Chiu et al.[15]	Facial Attribute Classification	ResNet	CelebA, UTK Face

[95] integrates residual connections and depth-wise separable convolutions from MobileNet and ResNet into its ResSep backbone. Specialized branches process features for visually similar classes, reducing inference costs by 2–4× and enhancing accuracy by up to 2.5% on CIFAR-100 and ImageNet. This reduction in inference time is critical for real-time applications such as autonomous vehicles and robotic systems, where rapid decision-making is crucial.

DynExit [132] and DQN [125] utilized early-exit concepts to reduce computational energy consumption of DNNs through hardware-centric approaches in image classification. DynExit introduced dynamic loss-weight adjustments and a hardware-friendly ResNet architecture, achieving a 43.6% reduction in FLOPS and a 1.2% accuracy improvement. DQN utilized a quantization-aware model selection approach, optimizing execution time and energy consumption through hardware and early-exit parameters. It achieved a 63.5% reduction in execution time and a 33% reduction in computational energy consumption across AlexNet, ResNet18, VGG-16, and MobileNet. EENets [17] improved hardware efficiency by implementing multiple exit blocks with learnable classification and confidence branches. This approach reduced computational costs by about five times compared to standard ResNets, achieving equivalent or better accuracy with only 20% of the computational cost. Neshatpour et al. [97] demonstrated effective resource management in low-power, real-time systems by combining context-aware pruning and threshold-based early-exits with ResNet, reducing computational costs by 38% with only a 1% accuracy loss. RANet [145] extends early-exit concepts to reduce spatial redundancy by adaptively adjusting input resolutions with multiple sub-networks, each equipped with its own classifier. This approach proves effective for applications with stringent computational budgets, such as real-time image classification in remote sensing and surveillance. Bolukbasi et al. [9] applied an early-exit policy for object recognition on ImageNet using AlexNet, GoogLeNet, and ResNet, achieving up to 2.8× speedup with under 1% top-five accuracy loss. LISTA-stop [14] added 14 exits to VGG-16, increasing Tiny-ImageNet accuracy from 77.78% to 83.26% post-Stage I. BlockDrop [138] used reinforcement learning to dynamically select blocks, achieving a 20% average speedup on CIFAR-10, CIFAR-100, and ImageNet, with ImageNet top-1 accuracy at 76.4%.

E^2CM [30] early-exit capabilities in unsupervised learning using a gradient-free strategy with ResNet. This allows training on low-power devices like wireless edge networks, significantly cutting computational costs and inference times. Li et al. [77] improved performance on CIFAR-10, CIFAR-100, and ImageNet in early-exit DNN frameworks using collaborative strategies, dense connections, and gradient equilibrium techniques. DDI [136] integrates input and resource-aware

dynamic inference with multi-grained skipping to selectively process model layers and channels based on input characteristics. This achieves superior accuracy-resource tradeoffs across diverse datasets. Using ResNet and DenseNet backbones, DDI reduces computational costs by up to four times while maintaining accuracy, essential for resource-constrained applications like drones. PersEPhonEE [75] personalizes early-exit DNN models for individual users by adjusting parameters and exit strategies based on real-time data. It delivers up to 3.2× higher throughput, 3.1× reduced computational costs, and 25.1× fewer parameters. It also accelerates early-exit block training by up to 23× . These improvements highlight the effectiveness of early-exit DNNs in personalizing real-time applications for resource-constrained environments. PTEENET [66] utilizes pre-trained DNN models as backbones and integrates side branches to address computational cost and architectural constraints. It exclusively trains these side branches, employing the backbone as a label generator for unlabeled datasets. Evaluated on SVHN and CIFAR-10 with ResNet, DenseNet, and VGG backbones, PTEENET significantly reduces computational costs while maintaining accuracy, demonstrating the versatility of early-exit DNN techniques for handling unlabeled data in resource-constrained environments.

LGViT [143] integrates early-exiting into vision transformers by incorporating heterogeneous exiting heads: a local perception head and a global aggregation head. This design balances efficiency and accuracy, achieving an average speed-up of 1.8× with less than 2% accuracy sacrifice across three vision datasets and three general ViTs. This makes LGViT suitable for resource-limited edge devices in multimedia services. T-RECX [27] extends early-exit strategies to tinyML, optimizing tiny-CNN models to reduce overthinking. It uses early-exit representations to reclaim lost accuracy, improving image classification, keyword spotting, and visual wake word detection from the MLPerf tiny benchmark. T-RECX achieves a 31.58% average reduction in FLOPS for a 1% accuracy tradeoff, consistently outperforming previous methods and enhancing tinyML applications effectively. LoCoExNet [58] is a low-cost early-exit DNN enhancing energy efficiency with optimized branch parameters and dynamic branch pruning. It reduces fc layer parameters without sacrificing accuracy, achieving 91% energy savings and 4.46× speedups on CIFAR-10 with VGG-16, demonstrating substantial energy and performance enhancements for CNN accelerators. EENet [79] enhances current early-exit strategies by integrating a dynamic power management module into state-of-the-art networks. It predicts early-exits during inference and adjusts computing configurations (frequency and voltage) accordingly, providing calibration advice based on varying workloads and timing constraints. This double-timescale power management approach achieves up to 63.8% energy savings compared to traditional networks and 21.5% savings over existing early-exit strategies, while significantly improving timing performance. Han et al. [35] improved early-exit DNN with a training method that imposes distinct objectives on individual blocks using grouping and overlapping strategies. This approach enhances early-exit performance without compromising later exits, making it suitable for low-latency applications like self-driving cars with dynamically changing speeds. Experiments on image classification and semantic segmentation validate its effectiveness, seamlessly integrating with existing multi-exit strategies without requiring model architecture modifications.

7.1.2 Other CV Applications. Table 10 covers early-exit DNN applications in areas such as object detection and recognition, image synthesis and segmentation, video analytics, and content generation. ESNO [45] and AdaDet [146] extended early-exit DNNs into object detection. ESNO, evaluated on Pascal VOC 2007 and 2012 datasets, detects objects with minimal cost and adapts detection operations to balance accuracy and computational cost. AdaDet [146] uses stochastic variables and an entropy-based criterion to measure detection uncertainty, allowing high-confidence samples to exit early while complex images undergo full computation. On the MS COCO dataset,

AdaDet achieves 41.7% anytime prediction with only 56.0G FLOPs. Sabet et al. [110] added early-exit branches to Faster-RCNN for video object detection, requiring full model execution only for frames with significant semantic differences, reducing computational cost by 34× . EENet [52] advances early-exit techniques in image recognition, image segmentation, and sentiment analysis by learning a scheduler to terminate inference early for high-confidence predictions. EENet's experiments across CIFAR-10/100, ImageNet, Cityscapes, SST-2, and AgNews demonstrate superior performance over heuristic-based methods while maximizing model accuracy within per-sample average inference budgets.

DEE [59], FlexDNN [22], and TLEE [133] tackle the latency and resource demands of CNNs in video analytics through early-exit models. DEE uses a contextual bandit algorithm for real-time online learning, enhancing performance by up to 98.1% in real-world scenarios. FlexDNN is a versatile framework for efficient on-device video analytics, dynamically adjusting model complexity based on input frame difficulty. Using an architecture search-based scheme, it optimizes side branch configurations, reducing computational costs by up to 49.8% and energy consumption by up to 1.9× , surpassing existing approaches in accuracy, processing time, frame drop rate, and energy efficiency. TLEE enables both temporal and layer-wise early exits for video recognition on resource-constrained devices, significantly reducing computational costs and inference latency without sacrificing accuracy. Tested on the NVIDIA Jetson Nano, TLEE surpasses existing state-of-the-art methods. FIANCEE [61] introduces early exit branches in generative DNNs for image synthesis, adapting computational paths based on input complexity. Tested on the Cityscapes and MegaPortraits datasets, it maintains quality thresholds while reducing computations by up to 50% for LPIPS (Learned Perceptual Image Patch Similarity) [152] scores of ≤ 0.1. Compatible with existing models, FIANCEE effectively balances quality and efficiency, making it particularly suitable for real-time applications like facial synthesis.

DeeDiff [121] addresses slow generation speeds in diffusion models through an early-exiting framework, incorporating a timestep-aware uncertainty estimation module. This approach reduces layers by 47.7% on CIFAR-10, Celeb-A, ImageNet, and MS-COCO while achieving superior FID (Frechet Inception Distance) scores [3]. ASE [94] similarly tackles slow sampling speeds, optimizing compute allocation by selectively skipping parameters based on a time-dependent schedule, leading to a 30% reduction in calculation costs without sacrificing FID scores across U-ViT and DiT models. Chiu et al. [15] investigate the tradeoff between accuracy and fairness in facial attribute classification, proposing multi-exit models with internal classifiers that improve fairness by 20.3% on CelebA and 10% on UTK Face, with minimal accuracy loss. Together, these studies highlight the effectiveness of early-exit strategies in enhancing efficiency and performance in various computer vision applications, making DNNs more suitable for low-resource and real-time deployments.

7.2 Early-exit DNN in NLP Applications

Early-exit DNNs have been effectively applied in NLP, similar to their use in CV. Table 11 summarizes relevant studies integrating side branches into transformer models like BERT [63], RoBERTa [88], and ALBERT [67]. PABEE [156], DeeBERT[140], BERxiT [141], PF-BERxiT [24], Leco [150], and DE^3-BERT [41] have extended early-exiting models in NLP with their effectiveness evaluated on the **General Language Understanding Evaluation** (**GLUE**) benchmark. PABEE enhances efficiency by using internal classifiers to stop inference when predictions stabilize, reducing layers and improving accuracy in BERT and ALBERT. DeeBERT integrates early exits with BERT and RoBERTa, reducing inference time by up to 40% with minimal quality loss, and facilitating real-time deployment. PF-BERxiT [24] addresses computational challenges in large language models with a parameter-efficient fine-tuning and adaptable early-exit strategy based on adjacent layer

Table 11. NLP-based Studies Using Early-exit DNNs

Reference	Application	Backbone Models	Datasets
BERxiT [141]	Lang. Understanding	BERT, ALBERT, RoBERTa	GLUE
DeeBERT [140]	Lang. Understanding	BERT, RoBERTa	GLUE
DE^3-BERT [41]	Lang. Understanding	BERT	GLUE
Leco [150]	Lang. Understanding	LBERT	GLUE
PABEE [156]	Lang. Understanding	BERT, ALBERT	GLUE
PF-BERxiT [24]	Lang. Understanding	ALBERT	GLUE
Elbayad et al. [21]	Machine translation	Transformer	WMT'14 En-Fr, IWSLT'14 De-En
Garg et al. [25]	Question Answering	BERT, RoBERTa, ELECTRA	WikiQA, ASNQ, SQuAD 1.1
ReasoNet [115]	Question Answering	ReasoNets	Daily Mail, SQuAD, SGR
EENet [52]	Sentiment Analysis	BERT	SST-2, AgNews
HiDEC [53]	Sentiment Analysis	BERT	SST-2, AgNews
Large [113]	Sentiment Analysis	BERT	SST, IMDB
LSTM-Jump [147]	Sentiment Analysis	LSTM	IMDB, Yelp, SST, RT, DBPedia, AGNews, CBT-CN/NE
CT [117]	Text Ranking	RoBERTa	WikiQA, TRECQA, ASNQ, GPD
Xin et al. [139]	Text Ranking	BERT	MS MARCO, ASNQ
OdeBERT [87]	User Intent classification	BERT, RoBERTa	SNIPS, FDQuestion, ECDT
Jumper [86]	Text Classification	Jumper	MR, AGNews, OI
FastBERT [84]	Text Classification	BERT	ChnSentiCorp, BR, SR, Weibo, THUCNews, AgNews, AmzF, DBpedia, Yahoo, YelpF/P

similarity scores. It outperforms competitors with minimal fine-tuned parameters on the GLUE benchmark, offering superior speedup-accuracy tradeoffs and making it a valuable choice for efficient NLP applications. Leco [150] improves multi-exit BERT models with a broad architectural search space and comparison-based early-exit mechanism, achieving a 1.1% performance boost on GLUE. DE^3-BERT [41] enhances performance-efficiency tradeoffs by employing prototypical networks for integrating local and global data, facilitating more reliable early-exits. Experiments on the GLUE benchmark demonstrate DE^3-BERT's superior performance, generality, and interpretability. Elbayad et al. [21] developed an early-exit transformer model based on a sequence-to-sequence architecture [126] for machine translation. Their approach achieved baseline transformer accuracy on the IWSLT German-English dataset [12] with 2.5× fewer decoder layers. Garg et al. [25] implemented an early-exit question-answering system using BERT, RoBERTa, and ELECTRA, reducing computational costs by 60%. ReasoNet [115], a reasoning network, incorporates early-exits through reinforcement learning techniques across multiple turns, outperforming baseline networks on machine comprehension datasets [108]. FastBERT [84] and Jumper [86] introduced dynamic inference time strategies for text classification through early-exits. FastBERT optimizes computation by dynamically adjusting inference based on demand, achieving 2 to 3 times faster execution than BERT across twelve NLP datasets (six in English and six in Chinese) [85, 107, 153], while maintaining accuracy and allowing speed variations from 1× to 12× . Jumper considers text classification as a series of decision-making processes, making early exits when evidence is sufficient, and achieves classification accuracy comparable to or exceeding state-of-the-art models. Soldaini et al. [117] and Xin et al. [139] applied early-exit techniques to text ranking applications. Soldaini et al. [117] developed a CT using RoBERTa as the backbone, which increased throughput and reduced execution costs by 37% on two English question-answering datasets [26]. Xin et al. [139] created an early-exit BERT model for document ranking that achieved 2.5× faster inference while maintaining high quality.

LSTM-Jump [147], Large [113], HiDEC [53], and EENet [52] employed early-exit techniques to enhance sentiment analysis. LSTM-Jump skips irrelevant information during text processing, accelerating the inference process by up to six times compared to standard sequential LSTM models, all while achieving the same or even better accuracy. Large attached multiple classifiers to the intermediate layers of BERT, using the prediction confidence scores to determine early exits. This approach significantly outperforms existing methods, offering up to a 5-fold improvement in efficiency while preserving accuracy. HiDEC [53] presents a hierarchical dynamic early-exit mechanism that adapts the number of processing layers based on the complexity of the input. This model enhances sentiment analysis efficiency by dynamically adjusting the computation based on input difficulty, resulting in faster inference times without compromising performance. It achieved over 100% speedup on the AgNews dataset while maintaining high accuracy. EENet advances the early-exit framework with a novel ensemble approach for sentiment analysis. By combining multiple early-exit strategies, EENet achieves significant speedups in inference times and enhances model robustness, delivering high accuracy and efficiency. OdeBERT [87] addresses the computational demands and latency of BERT in user intent classification through an early-exit model that incorporates deep supervision with internal classifiers and capsule networks. This approach accelerates BERT inference by up to 12 times on the ECDT, SNIPS, and FDQuestion datasets while maintaining performance on par with current state-of-the-art methods.

Wenjian et al. [137] and Lattanzi et al. [71] applied early-exit DNNs in signal processing. Wenjian et al. introduced DynamicSleepNet for sleep stage classification using electrophysiological signals, which balances accuracy and efficiency by connecting feature extraction modules to classifiers for early output when uncertainty is low. This adaptive approach reduces redundant training, lowers computational costs, and maintains accuracy. Validated on datasets like Sleep-EDF and ISRUC, it outperforms models like AttnSleepNet and SalientSleepNet in both speed (up to 23.81 times faster) and accuracy, making it practical for sleep medicine. Lattanzi et al. explored early-exit techniques in human activity recognition using 1D CNN, 2D CNN, LSTM, and their combinations, evaluated on datasets such as WISDM and UCI-HAR. They achieved up to 35× latency reduction without compromising accuracy. Their study also highlighted that early-exit strategies might not benefit low-latency LSTMs or 2D CNNs needing distant exit points, and they examined the impact of communication delays on performance across different devices. As deep learning-based biomedical signal processing is typically too heavy for small devices, early-exit versions like DynamicSleepNet and Lattanzi et al.'s techniques enhance deployment in resource-constrained devices by improving efficiency and reducing latency.

7.3 Early-exit DNN in Distributed Computing Applications

Recent computational frameworks such as 5G networks, IoT sensors, and edge computing follow a multi-tier paradigm, where decisions can be made at the network's edge or delegated to powerful decision centers such as cloud servers, incurring some communication latency [105, 157]. In these environments, multiple neural networks of varying complexity are designed to operate on different tiers, such as edge or cloud servers [89, 96]. Early-exit DNNs naturally fit into these paradigms by distributing DNN computation across tiers, as depicted in Figure 13. Table 12 summarizes early-exit DNN applications in distributed computing systems. DDNN [124] extends early-exit DNNs to mobile edge-cloud computing systems. They deployed a simple neural model on mobile devices and a medium-sized model with exit points on the edge server, evaluated on the **Multi-view Multi-camera Dataset** (**MMD**). If the mobile device's prediction confidence is insufficient, the output is sent to the edge server. If the edge server's prediction confidence remains insufficient, the output is then sent to the cloud server, which executes the largest model for the final prediction. To efficiently choose layers for edge execution and minimize inference

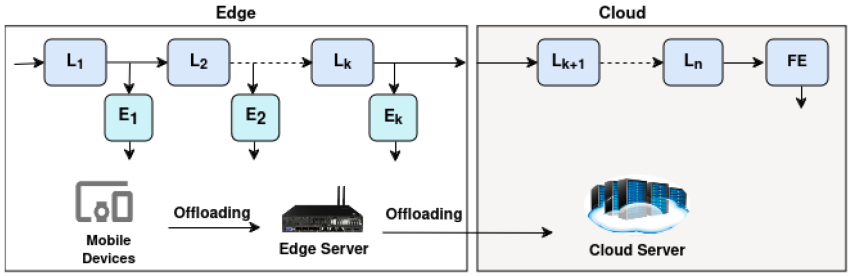


Fig. 13. Early-exit DNN processing in a distributed environment.

Table 12. Distribute Computing Application of Early-exit DNN

Application Scope	Datasets	Base Models	Reference
Align DNN segments in a distributed computing hierarchy	MMD	BNNs	DDNN [124]
DNN partitioning	Cat and Dog	Branchy- Alexnet	Pacheco et al. [99]
DNN partitioning and right-sizing	CIFAR-10	AlexNet	Boomerang [149]
DNN partitioning	CIFAR-10, Cat and Dog	ResNet, Inception V3	Ebrahimi et al. [20]
DNN partitioning	Caltech-256	MobileNetV2	Pacheco et al. [98]
Efficient edge offloading of inference	Caltech-256	MobileNetV2	Pacheco et al. [100]
DNN partitioning and right-sizing	CIFAR-10	Branchy- Alexnet	Edgent [82]
Dynamic network sizing	CIFAR-10, CIFAR-100	ResNet, WRN, NIN	Lo et al. [89]
Dynamic selective DNN execution	CIFAR-10, CIFAR-100	VGG-16, ResNet50	EdgeML [155]
Fast and reliable inference across devices and the cloud	CIFAR-100, ImageNet	ResNet, MobileNetV2, VGG-16, Inception-v3	SPINN [70]
Latency-aware inference	ImageNet, SST-2, AGNews	MsdNet	HiDEC [53]

time, Pacheco et al. [99] formulated BranchyNet partitioning as a shortest path problem, solving it with Dijkstra's algorithm. Boomerang [149] employs early-exit concepts for DNN right-sizing, enabling adaptive model partitioning between IoT devices and edge servers by carefully selecting exit and partition points to maximize performance while maintaining efficiency. Ebrahimi et al. [20] proposed a performance model that balances computation load across multiple servers using early-exit techniques.

Pacheco et al. [98] analyzed early-exit DNNs in an adaptive offloading scenario, implementing an image classification application using MobileNetV2 with early exits on the Caltech256 dataset. The early-exit DNN was split into two sections: the first, with DNN layers and side branches, was deployed on the edge device, while the second, containing subsequent layers, was on the cloud server. The edge device processes input samples up to the exit points, sending low-confidence predictions to the cloud. Although data offloading incurs communication delays, this approach saves inference time by processing "easy" images locally and offloading only complex ones. Their research demonstrated that early classification at the edge reduces data offloading and latency without sacrificing accuracy. For distorted images, more inferences were offloaded due to reduced branch accuracy [19]. To mitigate this, Pacheco et al. [100] introduced expert side branches trained on specific distortion types, allowing the edge to select the most suitable branches for improved offloading decisions. Lo et al. [89] developed a dynamic network structure with stochastic threshold values for different DNN output classes, determining the need for input transfer to remote computational units. Their early-exit model adjusts depth based on channel availability, achieving a 7% reduction in data offloading compared to existing frameworks while maintaining accuracy.

Edgent [82] is a device-edge collaboration framework designed to maximize system performance while maintaining minimal latency. It uses early-exit techniques for DNN partitioning and right-sizing. Computation is split among devices, edge, and cloud servers, with intermediate side branches attached to DNN layers to reduce execution time. Experiments show that collaborative execution of early-exit DNN models across multi-tier platforms enables low-latency applications. EdgeML [155] is a reinforcement learning-based AutoML framework that enhances the deployment of resource-demanding DNNs on low-power devices with minimal communication delay. It combines efficient work offloading with early-exit DNNs, providing flexible control over DNN execution and achieving up to an 8× performance improvement over existing frameworks. SPINN [70] is a distributed inference system that collaborates DNN computation among devices and cloud servers. It uses early-exit techniques to efficiently partition DNNs across distributed computational units. SPINN's early-exit policy maker improves adaptation to changing conditions and user demands, reducing server costs by 6.8× and improving accuracy by 20.7% compared to conventional counterparts. HiDEC [53] is a hierarchical DNN inference framework for edge and cloud computing, enhancing AI applications in CV, NLP, autonomous driving, and smart cities. It features resource-adaptive DNN training with multiple early-exits, a latency-aware inference scheduler for edge devices, and a dual thresholding approach for early-exits. Compared to conventional early-exit DNN models, HiDEC consistently outperforms manual rule-based policies, especially under tighter latency budgets. For instance, on AgNews, HiDEC achieves over 100% speed gains with minimal accuracy loss, outperforming other methods that gain less than 50% speed at the cost of more than 2.5% accuracy. HiDEC's latency-aware adaptive inference framework for distributed computing hierarchies optimizes early-exit policies to maximize accuracy within given inference budgets per sample.

8 Research Challenges and Future Directions

Early-exit DNNs have been a promising research topic, with numerous articles contributing to significant advances. Despite this progress, many open research problems remain. This section discusses the challenges and potential future directions in early-exit DNNs.

8.1 Expanding into New Modalities and Applications

As illustrated in Table 9, early-exit DNN research has primarily focused on CV tasks, particularly image classification using CNNs as a base model. Domains such as video processing, medical applications, disaster management, machine translation, and text processing have received comparatively less attention. Recent efforts are beginning to explore early-exit DNNs in cross-domain applications, employing DNN variants like recurrent neural networks, generative adversarial networks, graph neural networks, and transformers. Implementing dynamic early-exit strategies in these diverse models introduces new challenges that need addressing. Moreover, there are numerous untapped domains where early-exit techniques could be beneficial. Currently, early-exit DNN models tailored for specific tasks cannot be readily applied across different applications. For example, a classification-oriented early-exit DNN may not seamlessly transition to an object detection task due to the lack of standardized criteria for evaluating input sample complexity. Developing a universal early-exit DNN model applicable across multiple domains remains a formidable challenge.

8.2 Filling Theoretical and Practical Gaps

The hardware and libraries for DNNs are currently optimized for conventional models and are not well-suited for early-exit neural networks. As a result, actual implementation lags behind theoretical efficiency estimates. Investigating the optimization of hardware in conjunction with algorithms

and neural network libraries for implementing early-exit DNNs is an important research direction. This will improve the efficiency of early-exit DNNs. Designing an efficient early-exit DNN that is compatible with existing hardware and software is a key open research challenge that must be addressed.

8.3 Optimal Architectural Design

Attaching side branches to conventional DNNs can incur additional computational costs if they fail to produce correct early predictions. The structure and distribution of side branches and the adopted early-exit policy significantly impact the efficiency of early-exit DNNs. Developing an optimal configuration that balances the additional computational cost of side branches with the performance gains from early-exits is challenging. Optimal configurations, including backbone model selection, the number and structure of side branches, and their placement, have yet to be fully investigated.

8.4 Optimal Training Strategy

Effectively training early-exit DNNs is a critical task. In conventional DNNs, intermediate layers near the input extract lower-level features, while deeper layers extract more detailed features. Early-exit DNNs force intermediate layers near the input to extract coarse features via side branches. In a joint or end-to-end training strategy, this can create tension among the gradients of different side branches, reducing overall accuracy. A separate or two-stage training strategy may add extra overhead and reduce performance. Developing an optimal training strategy that combines the benefits of both approaches is an important research area to explore.

8.5 Effective Early-exit Policies

Most existing studies use a static early-exit policy with human-crafted rules or heuristics to determine exit criteria. Some studies employ learnable early-exit policies that reduce human intervention but need further research for accurate prediction. Developing a fully dynamic, learnable early-exit policy that considers the network's nature, input variations, and an unstable environment while deciding on early termination without compromising accuracy is a promising research direction.

8.6 Advancing Towards Explainable DNNs

Deep learning models are often non-transparent, making their interpretability a crucial research area. Early-exit DNNs, resembling the human visual system, can enhance the explainability of deep learning models by making them more transparent. Their dynamic architecture allows examination of intermediate layers, activations, and outcomes for specific inputs, offering insights into which parts of the model influence predictions. Research in early-exit DNNs with a focus on explainability is an important direction.

9 Conclusion

Deploying complex DNNs effectively in resource-constrained, low-latency applications poses significant challenges. Early-exit DNNs allow inference to stop early via side branches connected to the conventional DNN architecture. Besides speeding up inferences, they mitigate issues like vanishing gradients, overfitting, and overthinking. Early-exit DNNs offer flexibility in partitioning models and are ideal for distributed computing platforms. This article thoroughly explores the implementation of early-exit DNNs, offering a detailed analysis of recent advancements. To achieve optimal performance, side branches must be meticulously designed with appropriate neural layers and strategically placed along the DNN backbone. Designing an effective early-exit policy is

crucial as it heavily impacts performance. However, as this research field is still emerging, there is currently a lack of optimized configurations for constructing and placing side branches along the backbone model, as well as devising effective early-exit strategies. This survey underscores significant design issues and research challenges, aiming to stimulate further research in this exciting area. In the coming years, early-exit techniques are poised to become fundamental computational components rather than just nice-to-have features.

References

[1] Adrian Galdran, Aitor Alvarez-Gila, Maria Ines Meyer, Cristina L. Saratxaga, Teresa Araújo, Estibaliz Garrote, Guilherme Aresta, Pedro Costa, A. M. Mendonça, and Aurélio Campilho. 2017. Data-driven color augmentation techniques for deep skin image analysis. arXiv preprint arXiv:1703.03702 (2017).

[2] Enzo Baccarelli, Simone Scardapane, Michele Scarpiniti, Alireza Momenzadeh, and Aurelio Uncini. 2020. Optimized training and scalable implementation of conditional deep neural networks with early exits for fog-supported IoT applications. *Information Sciences* 521 (2020), 107–143.

[3] Fan Bao, Chongxuan Li, Yue Cao, and Jun Zhu. 2022. All are worth words: A ViT backbone for score-based diffusion models. In *Proceedings of the NeurIPS 2022 Workshop on Score-Based Methods*. Neural Information Processing Systems Foundation, New Orleans, LA, USA.

[4] Mohammad Mahdi Bejani and Mehdi Ghatee. 2021. A systematic review on overfitting control in shallow and deep neural networks. *Artificial Intelligence Review* 54, 8 (2021), 6391–6438.

[5] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. 2019. Shallow learning for deep networks. *Submitted to the International Conference on Learning Representations (ICLR'19)*.

[6] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. 2019. Greedy layerwise learning can scale to imagenet. In *Proceedings of the International Conference on Machine Learning*. PMLR, Long Beach, CA, USA, 583–593.

[7] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2006. Greedy layer-wise training of deep networks. In *Proceedings of the Advances in Neural Information Processing Systems 19*. MIT Press, Vancouver, British Columbia, Canada, 153–160.

[8] Konstantin Berestizshevsky and Guy Even. 2019. Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence. In *Proceedings of the International Conference on Artificial Neural Networks*. Springer, Munich, Germany, 306–320.

[9] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. In *Proceedings of the International Conference on Machine Learning*. PMLR, Sydney, Australia, 527–536.

[10] Andrew Brock, Theodore Lim, James Millar Ritchie, and Nicholas J. Weston. 2017. FreezeOut: Accelerate training by progressively freezing layers. In *Proceedings of the NIPS 2017 Workshop on Optimization*. Online, Long Beach, CA, United States, 5.

[11] Rich Caruana, Steve Lawrence, and C. Giles. 2000. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Proceedings of the Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp (Eds.). Vol. 13. MIT Press, Denver, CO, USA.

[12] Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th IWSLT evaluation campaign. In *Proceedings of the 11th International Workshop on Spoken Language Translation: Evaluation Campaign*. International Workshop on Spoken Language Translation (IWSLT), Lake Tahoe, CA, USA, 2–17. IWSLT 2014.

[13] Jiasi Chen and Xukan Ran. 2019. Deep learning with edge computing: A review. *Proceedings of the IEEE* 107, 8 (2019), 1655–1674.

[14] Xinshi Chen, Hanjun Dai, Yu Li, Xin Gao, and Le Song. 2020. Learning to stop while learning to predict. In *Proceedings of the International Conference on Machine Learning*. PMLR, Vienna, Austria, 1520–1530. ICML 2020.

[15] Ching-Hao Chiu, Hao-Wei Chung, Yu-Jen Chen, Yiyu Shi, and Tsung-Yi Ho. 2023. Fair multi-exit framework for facial attribute classification. arXiv preprint arXiv:2301.02989 (2023).

[16] Xin Dai, Xiangnan Kong, and Tian Guo. 2020. EPNet: Learning to exit with flexible multi-branch network. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. Association for Computing Machinery (ACM), Virtual Event, Ireland, 235–244. CIKM 2020.

[17] Edanur Demir. 2019. *Early-exit Convolutional Neural Networks*. . Middle East Technical University.

[18] Li Deng, Geoffrey Hinton, and Brian Kingsbury. 2013. New types of deep neural network learning for speech recognition and related applications: An overview. In *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, Vancouver, BC, Canada, 8599–8603. ICASSP 2013.

[19] Samuel Dodge and Lina Karam. 2016. Understanding how image quality affects deep neural networks. In *Proceedings of the 2016 8th International Conference on Quality of Multimedia Experience (QoMEX)*. IEEE, Lisbon, Portugal, 1–6.

[20] Maryam Ebrahimi, Alexandre da Silva Veith, Moshe Gabel, and Eyal de Lara. 2022. Combining DNN partitioning and early exit. In *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*. ACM, Portland, OR, USA, 25–30.

[21] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-adaptive Transformer. In *Proceedings of the ICLR 2020-8th International Conference on Learning Representations*. OpenReview.net, Addis Ababa, Ethiopia, 1–14.

[22] Biyi Fang, Xiao Zeng, Faen Zhang, Hui Xu, and Mi Zhang. 2020. Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision. In *Proceedings of the 2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE/ACM, San Jose, CA, USA, 84–95.

[23] Michael Figurnov, Maxwell D. Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. 2017. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Honolulu, HI, USA, 1039–1048.

[24] Xiangxiang Gao, Yue Liu, Tao Huang, and Zhongyu Hou. 2023. PF-BERxiT: Early exiting for BERT with parameter-efficient fine-tuning and flexible early exiting strategy. *Neurocomputing* 558 (2023), 126690.

[25] Siddhant Garg and Alessandro Moschitti. 2021. Will this question be answered? Question filtering via answer model distillation for efficient question answering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). ACL, Online and Punta Cana, Dominican Republic, 7329–7346.

[26] Siddhant Garg, Thien Vu, and Alessandro Moschitti. 2020. TANDA: Transfer and adapt pre-trained transformer models for answer sentence selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, New York, NY, USA, 7780–7788.

[27] Nikhil P. Ghanathe and Steve Wilton. 2023. T-RECX: Tiny-resource efficient convolutional neural networks with early-exit. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*. ACM, New York, NY, USA, 123–133.

[28] Mateus S. Gilbert, Roberto G. Pacheco, Rodrigo S. Couto, Marcello L. R. De Campos, and Miguel Elias M. Campista. 2023. Unlocking early-exiting semantic segmentation with branched networks. In *Proceedings of the 2023 IEEE Latin-American Conference on Communications (LATINCOM)*. IEEE, Panama City, Panama, 1–6.

[29] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, Chia Laguna Resort, Sardinia, Italy, 249–256.

[30] Alperen Görmez, Venkat R. Dasari, and Erdem Koyuncu. 2022. E^2CM: Early exit via class means for efficient supervised and unsupervised learning. In *Proceedings of the 2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Padua, Italy, 1–8.

[31] Alex Graves. 2016. Adaptive computation time for recurrent neural networks. In *Proceedings of the NIPS 2016 Deep Learning Symposium*. Neural Information Processing Systems Foundation, Barcelona, Spain, 19.

[32] Jiaqi Guan, Yang Liu, Qiang Liu, and Jian Peng. 2018. Energy-efficient amortized inference with cascaded deep classifiers. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. AAAI Press, Stockholm, Sweden, 2184–2190.

[33] Tian Guo. 2018. Cloud-based or on-device: An empirical study of mobile deep inference. In *Proceedings of the 2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, Orlando, FL, USA, 184–190.

[34] Lav Gupta, Raj Jain, and Gabor Vaszkun. 2015. Survey of important issues in UAV communication networks. *IEEE Communications Surveys and Tutorials* 18, 2 (2015), 1123–1152.

[35] Dong-Jun Han, Jungwuk Park, Seokil Ham, Namjin Lee, and Jaekyun Moon. 2023. Improving low-latency predictions in multi-exit neural networks via block-dependent losses. *IEEE Transactions on Neural Networks and Learning Systems* TNNLS.2023.3282249 (2023), 1–9.

[36] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proceedings of the ICLR 2016-4th International Conference on Learning Representations*. International Conference on Learning Representations (ICLR), San Juan, Puerto Rico.

[37] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. 2021. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 11 (2021), 7436–7456.

[38] P. Haseena Rahmath and Kuldeep Chaurasia. 2023. Adaptive early-exit inference in graph neural networks based hyperspectral image classification. In *Proceedings of the International Conference on Intelligent Systems Design and Applications*. Springer, 444–453.

[39] P. Haseena Rahmath, Kuldeep Chaurasia, Anika Gupta, and Vishal Srivastava. 2024. HyperGCN – a multi-layer multi-exit graph neural network to enhance hyperspectral image classification. *International Journal of Remote Sensing* 45, 14 (2024), 4848–4882.

[40] P. Haseena Rahmath, Vishal Srivastava, and Kuldeep Chaurasia. 2023. A strategy to accelerate the inference of a complex deep neural network. In *Proceedings of the Data Analytics and Management: ICDAM 2022*. Springer, online, 57–68.

[41] Jianing He, Qi Zhang, Weiping Ding, Duoqian Miao, Jun Zhao, Liang Hu, and Longbing Cao. 2024. DE3-BERT: Distance-enhanced early exiting for BERT based on prototypical networks. arXiv preprint arXiv:2402.05948 (2024).

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, Santiago, Chile, 1026–1034.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Las Vegas, NV, USA, 770–778.

[44] Jeff Heaton. 2018. Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The MIT press, 2016, 800 pp, ISBN: 0262035618. *Genetic Programming and Evolvable Machines* 19, 1 (2018), 305–307.

[45] Rory Hector, Muhammad Umar, Asif Mehmood, Zhu Li, and Shuvra Bhattacharyya. 2021. Scalable object detection for edge cloud environments. *Frontiers in Sustainable Cities* 3 (2021), 675889.

[46] Chris Hettinger, Tanner Christensen, Ben Ehlert, Jeffrey Humpherys, Tyler J. Jarvis, and Sean Wade. 2017. Forward thinking: Building and training neural networks one layer at a time. arXiv preprint arXiv:1706.02480 (2017).

[47] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[48] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015).

[49] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017).

[50] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Multi-scale dense networks for resource efficient image classification. arXiv preprint arXiv:1703.09844 (2017).

[51] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Honolulu, HI, USA, 4700–4708.

[52] Fatih Ilhan, Ka-Ho Chow, Sihao Hu, Tiansheng Huang, Selim Tekin, Wenqi Wei, Yanzhao Wu, Myungjin Lee, Ramana Kompella, Hugo Latapie, Gaowen Liu, and Ling Liu. 2024. Adaptive deep neural network inference optimization with EENet. In *Proceedings of the 2024 IEEE Winter Conference on Applications of Computer Vision, WACV 2024*. IEEE/CVF, Waikoloa, Hawaii, 10.

[53] Fatih Ilhan, Selim Furkan Tekin, Sihao Hu, Tiansheng Huang, Ka-Ho Chow, and Ling Liu. 2023. Hierarchical deep neural network inference for device-edge-cloud systems. In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion)*. Association for Computing Machinery, New York, NY, USA, 302–305.

[54] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*. PMLR, Lille, France, 448–456.

[55] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Salt Lake City, UT, USA, 2704–2713.

[56] Jithin Jagannath, Nicholas Polosky, Anu Jagannath, Francesco Restuccia, and Tommaso Melodia. 2019. Machine learning for wireless communications in the internet of things: A comprehensive survey. *Ad Hoc Networks* 93 (2019), 101913.

[57] Junguang Jiang, Ximei Wang, Mingsheng Long, and Jianmin Wang. 2020. Resource efficient domain adaptation. In *Proceedings of the 28th ACM International Conference on Multimedia (MM '20)*. Association for Computing Machinery, New York, NY, USA, 2220–2228.

[58] J. Jo, G. Kim, S. Kim, and J. Park. 2023. LoCoExNet: Low-cost early exit network for energy efficient CNN accelerator design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 12(2023), 4909–4921.

[59] Weiyu Ju, Wei Bao, Liming Ge, and Dong Yuan. 2021. Dynamic early exit scheduling for deep neural network inference through contextual bandits. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*. ACM, Gold Coast, Queensland, Australia, 823–832.

[60] Tejas Kannan, Nick Feamster, and Henry Hoffmann. 2023. Prediction privacy in distributed multi-exit neural networks: Vulnerabilities and solutions. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. Association for Computing Machinery, New York, NY, USA, 1123–1137.

[61] Polina Karpikova, Ekaterina Radionova, Anastasia Yaschenko, Andrei Spiridonov, Leonid Kostyushko, Riccardo Fabbricatore, and Aleksei Ivakhnenko. 2023. FIANCEE: Faster inference of adversarial networks via conditional early exits. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, New Orleans, LA, USA, 12032–12043.

[62] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *Proceedings of the International Conference on Machine Learning*. PMLR, Long Beach, CA, USA, 3301–3310.

[63] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, Vol. 1. ACL, Minneapolis, MN, USA, 2.

[64] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. In *Proceedings of the ICLR 2016-4th International Conference on Learning Representations*. International Conference on Learning Representations (ICLR), San Juan, Puerto Rico.

[65] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Communications of the ACM* 60, 6 (2017), 84–90.

[66] Assaf Lahiany and Yehudit Aperstein. 2022. PTEENet: Post-trained early-exit neural networks augmentation for inference cost optimization. *IEEE Access* 10 (2022), 69680–69687.

[67] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *Proceedings of the International Conference on Learning Representations*. OpenReview.net, 10.

[68] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. 2009. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research* 10, 1 (2009), 1–40.

[69] Stefanos Laskaridis, Alexandros Kouris, and Nicholas D. Lane. 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)*. Association for Computing Machinery, New York, NY, USA, 1–6.

[70] Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D. Lane. 2020. SPINN: Synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20)*. Association for Computing Machinery, London, United Kingdom, Article 37, 15 pages.

[71] E. Lattanzi, C. Contoli, and V. Freschi. 2023. Do we need early exit networks in human activity recognition?*Engineering Applications of Artificial Intelligence* 121 (2023), 106035.

[72] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.

[73] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. 2002. Efficient backprop. In *Proceedings of the Neural Networks: Tricks of the Trade*. Springer, New York, NY, USA, 9–50.

[74] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. 2015. Deeply-supervised nets. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, Guy Lebanon and S. V. N. Vishwanathan (Eds.). Proceedings of Machine Learning Research, Vol. 38, PMLR, San Diego, CA, USA, 562–570.

[75] Ilias Leontiadis, Stefanos Laskaridis, Stylianos I. Venieris, and Nicholas D. Lane. 2021. It's always personal: Using early exits for efficient on-device CNN personalisation. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications (HotMobile'21)*. Association for Computing Machinery, New York, NY, USA, 15–21.

[76] Sam Leroux, Steven Bohez, Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. 2017. The cascading neural network: Building the internet of smart things. *Knowledge and Information Systems* 52, 3 (2017), 791–814.

[77] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. 2019. Improved techniques for training adaptive deep networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. IEEE, Seoul, Korea (South), 1891–1900.

[78] Shuang Li, Jinming Zhang, Wenxuan Ma, Chi Harold Liu, and Wei Li. 2021. Dynamic domain adaptation for efficient inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Nashville, TN, USA, 7832–7841.

[79] X. Li, Y. Shen, A. Zou, and Y. Ma. 2023. EENet: Energy efficient neural networks with run-time power management. In *Proceedings of the 2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, San Francisco, CA, USA, 1–6.

[80] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. 2019. Edge computing for autonomous driving: Opportunities and challenges. *Proc. IEEE* 107, 8 (2019), 1697–1716.

[81] Yuyang Li, Yawen Wu, Xincheng Zhang, Jingtong Hu, and Inhee Lee. 2022. Energy-aware adaptive multi-exit neural network inference implementation for a millimeter-scale sensing system. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 7 (2022), 849–859.

[82] Sen Lin, Zhi Zhou, Zhaofeng Zhang, Xu Chen, and Junshan Zhang. 2021. On-demand accelerating deep neural network inference via edge computing. In *Proceedings of the Edge Intelligence in the Making: Optimization, Deep Learning, and Applications*. Springer, New York, NY, USA, 151–168.

[83] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*. OpenReview, New Orleans, LA, USA, 13.

[84] Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. FastBERT: A self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). ACL, Online, 6035–6044.

[85] Xin Liu, Qingcai Chen, Chong Deng, Huajun Zeng, Jing Chen, Dongfang Li, and Buzhou Tang. 2018. Lcqmc: A large-scale chinese question matching corpus. In *Proceedings of the 27th International Conference on Computational Linguistics*. ACL, Santa Fe, NM, USA, 1952–1962.

[86] Xianggen Liu, Lili Mou, Haotian Cui, Zhengdong Lu, and Sen Song. 2020. Finding decision jumps in text classification. *Neurocomputing* 371, C (2020), 177–187.

[87] Yuanxia Liu, Tianyong Hao, Hai Liu, Yuanyuan Mu, Heng Weng, and Fu Lee Wang. 2023. OdeBERT: One-stage Deep-supervised early-exiting BERT for fast inference in user intent classification. *ACM Transactions on Asian and Low-Resource Language Information Processing* 22, 5(2023), 1–18.

[88] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. arXiv preprint arXiv:1907.11692 (2019).

[89] Chi Lo, Yu-Yi Su, Chun-Yi Lee, and Shih-Chieh Chang. 2017. A dynamic deep neural network design for efficient workload allocation in edge computing. In *Proceedings of the 2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, Boston, MA, USA, 273–280.

[90] Pavel Mach and Zdenek Becvar. 2017. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys and Tutorials* 19, 3 (2017), 1628–1656.

[91] Enrique S. Marquez, Jonathon S. Hare, and Mahesan Niranjan. 2018. Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems* 29, 11 (2018), 5475–5485.

[92] Yoshitomo Matsubara and Marco Levorato. 2021. Neural compression and filtering for edge-assisted real-time object detection in challenged networks. In *Proceedings of the 2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, Milan, Italy, 2272–2279.

[93] Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. 2022. Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys* 55, 5 (2022), 1–30.

[94] T. Moon, M. Choi, E. Yun, J. Yoon, G. Lee, and J. Lee. 2023. Early exiting for accelerated inference in diffusion models. In *Proceedings of the ICML 2023 Workshop on Structured Probabilistic Inference and Generative Modeling*. PMLR, Honolulu, HI, USA, 14.

[95] Ravi Teja Mullapudi, William R. Mark, Noam Shazeer, and Kayvon Fatahalian. 2018. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Salt Lake City, UT, USA, 8080–8089.

[96] Feng Nan and Venkatesh Saligrama. 2017. Adaptive classification for prediction under a budget. In *Proceedings of the Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Vol. 30, Curran Associates, Inc., Long Beach, CA, USA.

[97] Katayoun Neshatpour, Farnaz Behnia, Houman Homayoun, and Avesta Sasan. 2019. Exploiting energy-accuracy tradeoff through contextual awareness in multi-stage convolutional neural networks. In *Proceedings of the 20th International Symposium on Quality Electronic Design (ISQED)*. IEEE, Santa Clara, CA, USA, 265–270.

[98] Roberto G. Pacheco, Kaylani Bochie, Mateus S. Gilbert, Rodrigo S. Couto, and Miguel Elias M. Campista. 2021. Towards edge computing using early-exit convolutional neural networks. *Information* 12, 10 (2021), 431.

[99] Roberto G. Pacheco and Rodrigo S. Couto. 2020. Inference time optimization using BranchyNet partitioning. In *Proceedings of the 2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Rennes, France, 1–6.

[100] Roberto G. Pacheco, Fernanda DVR Oliveira, and Rodrigo S. Couto. 2021. Early-exit deep neural networks for distorted images: Providing an efficient edge offloading. In *Proceedings of the 2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, Madrid, Spain, 1–6.

[101] Roberto G. Pacheco, Mark Shifrin, Rodrigo S. Couto, Daniel S. Menasché, Manjesh K. Hanawal, and Miguel Elias M. Campista. 2023. AdaEE: Adaptive early-exit DNN inference through multi-armed bandits. In *Proceedings of the ICC 2023-IEEE International Conference on Communications*. IEEE, Rome, Italy, 3726–3731.

[102] Ram Prasad Padhy, Sachin Verma, Shahzad Ahmad, Suman Kumar Choudhury, and Pankaj Kumar Sa. 2018. Deep neural network for autonomous uav navigation in indoor corridor environments. *Procedia Computer Science* 133 (2018), 643–650.

[103] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. 2016. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *Proceedings of the 2016 Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, Dresden, Germany, 475–480.

[104] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. 2017. Energy-efficient and improved image recognition with conditional deep learning. *ACM Journal on Emerging Technologies in Computing Systems* 13, 3 (2017), 1–21.

[105] Jihong Park, Sumudu Samarakoon, Mehdi Bennis, and Mérouane Debbah. 2019. Wireless network intelligence at the edge. *Proceedings of the IEEE* 107, 11 (2019), 2204–2239.

[106] Mary Phuong and Christoph H. Lampert. 2019. Distillation-based training for multi-exit architectures. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. IEEE/CVF, Seoul, South Korea, 1355–1364.

[107] Yuanyuan Qiu, Hongzheng Li, Shen Li, Yingdi Jiang, Renfen Hu, and Lijiao Yang. 2018. Revisiting correlations between intrinsic and extrinsic evaluations of word embeddings. In *Proceedings of the Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*, Maosong Sun, Ting Liu, Xiaojie Wang, Zhiyuan Liu, and Yang Liu (Eds.). Springer International Publishing, Cham, 209–221.

[108] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. ACL, Austin, TX, USA, 2383–2392.

[109] Francesco Restuccia and Tommaso Melodia. 2020. Deep learning at the physical layer: System challenges and applications to 5G and beyond. *IEEE Communications Magazine* 58, 10 (2020), 58–64.

[110] Amin Sabet, Jonathon Hare, Bashir Al-Hashimi, and Geoff V. Merrett. 2021. Temporal early exits for efficient video object detection. arXiv preprint arXiv:2106.11208 (2021).

[111] Simone Scardapane, Danilo Comminiello, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. 2020. Differentiable branching in deep networks for fast inference. In *Proceedings of the ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Barcelona, Spain, 4167–4171.

[112] Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. 2020. Why should we add early exits to neural networks?*Cognitive Computation* 12, 5 (2020), 954–966.

[113] Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. The right tool for the job: Matching model and instance complexities. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). ACL, Online, 6640–6651.

[114] Y. Sepehri, P. Pad, A. C. Yüzügüler, P. Frossard, and L. A. Dunbar. 2024. Hierarchical training of deep neural networks using early exiting. *IEEE Transactions on Neural Networks and Learning Systems* TNNLS.2024.3396628 (2024), 15.

[115] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2017. Reasonet: Learning to stop reading in machine comprehension. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Halifax, Nova Scotia, Canada, 1047–1055.

[116] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

[117] Luca Soldaini and Alessandro Moschitti. 2020. The cascade transformer: An application for efficient answer sentence selection. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). ACL, Online, 5697–5708.

[118] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[119] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. arXiv preprint arXiv:1505.00387 (2015).

[120] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE/CVF, Long Beach, CA, USA, 2820–2828.

[121] Shengkun Tang, Yaqing Wang, Caiwen Ding, Yi Liang, Yao Li, and Dongkuan Xu. 2023. DeeDiff: Dynamic uncertainty-aware early exiting for accelerating diffusion model generation. arXiv preprint arXiv:2309.17074 (2023).

[122] Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. 2019. Adaptive neural trees. In *Proceedings of the International Conference on Machine Learning*. PMLR, Long Beach, CA, USA, 6166–6175.

[123] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *Proceedings of the 2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, Cancun, Mexico, 2464–2469.

[124] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2017. Distributed deep neural networks over the cloud, the edge and end devices. In *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, Atlanta, GA, USA, 328–339.

[125] Abhishek Vashist, Sharan Vidash Vidya Shanmugham, Amlan Ganguly, and Sai Manoj P. D. 2022. DQN based exit selection in multi-exit deep neural networks for applications targeting situation awareness. In *Proceedings of the 2022 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, Las Vegas, NV, USA, 1–6.

[126] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., Long Beach, CA, USA.

[127] Andreas Veit and Serge Belongie. 2018. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, Munich, Germany, 3–18.

[128] Swagath Venkataramani, Abhishek Raghunathan, Jie Liu, and Muhammad Shoaib. 2015. Scalable-effort classifiers for energy-efficient machine learning. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, San Francisco, CA, USA, 1–6.

[129] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. 2018. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience* 2018, 1 (2018), 7068349.

[130] Tai Vu, Emily Wen, and Roy Nehoran. 2020. How not to give a FLOP: Combining regularization and pruning for efficient inference. arXiv preprint arXiv:2003.13593 (2020).

[131] Guangcong Wang, Xiaohua Xie, Jianhuang Lai, and Jiaxuan Zhuo. 2017. Deep growing learning. In *Proceedings of the IEEE International Conference on Computer Vision*. IEEE, Venice, Italy, 2812–2820.

[132] Meiqi Wang, Jianqiao Mo, Jun Lin, Zhongfeng Wang, and Li Du. 2019. Dynexit: A dynamic early-exit strategy for deep residual networks. In *Proceedings of the 2019 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, Nanjing, China, 178–183.

[133] Q. Wang, W. Fang, and N. N. Xiong. 2024. TLEE: Temporal-wise and layer-wise early exiting network for efficient video recognition on edge devices. *IEEE Internet of Things Journal* 11, 2(2024), 2842–2854.

[134] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, Fisher Yu, and Joseph E. Gonzalez. 2018. Idk cascades: Fast deep learning by learning not to overthink. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI)*. AUAI Press, Monterey, CA, USA, 11.

[135] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, Munich, Germany, 409–424.

[136] Yue Wang, Jianghao Shen, Ting-Kuei Hu, Pengfei Xu, Tan Nguyen, Richard Baraniuk, Zhangyang Wang, and Yingyan Lin. 2020. Dual dynamic inference: Enabling more efficient, adaptive, and controllable deep inference. *IEEE Journal of Selected Topics in Signal Processing* 14, 4 (2020), 623–633.

[137] W. Wenjian, X. Qian, X. Jun, and H. Zhikun. 2023. DynamicSleepNet: A multi-exit neural network with adaptive inference time for sleep stage classification. *Frontiers in Physiology* 14 (2023), 1171467.

[138] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. 2018. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Salt Lake City, UT, USA, 8817–8826.

[139] Ji Xin, Rodrigo Nogueira, Yaoliang Yu, and Jimmy Lin. 2020. Early exiting BERT for efficient document ranking. In *Proceedings of the SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*. ACL, Online, 83–88.

[140] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). ACL, Online, 2246–2251.

[141] Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021. BERxiT: Early exiting for BERT with better fine-tuning and extension to regression. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. ACL, Online, 91–104.

[142] Qunliang Xing, Mai Xu, Tianyi Li, and Zhenyu Guan. 2020. Early exit or not: Resource-efficient blind quality enhancement for compressed images. In *Proceedings of the Computer Vision—ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 275–292.

[143] Guanyu Xu, Jiawei Hao, Li Shen, Han Hu, Yong Luo, Hui Lin, and Jialie Shen. 2023. LGViT: Dynamic early exiting for accelerating vision transformer. In *Proceedings of the 31st ACM International Conference on Multimedia (MM'23)*. Association for Computing Machinery, New York, NY, USA, 9103–9114.

[144] Alexandru Rancea, Ionut Anghel, and Tudor Cioara. 2024. Edge Computing in Healthcare: Innovations, Opportunities, and Challenges. *Future Internet* 16, 9 (2024), 329.

[145] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. 2020. Resolution adaptive networks for efficient inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE/CVF, Seattle, WA, USA, 2369–2378.

[146] L. Yang, Z. Zheng, J. Wang, S. Song, G. Huang, and F. Li. 2024. AdaDet: An adaptive object detection system based on early-exit neural networks. *IEEE Transactions on Cognitive and Developmental Systems* 16, 1(2024), 332–345.

[147] Adams Wei Yu, Hongrae Lee, and Quoc Le. 2017. Learning to skim text. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL, Vancouver, Canada, 1880–1890.

[148] Haitao Yu, Dongliang Liu, Zhen Zhang, and Jiang Wang. 2023. A dynamic transformer network with early exit mechanism for fast detection of multiscale surface defects. *IEEE Transactions on Instrumentation and Measurement* 72 (2023), 1–10.

[149] Liekang Zeng, En Li, Zhi Zhou, and Xu Chen. 2019. Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial Internet of Things. *IEEE Network* 33, 5 (2019), 96–103.

[150] J. Zhang, M. Tan, P. Dai, and W. Zhu. 2023. Leco: Improving early exiting via learned exits and comparison-based exiting mechanism. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*. ACL, Stroudsburg, PA, USA, 298–309.

[151] Menglei Zhang, Michele Polese, Marco Mezzavilla, Jing Zhu, Sundeep Rangan, Shivendra Panwar, and Michele Zorzi. 2019. Will TCP work in mmWave 5G cellular networks?*IEEE Communications Magazine* 57, 1 (2019), 65–71.

[152] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Salt Lake City, UT, USA, 586–595.

[153] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of the Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.). Vol. 28. Curran Associates, Inc., Montreal, Quebec, Canada.

[154] Shaojun Zhang, Wei Li, Yongwei Wu, Paul Watson, and Albert Zomaya. 2018. Enabling edge intelligence for activity recognition in smart homes. In *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 228–236.

[155] Zhihe Zhao, Kai Wang, Neiwen Ling, and Guoliang Xing. 2021. Edgeml: An automl framework for real-time deep learning on the edge. In *Proceedings of the International Conference on Internet-of-things Design and Implementation*. IEEE, Piscataway, NJ, USA, 133–144.

[156] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems* 33 (2020), 18330–18341.

[157] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE* 107, 8 (2019), 1738–1762.

[158] Menglong Zhu and Andrey Zhmoginov Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Salt Lake City, UT, USA, 4510–4520.

[159] Wei Zhu, Xiaoling Wang, Yuan Ni, and Guotong Xie. 2021. GAML-BERT: Improving BERT early exiting by gradient aligned mutual learning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL, Online and Punta Cana, Dominican Republic, 3033–3044.

Received 27 November 2022; revised 18 September 2024; accepted 25 September 2024