



Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges

YOSHITOMO MATSUBARA and MARCO LEVORATO, University of California, Irvine, USA
FRANCESCO RESTUCCIA, Northeastern University, USA

Mobile devices such as smartphones and autonomous vehicles increasingly rely on deep neural networks (DNNs) to execute complex inference tasks such as image classification and speech recognition, among others. However, continuously executing the entire DNN on mobile devices can quickly deplete their battery. Although task offloading to cloud/edge servers may decrease the mobile device's computational burden, erratic patterns in channel quality, network, and edge server load can lead to a significant delay in task execution. Recently, approaches based on **split computing (SC)** have been proposed, where the DNN is split into a head and a tail model, executed respectively on the mobile device and on the edge server. Ultimately, this may reduce bandwidth usage as well as energy consumption. Another approach, called **early exiting (EE)**, trains models to embed multiple "exits" earlier in the architecture, each providing increasingly higher target accuracy. Therefore, the tradeoff between accuracy and delay can be tuned according to the current conditions or application demands. In this article, we provide a comprehensive survey of the state of the art in **SC** and **EE** strategies by presenting a comparison of the most relevant approaches. We conclude the article by providing a set of compelling research challenges.

CCS Concepts: • **Human-centered computing** → Ubiquitous and mobile computing; • **Computer systems organization** → *Embedded and cyber-physical systems*; • **Computing methodologies** → Neural networks;

Additional Key Words and Phrases: Split computing, edge computing, early exit, neural networks, deep learning

ACM Reference format:

Yoshitomo Matsubara, Marco Levorato, and Francesco Restuccia. 2022. Split Computing and Early Exiting for Deep Learning Applications: Survey and Research Challenges. *ACM Comput. Surv.* 55, 5, Article 90 (December 2022), 30 pages.

<https://doi.org/10.1145/3527155>

1 INTRODUCTION

The field of **deep learning (DL)** has evolved at an impressive pace over the last few years [68], with new breakthroughs continuously appearing in domains such as **computer vision (CV)**, **natural language processing (NLP)**, **digital signal processing (DSP)**, and wireless networking [56, 118],

This work was partially supported by the NSF under grant IIS-1724331, MLWiNS-2003237, CCF-2140154, and CNS-2134567. Authors' addresses: Y. Matsubara and M. Levorato, University of California, Irvine, Irvine, CA, 92697; emails: {yoshitom, levorato}@uci.edu; F. Restuccia, Northeastern University, 360 Huntington Ave., Boston, MA; email: f.restuccia@northeastern.edu.



This work is licensed under a Creative Commons Attribution-NoDerivs International 4.0 License.

© 2022 Copyright held by the owner/author(s).

0360-0300/2022/12-ART90

<https://doi.org/10.1145/3527155>

among others—we refer to [110] for a comprehensive survey on **DL**. For example, today’s state-of-the-art deep neural networks (DNNs) can classify thousands of images with unprecedented accuracy [51], while bleeding-edge advances in deep reinforcement learning (DRL) have shown to provide near-human capabilities in a multitude of complex optimization tasks, from playing dozens of Atari video games [99] to winning games of Go against top-tier players [127].

As **DL**-based classifiers improve their predictive accuracy, mobile applications such as speech recognition in smartphones [20, 45], real-time unmanned navigation [105], and drone-based surveillance [129, 170] are increasingly using DNNs to perform complex inference tasks. However, *state-of-the-art DNN models present computational requirements that cannot be satisfied by the majority of the mobile devices available today*. In fact, many state-of-the-art DNN models for difficult tasks—such as computer vision and natural language processing—are extremely complex. For instance, the EfficientDet [139] family offers the best performance for object detection tasks. While EfficientDet-D7 achieves a **mean average precision (mAP)** of 52.2%, it involves 52M parameters and will take seconds to be executed on strong embedded devices equipped with GPUs such as the NVIDIA Jetson Nano and Raspberry Pi. Notably, the execution of such complex models significantly increases energy consumption. While lightweight models specifically designed for mobile devices exist [122, 138], the reduced computational burden usually comes to the detriment of the model accuracy. For example, compared to ResNet-152 [43], the networks MnasNet [138] and MobileNetV2 [122] present up to 6.4% accuracy loss on the ImageNet dataset. YOLO-Lite [116] achieves a frame rate of 22 frames per second on some embedded devices but has a mAP of 12.36% on the COCO dataset [83]. To achieve 33.8% mAP on the COCO dataset, even the simplest model in the EfficientDet family, EfficientDet-D0, requires 3 times more FLOPs (2.5B)¹ than SSD-MobileNetV2 [122] (0.8B FLOPs). While SSD-MobileNetV2 is a lower-performance DNN specifically designed for mobile platforms and can process up to 6 fps, its mAP on the COCO dataset is 20%, and keeping the model running on a mobile device significantly increases power consumption. On the other hand, due to excessive end-to-end latency, cloud-based approaches are hardly applicable in most of the latency-constrained applications where mobile devices usually operate. Most of the techniques we overview in the survey can be applied to both mobile-device-to-edge-server and edge-server-to-cloud offloading. For the sake of clarity, we primarily refer to the former to explain the frameworks.

Recently, **edge computing (EC)** approaches [10, 88] have attempted to address the “latency vs. computation” conundrum by completely offloading the DNN execution to servers located very close to the mobile device, i.e., at the “edge” of the network. However, canonical **EC** does not consider that the quality of wireless links—although providing high throughput on average—can suddenly fluctuate due to the presence of erratic noise and interference patterns, which may impair performance in latency-bound applications. For example, mobility and impaired propagation have been shown to decrease throughput even in high-bandwidth wireless links [89, 169], while many **Internet of Things (IoT)** systems are based on communication technologies such as **Long Range (LoRa)** [121], which has a maximum data rate of 37.5 Kbps due to duty cycle limitations [1].

The severe offloading limitations of some mobile devices, coupled with the instability of the wireless channel (e.g., UAV network [36]), imply that the amount of data offloaded to edge should be decreased, while at the same time keeping the model accuracy as close as possible to the original. For this reason, **split computing (SC)** [60] and **early exiting (EE)** strategies [140] have been proposed to provide an intermediate option between **EC** and local computing. The key intuition behind **SC** and **EE** is similar to the one behind model pruning [38, 44, 74, 160] and knowledge distillation [46, 61, 98]—since modern DNNs are heavily over-parameterized [165, 166], their

¹In Tan et al. [139], FLOP denotes number of multiply-adds.

accuracy can be preserved even with substantial reduction in the number of weights and filters, thus representing the input with fewer parameters. Specifically, **SC** divides a larger DNN into head and tail models, which are respectively executed by the mobile device and edge server. **EE**, on the other hand, proposes the introduction of “subbranches” into the early layers of DNN models, so that the full computation of the model can be halted—and a prediction result provided—if the classifiers in the current subbranches have high confidence with the specific model input.

Motivation and Novel Contributions. The proliferation of DL-based mobile applications in the IoT and 5G landscapes implies that techniques such as **SC** and **EE** are not simply “nice-to-have” features, but will become fundamental computational components in the years to come. Although a significant amount of research work has been done in **SC** and **EE**, to the best of our knowledge, a comprehensive survey of the state of the art has not been conducted yet. Moreover, there are still a series of research challenges that need to be addressed to take **SC** and **EE** to the next level. For this reason, this article makes the following novel contributions:

- We summarize **SC** and **EE** studies with respect to approaches, tasks, and models. We first provide an overview of local, edge, split computing, and early-exit models in Section 2 by highlighting similarities and difference among them.
- We then discuss and compare the various approaches to **SC** and **EE** in Sections 4 and 5 by highlighting the training strategies and applications. Since code availability is fundamental for replicability/reproducibility [34],² we provide for each work its corresponding code repository, if available, so that interested readers can reproduce and learn from existing studies.
- We conclude the article by discussing in Section 6 a compelling agenda of research challenges in **SC** and **EE**, hoping to spur further contributions in these exciting and timely fields.

2 OVERVIEW OF LOCAL, EDGE, SPLIT COMPUTING, AND EARLY-EXIT MODELS

In this section, we provide an overview of local, edge, split computing, and early-exit models, which are the main computational paradigms that will be discussed in the article. Figure 1 provides a graphical overview of the approaches.

All these techniques operate on a DNN model $\mathcal{M}(\cdot)$ whose task is to produce the inference output y from an input x . Typically, x is a high-dimensional variable, whereas the output y has significantly lower dimensionality [143]. Split computing and early-exit approaches are contextualized in a setting where the system is composed of a mobile device and an edge server interconnected via a wireless channel. The overall goal of the system is to produce the inference output y from the input x acquired by the mobile device, by means of the DNN $y = \mathcal{M}(x)$ under—possibly time varying—constraints on:

Resources: (1) the computational capacity (roughly expressed as number operations per second) C_{md} and C_{es} of the mobile device and edge server, respectively, and (2) the capacity ϕ , in bits per second, of the wireless channel connecting the mobile device to the edge server

Performance: (1) the absolute of average value of the time from the generation of x to the availability of y , and (2) the degradation of the “quality” of the output y .

Split, edge, local, and early exiting strategies strive to find suitable operating points with respect to accuracy, end-to-end delay, and energy consumption, which are inevitably influenced by

²To address this problem, major machine learning venues (e.g., ICML, NeurIPS, CVPR, ECCV, NAACL, ACL, and EMNLP) adopt a reproducibility checklist as part of the official review process such as the ML Code Completeness Checklist. See <https://github.com/paperswithcode/releasing-research-code>.

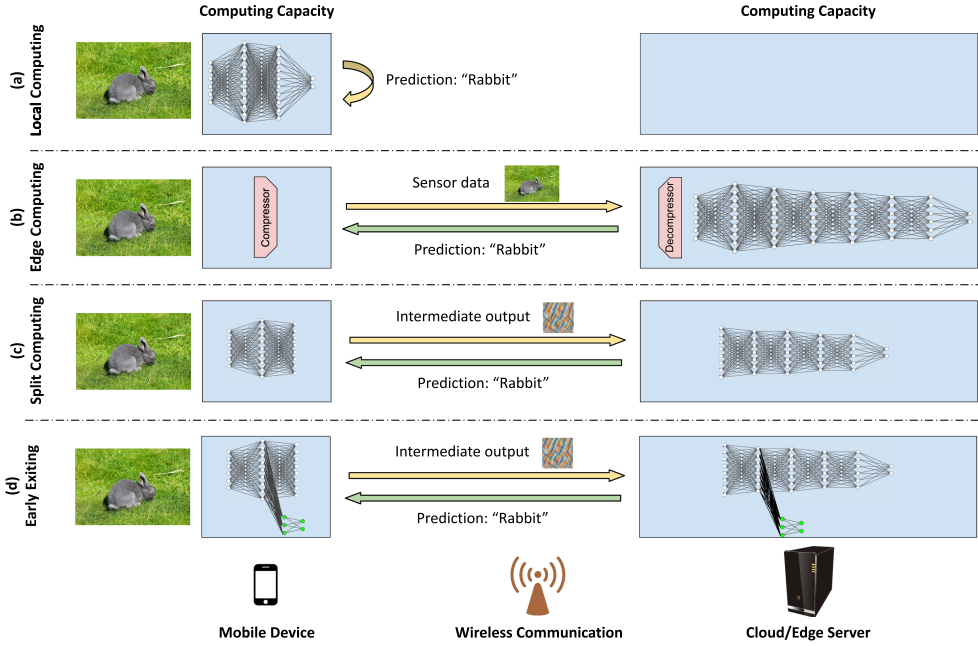


Fig. 1. Overview of (a) local, (b) edge, (c) split computing, and (d) early exiting: image classification as an example.

the characteristics of the underlying system. It is generally assumed that the computing and energy capacities of the mobile device are smaller than that of the edge server. As a consequence, if part of the workload is allocated to the mobile device, then the execution time increases, while the battery lifetime decreases. However, as explained later, the workload executed by the mobile device may result in a reduced amount of data to be transferred over the wireless channel, possibly compensating for the larger execution time and leading to smaller end-to-end delays.

2.1 Local and Edge Computing

We start with an overview of local and edge computing. In **local computing (LC)**, the function $\mathcal{M}(\mathbf{x})$ is entirely executed by the mobile device. This approach eliminates the need to transfer data over the wireless channel. However, the complexity of the best-performing DNNs most likely exceeds the computing capacity and energy consumption available at the mobile device. Usually, simpler models $\hat{\mathcal{M}}(\mathbf{x})$ are used, such as MobileNet [122] and MnasNet [138], which often have a degraded accuracy performance. Besides designing lightweight neural models executable on mobile devices, the widely used techniques to reduce the complexity of models are knowledge distillation [46] and model pruning/quantization [55, 73], described in Section 3.2. Some of the techniques are also leveraged in **SC** studies to introduce bottlenecks without sacrificing model accuracy, as will be described in the following sections.

In **EC**, the input \mathbf{x} is transferred to the edge server, which then executes the original model $\mathcal{M}(\mathbf{x})$. In this approach, which preserves full accuracy, the mobile device is not allocated computing workload, but the full input \mathbf{x} needs to be transferred to the edge server. This may lead to an excessive end-to-end delay in degraded channel conditions and erasure of the task in extreme conditions. A possible approach to reduce the load imposed to the wireless channel, and thus also transmission delay and erasure probability, is to compress the input \mathbf{x} . We define, then, the encoder and decoder models $\mathbf{z} = F(\mathbf{x})$ and $\hat{\mathbf{x}} = G(\mathbf{z})$, which are executed at the mobile device and

edge server, respectively. The distance $d(\mathbf{x}, \hat{\mathbf{x}})$ defines the performance of the encoding-decoding process $\hat{\mathbf{x}} = G(F(\mathbf{x}))$, a metric that is separate from, but may influence, the accuracy loss of $\mathcal{M}(\hat{\mathbf{x}})$ with respect to $\mathcal{M}(\mathbf{x})$, that is, of the model executed with the reconstructed input with respect to the model executed with the original input. Clearly, the encoding/decoding functions increase the computing load at both the mobile device and edge server side. A broad range of different compression approaches exists ranging from low-complexity traditional compression (e.g., JPEG compression for images in EC [101]) to neural compression models [4, 5, 162]. We remark that while the compressed input data, e.g., JPEG objects, can reduce the data transfer time in EC, those representations are designed to allow the accurate reconstruction of the input signal. Therefore, these approaches may (1) decrease privacy as a “reconstructable” representation is transferred to the edge server [147] and (2) result in a larger amount of data to be transmitted over the channel compared to representation specifically designed for the computing task as in bottleneck-based SC, as explained in the following sections.

2.2 Split Computing and Early Exiting

SC aims at achieving the following goals: (1) the computing load is distributed across the mobile device and edge server and (2) establishes a task-oriented compression to reduce data transfer delays. We consider a neural model $\mathcal{M}(\cdot)$ with L layers, and define \mathbf{z}_ℓ as the output of the ℓ th layer. Early implementations of SC select a layer ℓ and divide the model $\mathcal{M}(\cdot)$ to define the head and tail submodels $\mathbf{z}_\ell = \mathcal{M}_H(\mathbf{x})$ and $\hat{\mathbf{y}} = \mathcal{M}_T(\mathbf{z}_\ell)$, executed at the mobile device and edge server, respectively. In early instances of SC, the architecture and weights of the head and tail model are exactly the same as the first ℓ layers and last $L - \ell$ layers of $\mathcal{M}(\cdot)$. This simple approach preserves accuracy but allocates part of the execution of $\mathcal{M}(\cdot)$ to the mobile device, whose computing power is expected to be smaller than that of the edge server, so that the total execution time may be larger. The transmission time of \mathbf{z}_ℓ may be larger or smaller compared to that of transmitting the input \mathbf{x} , depending on the size of the tensor \mathbf{z}_ℓ . However, we note that in most relevant applications the size of \mathbf{z}_ℓ becomes smaller than that of \mathbf{x} only in later layers, which would allocate most of the computing load to the mobile device. More recent SC frameworks introduce the notion of *bottleneck* to achieve *in-model* compression toward the global task [90]. As formally described in the next section, a bottleneck is a compression point at one layer in the model, which can be realized by reducing the number of nodes of the target layer and/or by quantizing its output. We note that as SC realizes a task-oriented compression, it guarantees a higher degree of privacy compared to EC. In fact, the representation may lack information needed to fully reconstruct the original input data.

Another approach to enable mobile computing is referred to as EE. The core idea is to create models with multiple “exits” across the model, where each exit can produce the model output. Then, the first exit providing a target confidence on the output is selected. This approach tunes the computational complexity, determined by the exit point, to the sample or to system conditions. Formally, we can define a sequence of models \mathcal{M}_i and \mathcal{B}_i , $i = 1, \dots, N$. Model \mathcal{M}_i takes as input \mathbf{z}_{i-1} (the output of model \mathcal{M}_{i-1}) and outputs \mathbf{z}_i , where we set $\mathbf{z}_0 = \mathbf{x}$. The branch models \mathcal{B}_i take as input \mathbf{z}_i and produce the estimate of the desired output \mathbf{y}_i . Thus, the concatenation of $\mathcal{M}_1, \dots, \mathcal{M}_N$ results in an output analogous to that of the original model. Intuitively, the larger the number of models used to produce the output \mathbf{y}_i , the better the accuracy. Thus, while SC optimizes intermediate representations to preserve information toward the final task (e.g., classification) for the whole dataset, early-exit models take a “per sample” control perspective. Each sample will be sequentially analyzed by concatenations of \mathcal{M}_i and \mathcal{B}_i sections until a predefined confidence level is reached. The hope is that a portion of the samples will require a smaller number of sections compared to executing the whole sequence.

3 BACKGROUND OF DEEP LEARNING FOR MOBILE APPLICATIONS

In this section, we provide an overview of recent approaches to reduce the computational complexity of DNN models for resource-constrained mobile devices. These approaches can be categorized into two main classes: (1) approaches that attempt to directly design lightweight models and (2) model compression.

3.1 Lightweight Models

From a conceptual perspective, the design of small deep learning models is one of the simplest ways to reduce inference cost. However, there is a tradeoff between model complexity and model accuracy, which makes this approach practically challenging when aiming at high model performance. The MobileNet series [47, 48, 122] is one among the most popular lightweight models for computer vision tasks, where Howard et al. [48] describe the first version, MobileNetV1. By using a pair of depth-wise and point-wise convolution layers in place of standard convolution layers, the design drastically reduces model size, and thus computing load. Following this study, Sandler et al. [122] proposed MobileNetV2, which achieves improved accuracy. The design is based on MobileNetV1 [48] and uses the bottleneck residual block, a resource-efficient block with inverted residuals and linear bottlenecks. Howard et al. [47] present MobileNetV3, which further improves the model accuracy and is designed by a hardware-aware neural architecture search [138] with NetAdapt [161]. The largest variant of MobileNetV3, MobileNetV3-Large 1.0, achieves a comparable accuracy to ResNet-34 [43] for the ImageNet dataset, while reducing by about 75% the model parameters.

While many of the lightweight neural networks are often manually designed, there are also studies on automating the **neural architecture search (NAS)** [173]. For instance, Zoph et al. [174] design a novel search space through experiments with the CIFAR-10 dataset [63], which is then scaled to larger, higher-resolution image datasets such as the ImageNet dataset [120], to design their proposed model: NASNet. Leveraging the concept of NAS, some studies design lightweight models in a platform-aware fashion. Dong et al. [23] propose the **Device-aware Progressive Search for Pareto-optimal Neural Architectures (DDP-Net)** framework, which optimizes the network design with respect to two objectives: device-related (e.g., inference latency and memory usage) and device-agnostic (e.g., accuracy and model size) objectives. Similarly, Tan et al. [138] propose an automated **mobile neural architecture search (MNAS)** method and design the MnasNet models by optimizing both model accuracy and inference time.

3.2 Model Compression

A different approach to produce small DNN models is to “compress” a large model. Model pruning and quantization [38, 39, 55, 79] are the dominant model compression approaches. The former removes parameters from the model, while the latter uses fewer bits to represent them. In both these approaches, a large model is trained first and then compressed, rather than directly designing a lightweight model followed by training. Jacob et al. [55] empirically show that their quantization technique leads to an improved tradeoff between inference time and accuracy on MobileNet [48] for image classification tasks on Qualcomm Snapdragon 835 and 821 compared to the original, float-only MobileNet. For what concerns model pruning, Li et al. [75] and Liu et al. [86] demonstrate that it is difficult for model pruning itself to accelerate inference while achieving strong performance guarantees on general-purpose hardware due to the unstructured sparsity of the pruned model and/or kernels in layers.

Knowledge distillation [8, 46] is another popular model compression method. While model pruning and quantization make trained models smaller, the concept of knowledge distillation is to

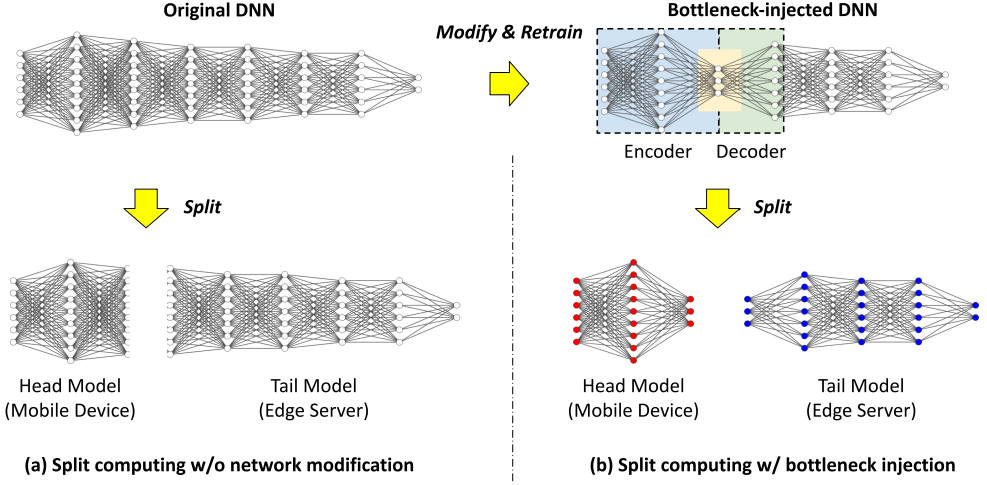


Fig. 2. Two different SC approaches.

provide outputs extracted from the trained model (called “teacher”) as informative signals to train smaller models (called “student”) in order to improve the accuracy of predesigned small models. Thus, the goal of the process is that of *distilling knowledge of a trained teacher model into a smaller student model* for boosting accuracy of the smaller model without increasing model complexity. For instance, Ba and Caruana [3] propose a method to train small neural networks by mimicking the detailed behavior of larger models. The experimental results show that models trained by this mimic learning method achieve performance close to that of deeper neural networks on some phoneme recognition and image recognition tasks. The formulation of some knowledge distillation methods will be described in Section 4.4.

4 SPLIT COMPUTING: A SURVEY

This section discusses the existing state of the art in SC. Figure 2 illustrates the existing SC approaches. They can be categorized into either (1) *without network modification* or (2) *with bottleneck injection*. We first present SC approaches without DNN modification in Section 4.1. We then discuss the motivations behind the introduction of SC with bottlenecks in Section 4.2, which are then discussed in detail in Section 4.3. Since the latter require specific training procedures, we devote Section 4.4 to their discussion.

4.1 Split Computing without DNN Modification

In this class of approaches, the architecture and weights of the head $\mathcal{M}_H(\cdot)$ and tail $\mathcal{M}_T(\cdot)$ models are exactly the same as the first ℓ layers and last $L - \ell$ layers of $\mathcal{M}(\cdot)$. To the best of our knowledge, Kang et al. [60] proposed the first SC approach (called “Neurosurgeon”), which searches for the best partitioning layer in a DNN model for minimizing total (end-to-end) latency or energy consumption. Formally, inference time in SC is the sum of processing time on a mobile device, delay of communication between a mobile device and edge server, and processing time on an edge server.

Interestingly, their experimental results show that the best partitioning (splitting) layers in terms of energy consumption and total latency for most of the considered models result in either their input or output layers. In other words, deploying the whole model on either a mobile device or an edge server (i.e., local computing or EC) would be the best option for such DNN models. Following

Table 1. Studies on **SC** without Architectural Modifications

Work	Task(s)	Dataset(s)	Model(s)	Metrics	Code
Kang et al. [60]	Image classification Speech recognition Part-of-speech tagging Named entity recognition Word chunking	N/A (No task-specific metrics)	AlexNet [64] VGG-19 [128] DeepFace [137] LeNet-5 [69] Kaldi [111] SENNA [17]	D, E, L	
Li et al. [76]	Image classification	N/A (No task-specific metrics)	AlexNet [64]	C, D	
Jeong et al. [58]	Image classification	N/A (No task-specific metrics)	GoogLeNet [135] AgeNet [72] GenderNet [72]	D, L	
Li et al. [73]	Image classification	ImageNet [120]	AlexNet [64] VGG-16 [128] ResNet-18 [43] GoogLeNet [135]	A, D, L	
Choi and Bajić [13]	Object detection	VOC 2007 [28]	YOLO9000 [115]	A, C, D, L	
Eshratifar et al. [25]	Image classification Speech recognition	N/A (No task-specific metrics)	AlexNet [64] OverFeat [125] NiN [80] VGG-16 [128] ResNet-50 [43]	D, E, L	
Zeng et al. [168]	Image classification	CIFAR-10 [63]	AlexNet [64]	A, D, L	
Cohen et al. [16]	Image classification Object detection	ImageNet (2012) [120] COCO 2017 [83]	VGG-16 [128] ResNet-50 [43] YOLOv3 [116]	A, D	
Pagliari et al. [106]	Natural language inference Reading comprehension Sentiment analysis	N/A (No task-specific metrics)	RNNs	E, L	
Itahara et al. [53]	Image classification	CIFAR-10 [63]	VGG-16 [128]	A, D	

A: Model accuracy, C: Model complexity, D: Transferred data size, E: Energy consumption, L: Latency, T: Training cost.

the work by Kang et al. [60], the research communities explored various **SC** approaches mainly focused on **CV** tasks such as image classification. Table 1 summarizes the studies on **SC** without architectural modifications.

Jeong et al. [58] used this partial offloading approach as a privacy-preserving way for computation offloading to blind the edge server to the original data captured by client. Leveraging neural network quantization techniques, Li et al. [73] discussed best splitting point in DNN models to minimize inference latency and showed that quantized DNN models did not degrade accuracy compared to the (pre-quantized) original models. Choi and Bajić [13] proposed a feature compression strategy for object detection models that introduces a quantization/video-coding-based compressor to the intermediate features in YOLO9000 [115].

Eshratifar et al. [25] propose JointDNN for collaborative computation between the mobile device and cloud and demonstrate that using either local computing only or cloud computing only is not an optimal solution in terms of inference time and energy consumption. Different from [60], they consider not only discriminative deep learning models (e.g., classifiers) but also generative deep learning models and autoencoders as benchmark models in their experimental evaluation. Cohen et al. [16] introduce a technique to code the output of the head portion in a split DNN to a wide range of bit-rates and demonstrate the performance for image classification and object detection tasks. Pagliari et al. [106] first discuss the collaborative inference for simple recurrent neural networks, and their proposed scheme is designed to automatically select the best inference device for each input data in terms of total latency or end-device energy. Itahara et al. [53] use dropout layers [133] to emulate a packet loss scenario rather than for the sake of compression and discuss the robustness of VGG-based models [128] for split computing.

Table 2. Statistics of Image Classification Datasets in SC Studies

	MNIST	CIFAR-10	CIFAR-100	ImageNet (2012)
# labeled train/dev(test) samples	60k/10k	50k/10k	50k/10k	1,281k/50k
# object categories	10	10	100	1,000
Input tensor size	$1 \times 32 \times 32$	$3 \times 32 \times 32$	$3 \times 32 \times 32$	$3 \times 224 \times 224^*$
JPEG data size [KB/sample]	0.9657	1.790	1.793	44.77

* A standard (resized) input tensor size for DNN models.

While only a few studies in Table 1 heuristically choose splitting points [13, 16], most of the other studies [25, 58, 60, 73, 76, 106, 168] in Table 1 analyze various types of cost (e.g., computational load and energy consumption on mobile device, communication cost, and/or privacy risk) to partition DNN models at each of their splitting points. Based on the analysis, performance profiles of the split DNN models are derived to inform selection. Concerning metrics, many of the studies in Table 1 do not discuss task-specific performance metrics such as accuracy. This is in part because the proposed approaches do not modify the input or intermediate representations in the models (i.e., the final prediction will not change). On the other hand, Choi and Bajić [13], Cohen et al. [16], and Li et al. [73] introduce lossy compression techniques to intermediate stages in DNN models, which may affect the final prediction results. Thus, discussing the tradeoff between compression rate and task-specific performance metrics would be essential for such studies. As shown in the table, such tradeoff is discussed only for CV tasks, and many of the models considered in such studies have weak performance compared with state-of-the-art models and complexity within reach of modern mobile devices. Specific to image classification tasks, most of the models considered in the studies listed in Table 1 are more complex and/or the accuracy is comparable to or lower than that of lightweight baseline models such as MobileNetV2 [122] and MnasNet [138]. Thus, in future work, more accurate models should be considered to discuss the performance tradeoff and further motivate SC approaches.

4.2 The Need for Bottleneck Injection

While Kang et al. [60] empirically show that executing the whole model on either the mobile device or edge server would be best in terms of total inference and energy consumption for most of their considered DNN models, their proposed approach finds the best partitioning layers inside some of their considered CV models (convolutional neural networks (CNNs)) to minimize the total inference time. There are a few trends observed from their experimental results: (1) communication delay to transfer data from the mobile device to edge server is a key component in SC to reduce total inference time; (2) all the neural models they considered for NLP tasks are relatively small (consisting of only a few layers), which potentially resulted in finding that the output layer is the best partition point (i.e., local computing) according to their proposed approach; and (3) similarly, not only DNN models they considered (except VGG [128]) but also the size of the input data to the models (see Table 2) are relatively small, which gives more advantage to EC (fully offloading computation). In other words, it highlights that complex CV tasks requiring large (high-resolution) images for models to achieve high accuracy such as ImageNet and COCO datasets would be essential to discuss the tradeoff between accuracy and execution metrics to be minimized (e.g., total latency, energy consumption) for SC studies. The key issue is that straightforward SC approaches like Kang et al. [60] rely on the existence of *natural bottlenecks*—that is, intermediate layers whose output z_ℓ tensor size is smaller than the input—inside the model. Without such natural

bottlenecks in the model, straightforward splitting approaches would fail to improve performance in most settings [6, 35].

Some models, such as AlexNet [64], VGG [128], and DenseNet [51], possess such layers [90]. However, recent DNN models such as ResNet [43], Inception-v3 [136], Faster R-CNN [117], and Mask R-CNN [42] do not have natural bottlenecks in the early layers; that is, splitting the model would result in compression only when assigning a large portion of the workload to the mobile device. As discussed earlier, reducing the communication delay is key to minimize total inference time in **SC**. For these reasons, introducing *artificial bottlenecks* to DNN models by modifying their architecture is a recent trend and has been attracting attention from the research community. Since the main role of such encoders in **SC** is to compress intermediate features rather than to complete inference, the encoders usually consist of only a few layers. Also, the resulting encoders in **SC** to be executed on constrained mobile devices are often much smaller (e.g., 10K parameters in the encoder of ResNet-based **SC** model [94]) than lightweight models such as MobileNetV2 [122] (3.5M parameters) and MnasNet [138] (4.4M parameters). Thus, even if the model accuracy is either degraded or comparable to such small models, **SC** models are still beneficial in terms of computational burden and energy consumption at the mobile devices.

4.3 Split Computing with Bottleneck Injection

This class of models can be described as composed of three sections: \mathcal{M}_E , \mathcal{M}_D , and \mathcal{M}_T . We define $\mathbf{z}_\ell|\mathbf{x}$ as the output of the ℓ th layer of the original model given the input \mathbf{x} . The concatenation of the \mathcal{M}_E and \mathcal{M}_D models is designed to produce a possibly noisy version $\hat{\mathbf{z}}_\ell|\mathbf{x}$ of $\mathbf{z}_\ell|\mathbf{x}$, which is taken as input by \mathcal{M}_T to produce the output $\hat{\mathbf{y}}$, on which the accuracy degradation with respect to \mathbf{y} is measured. The models \mathcal{M}_E , \mathcal{M}_D function as specialized encoders and decoders in the form $\hat{\mathbf{z}}_\ell = \mathcal{M}_D(\mathcal{M}_E(\mathbf{x}))$, where $\mathcal{M}_E(\mathbf{x})$ produces the latent variable \mathbf{z} . In words, the two first sections of the modified model transform the input \mathbf{x} into a version of the output of the ℓ th layer via the intermediate representation \mathbf{z} , thus functioning as encoder/decoder functions. The model is split after the first section; that is, \mathcal{M}_E is the head model, and the concatenation of \mathcal{M}_D and \mathcal{M}_T is the tail model. Then, the tensor \mathbf{z} is transmitted over the channel. The objective of the architecture is to minimize the size of \mathbf{z} to reduce the communication time while also minimizing the complexity of \mathcal{M}_E (that is, the part of the model executed at the—weaker—mobile device) and the discrepancy between \mathbf{y} and $\hat{\mathbf{y}}$. The layer between \mathcal{M}_E and \mathcal{M}_D is the injected bottleneck.

Table 3 summarizes **SC** studies with bottleneck-injected strategies. To the best of our knowledge, the papers in [26] and [90] were the first to propose altering existing DNN architectures to design relatively small bottlenecks at early layers in DNN models, instead of introducing compression techniques (e.g., quantization, autoencoder) to the models, so that communication delay (cost) and total inference time can be further reduced. Following these studies, Hu and Krishnamachari [49] introduce bottlenecks to MobileNetV2 [122] (modified for CIFAR datasets) in a similar way for **SC** and discuss end-to-end performance evaluation. Choi et al. [14] combine multiple compression techniques such as quantization and tiling besides convolution/deconvolution layers and design a feature compression approach for object detectors. Similar to the concept of bottleneck injection, Shao and Zhang [126] find that over-compression of intermediate features and inaccurate communication between computing devices can be tolerated unless the prediction performance of the models is significantly degraded by them. Also, Jankowski et al. [57] propose introducing a reconstruction-based bottleneck to DNN models, which is similar to the concept of BottleNet [26]. A comprehensive discussion on the delay/complexity/accuracy tradeoff can be found in [91, 164].

These studies are all focused on image classification. Other CV tasks present further challenges. For instance, state-of-the-art object detectors such as R-CNN models have more narrow range of layers that we can introduce bottlenecks due to the network architecture, which has multiple

Table 3. Studies on SC with Bottleneck Injection Strategies

Work	Task(s)	Dataset(s)	Base Model(s)	Training	Metrics	Code
Eshratifar et al. [26]	Image classification	minilImageNet [130]	ResNet-50 [43] VGG-16 [128]	CE-based	A, D, L	
Matsubara et al. [90, 91]	Image classification	Caltech 101 [29] ImageNet [120]	DenseNet-169 [51] DenseNet-201 [51] ResNet-152 [43] Inception-v3 [136]	HND KD CE-based	A, C, D, L, T	Link
Hu and Krishnamachari [49]	Image classification	CIFAR-10/100 [63]	MobileNetV2 [122]	CE-based	A, D, L	
Choi et al. [14]	Object detection	COCO 2014 [83]	YOLOv3 [116]	Reconstruct.	A, D	
Shao and Zhang [126]	Image classification	CIFAR-100 [63]	ResNet-50 [43] VGG-16 [128]	CE-based (Multi-stage)	A, C, D	
Jankowski et al. [57]	Image classification	CIFAR-100 [63]	VGG-16 [128]	CE + \mathcal{L}_2 (Multi-stage)	A, C, D	
Matsubara et al. [93, 94]	Object detection Keypoint detection	COCO 2017 [83]	Faster R-CNN [117] Mask R-CNN [42] Keypoint R-CNN [42]	HND GHND	A, C, D, L	Link
Yao et al. [164]	Image classification Speech recognition	ImageNet [120] LibriSpeech [107]	ResNet-50 [43] Deep Speech [40]	Reconstruct. + KD	A, D, E, L, T	Link*
Assine et al. [2]	Object detection	COCO [83]	EfficientDet [139]	GHND-based	A, C, D	Link
Sbai et al. [124]	Image classification	Subset of ImageNet [120] (700 out of 1,000 classes)	MobileNetV1 [48] VGG-16 [128]	Reconstruct. + KD	A, C, D	
Lee et al. [70]	Object detection	COCO [83]	YOLOv5 [144]	CE-based	A, C, D, L	
Matsubara et al. [92]	Image Classification	ImageNet [120]	DenseNet-169 [51] DenseNet-201 [51] ResNet-152 [43]	Reconst. HND GHND CE/KD (Multi-stage)	A, C, D, E L	Link
Matsubara et al. [96, 97]	Image Classification Object detection Semantic Segmentation	ImageNet [120] COCO [83] PASCAL VOC [27]	ResNet-50 [43] ResNet-101 [43] RegNetY-6.4GF [113] Hybrid ViT [134] RetinaNet [82] Faster R-CNN [117] DeepLabv3 [11]	GHND CE/KD+Rate (Multi-stage)	A, C, D, L	Link (2021) Link (2022)

A: Model accuracy, C: Model complexity, D: Transferred data size, E: Energy consumption, L: Latency, T: Training cost.

*The repository is incomplete and lacks instructions to reproduce the reported results for vision and speech datasets.

forward paths to forward outputs from intermediate layers to the **feature pyramid network (FPN)** [81]. The head network distillation training approach—discussed later in this section—was used by Matsubara and Levorato [94] to address some of these challenges and reduce the amount of data transmitted over the channel by 94% while degrading mAP loss by 1 point. Assine et al. [2] introduce bottlenecks to the EfficientDet-D2 [139] object detector and apply the training method based on the generalized head network distillation [94] and mutual learning [159] to the modified model. Following the studies on SC for resource-constrained edge computing systems [90, 91, 164], Sbai et al. [124] introduce autoencoder to small classifiers and train them on a subset of the ImageNet dataset in a similar manner. These studies discuss the tradeoff between accuracy and memory size on mobile devices, considering communication constraint-based 3G and LoRa technologies [121]. Similar to [2, 93, 94], Lee et al. [70] design a lightweight encoder for an object detector on the mobile device followed by both a module to amplify the compressed feature and the object detector to be executed on the edge server. Matsubara et al. [92] empirically show that bottleneck-injected models can be further improved by elaborating the methods to train the models. The resulting models outperform models with autoencoder-based feature compression (e.g., Figure 5) in terms of the tradeoff between model accuracy and transferred data size.

Matsubara et al. [97] propose a supervised compression method for resource-constrained edge computing systems, which adapts ideas from knowledge distillation and neural image compression [4, 5]. Their student model (namely, *Entropic Student*) contains a lightweight encoder with a

learnable prior, which quantizes and entropy-codes latent representations under a prior probability model for efficiently saving the size of data to be offloaded to the edge server. By adjusting a balancing weight in their loss function during training, we can control the tradeoff between data size (rate) and model accuracy (distortion). The performance of the entropic student model was demonstrated for three large-scale downstream supervised tasks: image classification (ImageNet), object detection (COCO), and semantic segmentation (COCO, PASCAL VOC). Notably, the representation produced by a single trained encoder of the entropic student model can serve multiple downstream tasks. Following the study, Matsubara et al. [96] further investigate this approach and empirically show that it generalizes to other reference models (e.g., ResNet-101 [43], RegNetY-6.4GF [113], Hybrid ViT [134]). Through experiments, the study also points out that simply introducing such bottleneck layers at later layers in a model can improve the conventional **rate-distortion (R-D)** tradeoff, which will result in most of the computational load being assigned to a weak mobile device.

In contrast to **SC** studies without bottlenecks in Table 1, many of the studies on bottleneck injection strategies in Table 3 are published with code that would help the research communities replicate/reproduce the experimental results and build on existing studies.

4.4 SC with Bottlenecks: Training Methodologies

Given that recent **SC** studies with bottleneck injection strategies result in more or less accuracy loss compared to the original models (i.e., without injected bottlenecks), various training methodologies are used and/or proposed in such studies. Some of the training methods are designed specifically for architectures with injected bottlenecks. We now summarize the differences between the various training methodologies used in recent **SC** studies.

We recall that \mathbf{x} and \mathbf{y} are an input (e.g., an RGB image) and the corresponding label (e.g., one-hot vector), respectively. Given an input \mathbf{x} , a DNN model \mathcal{M} returns its output $\hat{\mathbf{y}} = \mathcal{M}(\mathbf{x})$ such as class probabilities in classification task. Each of the L layers of model \mathcal{M} can be either low-level (e.g., convolution [69], batch normalization [52], ReLU [100]) or high-level layers (e.g., residual block in ResNet [43] and dense block in DenseNet [51]), which are composed by multiple low-level layers. $\mathcal{M}(\mathbf{x})$ is a sequence of the L layer functions f_j s, and the j th layer transforms \mathbf{z}_{j-1} , the output from the previous $(j - 1)$ th layer:

$$\mathbf{z}_j = \begin{cases} \mathbf{x} & j = 0 \\ f_j(\mathbf{z}_{j-1}, \theta_j) & 1 \leq j < L, \\ f_L(\mathbf{z}_{L-1}, \theta_L) = \mathcal{M}(\mathbf{x}) = \hat{\mathbf{y}} & j = L \end{cases} \quad (1)$$

where θ_j denotes the j th layer's hyperparameters and parameters to be optimized during training.

Cross-entropy-based Training

To optimize parameters in a DNN model, we first need to define a loss function and update the parameters by minimizing the loss value with an optimizer such as stochastic gradient descent and Adam [62] during training. In image classification, a standard method is to train a DNN model \mathcal{M} in an end-to-end manner using the cross-entropy like many of the studies [26, 49, 91] in Table 3. For simplicity, here we focus on the categorical cross-entropy and assume $c \equiv \mathbf{y}$ is the correct class index given a model input \mathbf{x} . Given a pair of \mathbf{x} and c , we obtain the model output $\hat{\mathbf{y}} = \mathcal{M}(\mathbf{x})$, and then the (categorical) cross-entropy loss is defined as

$$\mathcal{L}_{CE}(\hat{\mathbf{y}}, c) = -\log \left(\frac{\exp(\hat{\mathbf{y}}_c)}{\sum_{j \in C} \exp(\hat{\mathbf{y}}_j)} \right), \quad (2)$$

where $\hat{\mathbf{y}}_j$ is the class probability for the class index j , and C is a set of considered classes ($c \in C$).

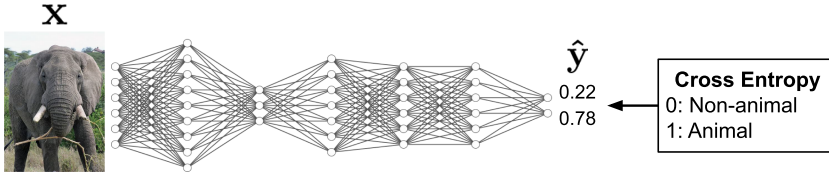


Fig. 3. Cross-entropy-based training for bottleneck-injected DNN.

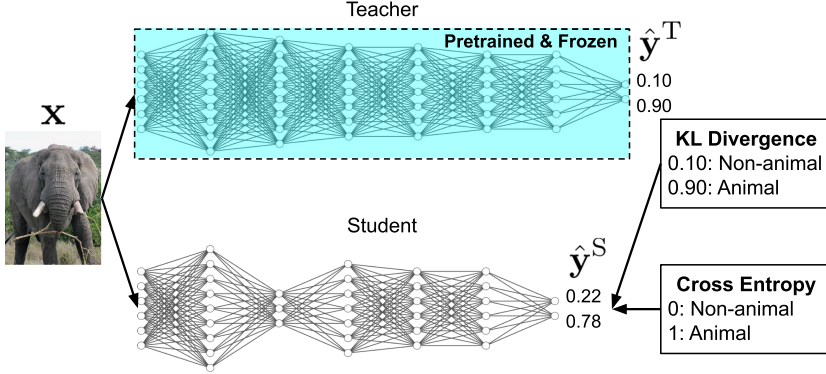


Fig. 4. Knowledge distillation for bottleneck-injected DNN (student), using a pretrained model as teacher.

As shown in Equation (2), the loss function used in cross-entropy-based training methods are used as a function of the final output \hat{y} , and thus are not designed for **SC** frameworks. While Eshratifar et al. [26], Hu and Krishnamachari [49], Lee et al. [70], and Shao and Zhang [126] use cross-entropy to train bottleneck-injected DNN models in end-to-end manners (Figure 3), Matsubara et al. [91] empirically show that these methods cause a larger accuracy loss in complex tasks such as the ImageNet dataset [120] compared to other more advanced techniques, including knowledge distillation.

Knowledge Distillation

Complex DNN models are usually trained to learn parameters for discriminating between a large number of classes (e.g., 1,000 in ImageNet dataset) and are often overparameterized. **Knowledge distillation (KD)** [3, 46, 78] is a training scheme to address this problem, and trains a DNN model (called “student”) using additional signals from a pretrained DNN model (called “teacher” and often larger than the student). In standard cross-entropy-based training—that is, using “hard targets” (e.g., one-hot vectors)—we face a side effect that the trained models assign probabilities to all of the incorrect classes. From the relative probabilities of incorrect classes, we can see how large models tend to generalize.

As illustrated in Figure 4, by distilling the knowledge from a pretrained complex model (teacher), a student model can be more generalized and avoid overfitting to the training dataset, using the outputs of the teacher model as “soft targets” in addition to the hard targets [46].

$$\mathcal{L}_{\text{KD}}(\hat{y}^S, \hat{y}^T, y) = \alpha \mathcal{L}_{\text{task}}(\hat{y}^S, y) + (1 - \alpha) \tau^2 \text{KL}(q(\hat{y}^S), p(\hat{y}^T)), \quad (3)$$

where α is a balancing factor (hyperparameter) between the *hard target* (left term) and *soft target* (right term) losses, and τ is another hyperparameter called *temperature* to soften the outputs of

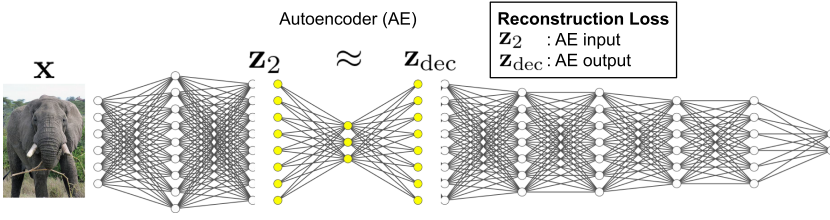


Fig. 5. Reconstruction-based training to compress intermediate output (here z_2) in DNN by Autoencoder (AE) (yellow).

teacher and student models in Equation (4). $\mathcal{L}_{\text{task}}$ is a task-specific loss function, and it is a cross-entropy loss in image classification tasks, i.e., $\mathcal{L}_{\text{task}} = \mathcal{L}_{\text{CE}}$. KL is the Kullback-Leibler divergence function, where $q(\hat{y}^S)$ and $p(\hat{y}^T)$ are probability distributions of student and teacher models for an input x , that is, $q(\hat{y}^S) = [q_1(\hat{y}^S), \dots, q_{|C|}(\hat{y}^S)]$ and $p(\hat{y}^T) = [p_1(\hat{y}^S), \dots, p_{|C|}(\hat{y}^T)]$:

$$q_k(\hat{y}^S) = \frac{\exp\left(\frac{\hat{y}_k^S}{\tau}\right)}{\sum_{j \in C} \exp\left(\frac{\hat{y}_j^S}{\tau}\right)}, \quad p_k(\hat{y}^T) = \frac{\exp\left(\frac{\hat{y}_k^T}{\tau}\right)}{\sum_{j \in C} \exp\left(\frac{\hat{y}_j^T}{\tau}\right)}. \quad (4)$$

Using the ImageNet dataset, it is empirically shown in Matsubara et al. [91] that all the considered bottleneck-injected student models trained with their teacher models (original models without injected bottlenecks) consistently outperform those trained without the teacher models. This result matches a widely known trend in knowledge distillation reported in Ba and Caruana [3]. However, similar to cross-entropy, the knowledge distillation is still not aware of bottlenecks we introduce to DNN models and may result in significant accuracy loss as suggested by Matsubara et al. [91].

Reconstruction-based Training

As illustrated in Figure 5, Choi et al. [14], Jankowski et al. [57], Sbair et al. [124], and Yao et al. [164] inject AE models into existing DNN models and train the injected components by minimizing the reconstruction error. First, manually an intermediate layer in a DNN model (say its j th layer) is chosen, and the output of the j th layer z_j is fed to the encoder f_{enc} whose role is to compress z_j . The encoder's output z_{enc} is a compressed representation, i.e., bottleneck, to be transferred to the edge server, and the following decoder f_{dec} decompresses the compressed representation and returns z_{dec} . As the decoder is designed to reconstruct z_j , its output z_{dec} should share the same dimensionality with z_j . Then, the injected AEs are trained by minimizing the following reconstruction loss:

$$\begin{aligned} \mathcal{L}_{\text{Recon.}}(z_j) &= \|z_j - f_{\text{dec}}(f_{\text{enc}}(z_j; \theta_{\text{enc}}); \theta_{\text{dec}}) + \epsilon\|_n^m, \\ &= \|z_j - z_{\text{dec}} + \epsilon\|_n^m, \end{aligned} \quad (5)$$

where $\|z\|_n^m$ denotes the m^{th} power of the n -norm of z , and ϵ is an optional regularization constant. For example, Choi et al. [14] set $m = 1$, $n = 2$, and $\epsilon = 10^{-6}$, and Jankowski et al. [57] use $m = n = 1$ and $\epsilon = 0$. Inspired by the idea of knowledge distillation [46], Yao et al. [164] also consider additional squared errors between intermediate feature maps from models with and without bottlenecks as additional loss terms like generalized head network distillation [94], described later. While Yao et al. [164] show a high compression rate with small accuracy loss by injecting encoder-decoder architectures into existing DNN models, such strategies [14, 57, 124, 164] increase

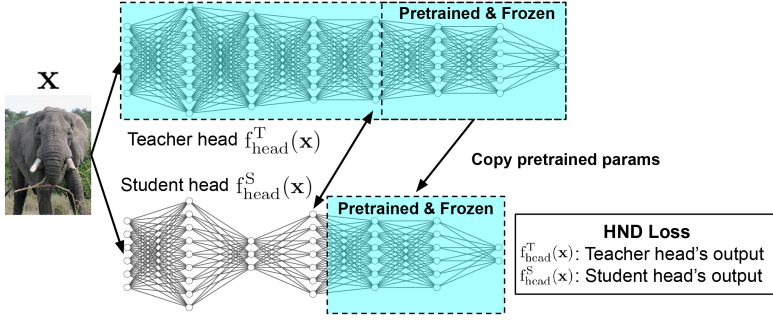


Fig. 6. Head network distillation for bottleneck-injected DNN (student), using a pretrained model as teacher. The student model's tail portion is copied from that of its teacher model with respect to the architecture and pretrained parameters.

computational complexity as a result. Suppose the encoder and decoder consist of L_{enc} and L_{dec} layers, respectively; then the total number of layers in the altered DNN model is $L + L_{\text{enc}} + L_{\text{dec}}$.

Head Network Distillation

The training methods described above are focused on either end-to-end or encoder-decoder training. The first approach often requires hard targets such as one-hot vectors and more training cost, while the latter can focus on the injected components (encoder and decoder) during training, but the additional components (layers) will increase the complexity of the DNN model. To reduce both training cost and model complexity while preserving accuracy, it is proposed in Matsubara et al. [90] to use **head network distillation (HND)** to distill the head portion of the DNN—which contains a bottleneck—leveraging pretrained DNN models. Figure 6 illustrates this approach.

The original pretrained DNN (consisting of L layers) is used as a starting point, whose architecture (in the head part) is simplified. As only the teacher's head portion is altered, the tail portion of the student model is identical to that of the teacher model with respect to architecture and the same pretrained parameters can be maintained. Thus, head network distillation requires only the first layers of the teacher and student models in the training session as the student head model f_{head}^S will be trained to mimic the behavior of teacher's head model f_{head}^T given an input x :

$$\mathcal{L}_{\text{HND}}(x) = \|f_{\text{head}}^S(x; \theta_{\text{head}}^S) - f_{\text{head}}^T(x; \theta_{\text{head}}^T)\|^2, \quad (6)$$

where f_{head}^S and f_{head}^T are sequences of the first L_{head}^S and L_{head}^T layers in student and teacher models ($L_{\text{head}}^S \ll L^S$, and $L_{\text{head}}^T \ll L$), respectively.

Experimental results with the ImageNet (ILSVRC 2012) dataset show that given a bottleneck-introduced model, the head network distillation method consistently outperforms cross-entropy-based training [26, 49, 126] and knowledge distillation methods in terms of not only training cost but also accuracy of the trained model. This method is extended in Matsubara and Levorato [94], where the generalized **head network distillation technique (GHND)** is proposed for complex object detection tasks and models. We note that these tasks require finer feature maps mimicking those at intermediate layers in the original pretrained object detectors. The loss function in this approach is

$$\mathcal{L}_{\text{GHND}}(x) = \sum_{j \in \mathcal{J}} \lambda_j \cdot \mathcal{L}_j(x, f_{1-L_j}^S, f_{1-L_j}^T), \quad (7)$$

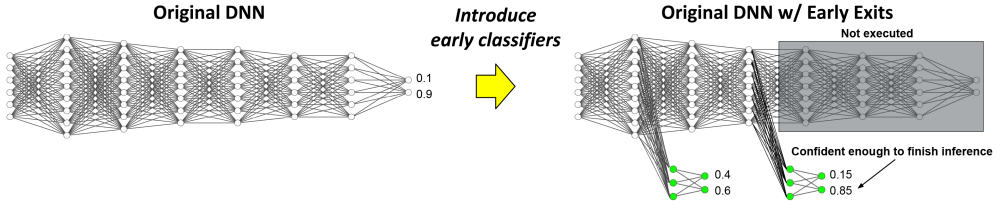


Fig. 7. Illustration of two early exits (green) introduced to DNN.

where j is the loss index, λ_j is a scale factor (hyperparameter) associated with loss \mathcal{L}_j , and $f_{1-L_j}^S$ and $f_{1-L_j}^T$ indicate the corresponding sequences of the first L_j^S and L_j^T layers in the student and teacher models (functions of input data \mathbf{x}), respectively. The total loss, then, is a linear combination of $|\mathcal{J}|$ weighted losses. Following Equation (7), the previously proposed head network distillation technique [90] can be seen as a special case of **generalized head network distillation (GHND)**. **GHND** significantly improved the object detection performance in bottleneck-injected R-CNN models on the COCO 2017 dataset while achieving a high compression rate.

5 EARLY EXITING: A SURVEY

This section presents a survey of the state of the art in **EE** strategies. We first provide a compendium of work focused on **CV** and **NLP** applications in Sections 5.2 and 5.3, respectively. Section 5.4 summarizes training methodologies used in the **EE** studies.

5.1 Rationale behind EE

The core idea of **EE**, first proposed by Teerapittayanon et al. [140], is to circumvent the need to make DNN models smaller by introducing early exits in the DNN, where execution is terminated at the first exit achieving the desired confidence on the input sample. For instance, some samples in test datasets (and in real-world problems) will be easy for a DNN model, but others may not be, depending on the ML models we use. Thus, **EE** ends the inference process with fewer transforms (layers) for such easy samples so that the overall inference time and computation cost are reduced.

Figure 7 illustrates an example of early classifiers (subbranches) introduced in a DNN model. In this example, the second early classifier has sufficient confidence in its output (class probability is 0.85 out of 1.0) to terminate the inference for the input sample so that the following layers are not executed. Note that all the exits are executed until the desired confidence is reached; that is, the computational complexity up to that point increases. Thus, the classifiers added to the DNN model need to be simple; that is, they need to have fewer layers than the layers after the branches. Otherwise, the overall inference cost will increase on average rather than decrease. Teerapittayanon et al. [141] also apply this idea to mobile-edge-cloud computing systems; the smallest neural model is allocated to the mobile device, and if that model's confidence for the input is not large enough, the intermediate output is forwarded to the edge server, where inference will continue using a mid-sized neural model with another exit. If the output still does not reach the target confidence, the intermediate layer's output is forwarded to the cloud, which executes the largest neural model. **EE** strategies have been widely investigated in the literature, as summarized in Table 4.

As shown in Tables 1 and 3, most of the studies on **SC** were focused on computer vision. For **EE**, we can confirm a good balance between the studies with computer vision and **NLP** applications as summarized in Table 4, with structural/conceptual differences between the two domains. Moreover,

Table 4. Studies on Early Exiting Strategies

Work	Task(s)	Dataset(s)	Base Model(s)	Metrics	Code
Teerapittayanon et al. [140]	Image classification	MNIST [69] CIFAR-10 [63]	LeNet-5 [69] AlexNet [64] ResNet [43]	A, L	Link
Teerapittayanon et al. [141]	Image classification*	Multi-camera multi-object detection [119]	Distributed DNNs	A, D	Link
Lo et al. [87]	Image classification	CIFAR-10/100 [63]	NiN [80] ResNet [43] WRN [167]	A, C	
Neshatpour et al. [102]	Image classification	ImageNet [120]	AlexNet [64]	A, C, L	
Zeng et al. [168]	Image classification	CIFAR-10 [63]	AlexNet [64]	A, D, L	
Wang et al. [148]	Image classification	CIFAR-10/100 [63]	ResNet [43]	A, C	
Li et al. [77]	Image classification	CIFAR-10/100 [63] ImageNet (2012) [120]	MSDNet [50]	A, C	Link
Phuong and Lampert [108]	Image classification	CIFAR-100 [63] ImageNet (2012) [120]	MSDNet [50]	A	Link
Elbayad et al. [24]	Machine translation	IWSLT'14 De-En WMT'14 En-Fr	Transformer [145]	A, C	
Wang et al. [150]	Image classification	CIFAR-100 [63] ImageNet (2012) [120]	ResNet [43] DenseNet [51]	A, C, E	
Yang et al. [158]	Image classification	CIFAR-10/100 [63] ImageNet [120]	RANet	A, C	Link
Soldaini and Moschitti [132]	Text ranking	WikiQA [163], TREC QA [149], ASNQ [32], GPD	RoBERTa [85]	A, C	Link
Liu et al. [84]	Text classification	ChnSentiCorp, Book review [112], Shopping review, Weibo, THUCNews, Ag.News, Amz.F, DBpedia, Yahoo, Yelp.F, Yelp.P [171]	BERT [21]	A, C, T	Link
Xin et al. [156]	GLUE [146]	SST-2 [131], MRPC [22], QQP [54], MNLI [152], QNLI [114], RTE [7, 18, 33, 37]	BERT [21] RoBERTa [85]	A, C	Link
Xing et al. [157]	Quality enhancement	RAISE [19]	Dynamic DNN	A, C	Link
Laskaridis et al. [67]	Image classification	CIFAR-100 [63] ImageNet (2012) [120]	ResNet-56 [43] ResNet-50 [43] Inception-v3 [136]	A, E, L	
Xin et al. [155]	Text ranking	MS MARCO [104] ASNQ [32]	BERT [21]	A, L	Link
Zhou et al. [172]	GLUE [146]	CoLA [151], SST-2 [131], MRPC [22], STS-B [9], QQP [54], MNLI [152], QNLI [114], WNLI [71], RTE [7, 18, 33, 37]	BERT [21] ALBERT [66]	A, C, L, T	Link
Matsubara and Levorato [94]	Keypoint detection	COCO 2017 [83]	Keypoint R-CNN [42]	A, D, L	Link
Garg and Moschitti [31]	Text ranking Question answering	WikiQA [163], ASNQ [32] SQuAD 1.1 [114]	BERT [21] RoBERTa [85] ELECTRA [15]	A, L	Link
Wolczyk et al. [153]	Image classification	CIFAR-10/100 [63], Tiny ImageNet	ResNet-56 [43] MobileNet [48] WideResNet [167] VGG-16BN [128]	A, L	Link
Chiang et al. [12]	Image classification	CIFAR-100 [63]	VGG-11 [128] VGG-13 [128] VGG-16 [128] VGG-19 [128]	A, L	
Pomponi et al. [109]	Image classification	SVHN [103], CIFAR-10/100 [63]	AlexNet [64] VGG-11 [128] ResNet-20 [43]	A	Link

A: Model accuracy, C: Model complexity, D: Transferred data size, E: Energy consumption, L: Latency, T: Training cost.

*The authors extract annotated objects from the original dataset for multi-camera object detection and use the extracted images for an image classification task.

CNN (e.g., AlexNet [64] and ResNet [43]) and Transformer-based models (e.g., BERT [21]) are mostly discussed in the EE studies for computer vision and NLP, respectively. For these reasons, we categorize the EE papers by task domain in Sections 5.2 and 5.3.

5.2 EE for CV Applications

Similar to the **SC** studies we discussed in Section 4, the research community mainly focused on **EE** approaches applied to **CV** tasks.

Design approaches

Wang et al. [150] propose a unified Dual Dynamic Inference that introduces the following features to a DNN model: **Input-Adaptive Dynamic Inference (IADI)** and **Resource-Adaptive Dynamic Inference (RADI)**. The IADI dynamically determines which sub-networks to be executed for cost-efficient inference, and the RADI leverages the concept of **EE** to offer “anytime classification.” Using the concept of **EE**, Lo et al. [87] propose two different methods: (1) authentic operation and (2) dynamic network sizing. The first approach is used to determine whether the model input is transferred to the edge server, and the latter dynamically adjusts the number of layers to be used as an auxiliary neural model deployed on the mobile device for efficient usage of communication channels in **EC** systems. Neshatpour et al. [102] decompose a DNN’s inference pipeline into multiple stages and introduce **EE** (termination) points for energy-efficient inference.

Training Approaches

Wang et al. [148] focus on training methods for DNNs with an early exit and observe that prior **EE** approaches suffered from the burden of manually tuning balancing weights of early-exit losses to find a good tradeoff between computational complexity and overall accuracy. To address this problem, the authors propose a strategy to dynamically adjust the loss weights for the ResNet models they consider. Li et al. [77] and Phuong and Lampert [108] introduce multiple early exits to DNN models and apply knowledge distillation to each of the early exits as students, using their final classifiers as teacher models. Similar to other studies, the DNNs with early exits are designed to finish inference for “easy” samples by early sub-classifiers based on confidence thresholds defined beforehand.

Inference Approaches

Yang et al. [158] leverage **EE** strategies for multi-scale inputs and propose an approach to classify “easy” samples with smaller neural models. Different from prior studies, their proposed approach scales up the input image (use higher-resolution image as input), depending on the classification difficulty of the sample. Laskaridis et al. [67] design a distributed inference system that employs synergistic device-cloud computation for collaborative inference, including an **EE** strategy (referred to as progressive inference in their work). Xing et al. [157] apply **EE** strategies to quality enhancement tasks and propose a resource-efficient blind quality enhancement approach for compressed images. By identifying “easy” samples in the tasks, they dynamically process input samples with/without early exits. Zeng et al. [168] combine **EE** and **SC** approaches and propose a framework named Boomerang, which is designed to automate end-to-end DNN inference planning for IoT scenarios; they introduce multiple early exits in AlexNet [64]. Their proposed framework profiles the model to decide its partition (splitting) point.

In addition to introducing and training bottleneck points for object detectors, Matsubara and Levorato [94] introduce a *neural filter* in an early stage of the head-distilled Keypoint R-CNN model. Similarly to **EE** frameworks, the filter identifies pictures without objects of interest and triggers termination of the execution before the output of the bottleneck is forwarded. Wołczyk et al. [153] propose Zero Time Waste, a method in which each early exit reuses predictions returned by its predecessors. The method adds direct connections between early exits and combines outputs of the previous early exits like an ensemble model. Through experiments with multiple image

classification datasets and model architectures, they demonstrate that their proposed method improves a tradeoff between accuracy and inference time compared to other early-exit methods. Extending the idea of BranchyNet [140], Chiang et al. [12] formulate the early-exit (branch) placement problem. They propose a dynamic programming algorithm to address the problem and discuss the tradeoff between model accuracy and inference time. Pomponi et al. [109] introduce multiple early exits to a classifier and train the entire multi-exit model jointly. Using multiple base models, they discuss various early-exit stopping criteria. Many studies on EE for CV tasks publish their source code to ensure replicability of their work.

5.3 EE for NLP Applications

Interestingly, EE approaches have been widely studied not only in CV tasks—the main application of SC—but also in NLP tasks. Recent studies introduce subbranches (early exits) to transformer-based models such as BERT [21]. While these transformer-based models achieve state-of-the-art performance in NLP tasks, they have an extremely large number of parameters; e.g., BERT [21] has up to 355 million parameters, whereas the largest image classification model used in SC studies (Tables 1 and 3), ResNet-152, has 60 million parameters.

In Elbayad et al. [24] an EE technique for NLP tasks is developed for transformer sequence-to-sequence models [145] in machine translation tasks. The decoder networks in the considered transformer models can be trained by either aligned training or mixed training methods. The former method optimizes all classifiers in the decoder network simultaneously. However, when a different classifier (exit) is chosen for each token (e.g., word) at test time, some of the hidden states from previous time steps may be missed and then the input states to the following decoder network will be misaligned (mismatched). The latter method addresses this issue. In mixed sample training, several paths of random exits are sampled at which the model is assumed to have exited for reducing the mismatch by feeding hidden states from different decoder depths of previous time steps.

For different tasks, Soldaini and Moschitti [132], Xin et al. [156], and Liu et al. [84] propose EE frameworks based on BERT [21] and RoBERTa [85] that share almost the same network architecture. Focused on text ranking, specifically answer sentence selection tasks with question-answering datasets, Soldaini and Moschitti [132] add classification layers to intermediate stages of RoBERTa to build sequential (neural) rerankers [95] inside as early exits and propose the Cascade Transformer models. Focusing on powerful transformer models for industrial scenarios, Liu et al. [84] discuss the effectiveness on 12 (6 English and 6 Chinese) NLP datasets of BERT models when early classifiers are introduced. Similar to the studies by Li et al. [77] and Phuong and Lampert [108], Liu et al. [84] leverage knowledge distillation [46] to train early classifiers, treating the final classifier of the BERT model and their introduced early classifiers as teacher and student classifiers, respectively. Xin et al. [156] target **general language understanding evaluation (GLUE)** tasks [146] and introduce early exits after each of 12 transformer blocks in BERT and RoBERTa models.

While the Cascade Transformer [132] disregards a fixed portion of candidates (samples) given a query in answer sentence selection tasks, Xin et al. [155] use a score-based EE strategy for a BERT architecture for text ranking tasks. Zhou et al. [172] introduce early classifiers to BERT and ALBERT [66] models and discuss adversarial robustness using the ALBERT models with and without the early exits. Using an adversarial attack method [59], the authors feed perturbed input data (called adversarial examples [65]) to their trained models and show how robust their models are against the adversarial attack, compared to those without early classifiers. Garg and Moschitti [31] propose an approach to filter out questions in answer sentence selection and question-answering tasks. Leveraging the concept of knowledge distillation, they train a question filter model (student),

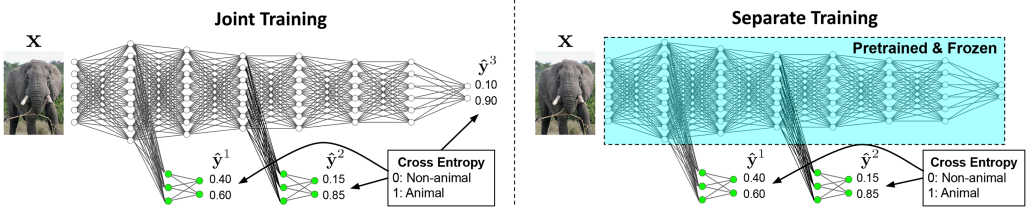


Fig. 8. Examples of joint and separate training methods for DNN with early exits.

whose input is a query, by mimicking the top-1 candidate score of the answer model (teacher), whose input is a pair of query and the list of the candidate answers. When the trained question filter model finds a query answerable for the answer model, the subsequent inference pipeline will be executed. Otherwise, the question filter model terminates the inference process for the query (i.e., early exit) to save the overall inference cost.

Most of the studies on **EE** for **NLP** tasks in Table 4 are published with source code to ensure replicable results. Notably, this application domain enjoys a well-generalized open source framework—Huggingface’s Transformers [154]—which provides state-of-the-art (pretrained) Transformer models, including the BERT, RoBERTa, and ALBERT models used in the above studies.

5.4 Training Methodologies for EE Strategies

To introduce **EE** strategies, the early classifiers need to be trained in addition to the base models. We can categorize the training methodologies used in **EE** studies into two main classes: *joint training* and *separate training*, illustrated in Figure 8 and described in the next sections.

Joint Training

Most of the training methods used in existing works belong to this category. Joint training trains all the (early) classifiers in a model simultaneously (left part of Figure 8). Specifically, these studies [24, 67, 87, 109, 132, 140, 141, 148, 150, 155, 157, 158, 168, 172] define a loss function for each of the classifiers and minimize the weighted sum of cross-entropy losses per sample as follows:

$$\mathcal{L}_{\text{Joint}}([\hat{y}^1, \dots, \hat{y}^N], c) = \sum_{j=1}^N \lambda_j \mathcal{L}_{\text{CE}}(\hat{y}^j, c), \quad (8)$$

where $[\hat{y}^1, \dots, \hat{y}^N]$ indicates outputs from N (early) classifiers, and the correct label c is shared across all the classifiers in a model. Note that the base model (final classifier) is also counted as one of the N classifiers, and $N - 1$ early classifiers are introduced to the base model.

Instead, Li et al. [77] and Phuon and Lampert [108] use a knowledge distillation-based loss such as Equation (3) by treating the final classifier (last exit) as a teacher model and all the early classifiers as student models. This approach is based on the assumption that the last classifier will achieve the highest accuracy among all the (early) classifiers in the model, and early classifiers (students) could learn from the last classifier as a teacher model.

Separate Training

A few studies [31, 84, 94, 156] suggest training the early classifiers separately. This approach can be interpreted as a two-stage training paradigm that trains a model in the first stage and then trains the early classifiers introduced to the pretrained model whose parameters are fixed in the second stage (see Figure 8 (right)). For instance, Xin et al. [156] fine-tune a BERT model in the first stage following Devlin et al. [21]. Then, the early classifiers are introduced in the model and

trained while all the parameters of the BERT model learned in the first stage are kept frozen. Liu et al. [84] adopt a similar approach, but in the second training stage, knowledge distillation is used to train the early classifiers. Different from SC studies using knowledge distillation, the teacher model is fixed, and only the additional parameters corresponding to the early classifiers are trained. Wolczyk et al. [153] introduce early exits to a pretrained model. Using the cross-entropy loss, they train the introduced early exits.

6 SPLIT COMPUTING AND EARLY EXITING: RESEARCH CHALLENGES

In this section, we describe some of the research challenges in the SC and EE domains.

Evaluation of SC and EE in More Practical Settings

Due to the cross-disciplinary nature of this research area, it is essential to design practical and convincing evaluation settings to demonstrate the effectiveness of proposed approaches. As shown in Tables 3 and 4, the techniques proposed in many of the recent related studies are validated only on small-scale datasets such as MNIST and CIFAR datasets, which leads to some concerns on the input data size in relation to compression. Indeed, Table 2 suggests that the input data size in many of such datasets is relatively small (e.g., smaller than 2 kilobytes per image with a resolution of 32×32 pixels). The low resolution of the input size may enable conventional EC, where the mobile device fully offloads the computing task by transferring the input data to an edge server. In fact, the transmission of such a small amount of data would require a short time even in settings with limited communication capacity. As a consequence, executing even small head models on a resource-limited mobile device could lead to an overall delay increase.

Based on the above discussion, it becomes apparent that the models and datasets, in addition to the wireless and computing environments, are of paramount importance when assessing the performance of SC and EE schemes. Of particular relevance is the evaluation of accuracy, which is not provided in some of the early studies (e.g., [43, 122, 128]), and the consideration of state-of-the-art models and datasets, which are largely used in the machine learning community. For instance, the use of small models, such as MobileNetV2, ResNet-50, and VGG-16, which are likely overparameterized for simple classification tasks, could lead to wrong conclusions when injecting bottlenecks. Conversely, it was shown in [90] how challenging it is to inject bottlenecks when considering complex vision tasks such as classification on the ImageNet dataset [120].

Optimization of Bottleneck Design and Placement in SC

The study of the architecture and placement of the bottleneck in a DNN model is also of considerable importance. As suggested in [96], important metrics include (1) bottleneck data size (or compression rate), (2) complexity of the head model executed on the mobile device, and (3) resulting model accuracy. As a principle, the smaller the bottleneck representation is, the lower the communication cost between the mobile device and edge server will be. In general, the objective of SC is to generate a bottleneck whose data size is smaller than that of input data such as JPEG file size of input data, which is in turn much smaller than the data size of the input tensor (32-bit floating point), as the communication delay is a key component to reduce overall inference time [90, 91, 94, 158]. Secondly, since mobile devices often have limited computing resources and may have other constraints such as energy consumption due to their battery capacities, SC should aim at minimizing their computational load by making head models as lightweight as possible. For instance, designing a small bottleneck at a very early stage of the DNN model enables a reduction in the computational complexity of the head model [93, 94].

On top of these two criteria, the resulting model accuracy by the bottleneck injection should not be compromised as the introduced bottleneck removes more or less information at the placement

compared to the original model. A reasonable lower bound of the model accuracy in **SC** would be that of widely recognized lightweight models, e.g., MobileNetV2 [122] for ImageNet dataset, considering a local computing system where such lightweight models can be efficiently executed. In general, it is challenging to optimize bottleneck design and placement with respect to all three different metrics, and existing studies empirically design the bottlenecks and determine the placements. Thus, theoretical discussion on bottleneck design and placement should be an interesting research topic for future work.

Dynamic Control of Exits in EE

In most of the recent studies, early exits are used when one of the introduced early classifiers (exits) is confident enough in its prediction. However, users are required to determine a threshold for each of the classifiers beforehand at least for one early classifier in the original model where we introduce the early classifier. For example, if the first classifier's prediction score is greater than 0.9 in the range of 0.0 and 1.0, then the inference for the input is terminated.

To achieve more efficient inference without significantly sacrificing the accuracy of the original model, the system needs to find a balance between (early) classifiers. As recent studies introduce multiple early exits to a model at different stages, such optimizations are challenging. In addition to manually defining such a threshold for each of the classifiers based on empirical results, a possibly interesting direction is the optimization of the decision-making process, that is, at which (early) classifier we should terminate the inference for a given input, without a set of thresholds defined beforehand based on system characteristics.

Expanding the Application Domain of SC and EE

The application domains of **SC** and (in minor part) **EE** remain primarily focused on image classification. This focus may be explained by the size of the input, which makes compression a relevant problem in many settings and the complexity of the models and tasks. However, there are many other unexplored domains from which **SC** would benefit. Real-time health conditions monitoring via wearable sensors is a notable example of an application where a significant amount of data is transferred from sensors to edge servers such as cellular phones and home hubs. For instance, the detection and monitoring of heart anomalies (e.g., arrhythmia) from ECG [30] require the processing of high-rate samples (e.g., 100 to 1,000 per heart cycle) using high-complexity DNN models [41]. Health monitoring applications pose different challenges compared to **CV**-based applications. Indeed, in the former, both the computing capacity and the bandwidth available to the system are often smaller compared to the latter scenario, and conceptual advancements are required.

Toward an Information-theoretic Perspective

The key intuition behind the success of **SC** and **EE** is similar to what has led to the success of techniques such as model pruning [38, 44, 74, 160] and knowledge distillation [46, 61, 98]: most state-of-the-art DNNs are significantly overparameterized [165, 166]. A possible approach to justify **SC** and **EE** can be found in the study of *information bottlenecks (IBs)*, which were introduced in [142] as a compression technique in which a random variable X is compressed while preserving relevant information about another random variable Y . The IB method has been applied in [143] to quantify mutual information between the network layers and derive an information theory limit on DNN efficiency. This has led to attempts at explaining the behavior of deep neural networks with the information bottleneck formalism [123].

Despite these early attempts, a strong connection between this relatively new perspective and the techniques described in this article is still elusive. Some of the approaches and architectures discussed in this article are meaningful attempts to efficiently extract a compressed representation

of the input and provide sufficient information toward a certain task early in the network layers. The emerging IB formalism is a promising approach to enable the first moves in the information-theoretic analysis of neural-network-based transformations. We believe that this interpretation could serve as a foundation for an in-depth study of structural properties for both **SC** and **EE**.

7 CONCLUSIONS

Mobile devices such as smartphones and drones have now become an integral part of our daily lives. These devices increasingly utilize deep neural networks (DNNs) to execute complex inference tasks such as image classification and speech recognition, among others. For this reason, in this article, we provided a comprehensive survey of the state of the art in **split computing (SC)** and **early exiting (EE)** by presenting a thorough comparison of the most relevant approaches. We also provided a set of compelling research challenges that need to be addressed to improve existing work in the field. We hope this survey will elicit further research in these emerging fields.

REFERENCES

- [1] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melia-Segui, and Thomas Watteyne. 2017. Understanding the limits of LoRaWAN. *IEEE Communications Magazine* 55, 9 (2017), 34–40.
- [2] Juliano S. Assine, J. C. S. Santos Filho, and Eduardo Valle. 2021. Single-training collaborative object detectors adaptive to bandwidth and computation. *arXiv preprint arXiv:2105.00591* (2021).
- [3] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep? In *Neural Information Processing Systems 2014*. 2654–2662.
- [4] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. 2017. End-to-end optimized image compression. In *International Conference on Learning Representations*.
- [5] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. 2018. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*.
- [6] Marco V. Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. 2013. To offload or not to offload? The bandwidth and energy costs of mobile cloud computing. In *Proceedings of IEEE International Conference on Computer Communications 2013*. 1285–1293.
- [7] Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth PASCAL recognizing textual entailment challenge. In *Proceedings of Text Analysis Conference (TAC'09)*.
- [8] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 535–541.
- [9] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and cross-lingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval'17)*. 1–14.
- [10] Jiasi Chen and Xukan Ran. 2019. Deep learning with edge computing: A review. *Proceedings of the IEEE* 107, 8 (2019), 1655–1674.
- [11] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. 2017. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587* (2017).
- [12] Chang-Han Chiang, Pangfeng Liu, Da-Wei Wang, Ding-Yong Hong, and Jan-Jan Wu. 2021. Optimal branch location for cost-effective inference on branchynet. In *2021 IEEE International Conference on Big Data (Big Data'21)*. IEEE, 5071–5080.
- [13] Hyomin Choi and Ivan V. Bajić. 2018. Deep feature compression for collaborative object detection. In *2018 25th IEEE International Conference on Image Processing (ICIP'18)*. IEEE, 3743–3747.
- [14] Hyomin Choi, Robert A. Cohen, and Ivan V. Bajić. 2020. Back-and-forth prediction for deep tensor compression. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'20)*. IEEE, 4467–4471.
- [15] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2019. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.
- [16] Robert A. Cohen, Hyomin Choi, and Ivan V. Bajić. 2020. Lightweight compression of neural network feature tensors for collaborative intelligence. In *2020 IEEE International Conference on Multimedia and Expo (ICME'20)*. IEEE, 1–6.
- [17] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12 (2011), 2493–2537.

- [18] Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Proceedings of the 1st International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*. 177–190.
- [19] Duc-Tien Dang-Nguyen, Cecilia Pasquini, Valentina Conotter, and Giulia Boato. 2015. RAISE: A raw images dataset for digital image forensics. In *Proceedings of the 6th ACM Multimedia Systems Conference*. 219–224.
- [20] Li Deng, Geoffrey Hinton, and Brian Kingsbury. 2013. New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 8599–8603.
- [21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.
- [22] William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the 3rd International Workshop on Paraphrasing (IWP'05)*.
- [23] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. 2018. DPP-Net: Device-aware progressive search for Pareto-optimal neural architectures. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 517–531.
- [24] Maha Elbayad, Jiatao Gu, E. Grave, and M. Auli. 2020. Depth-adaptive transformer. In *International Conference on Learning Representations*.
- [25] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2019. JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing* (2019).
- [26] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud Pedram. 2019. BottleNet: A deep learning architecture for intelligent mobile cloud computing services. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED'19)*. 1–6. <https://ieeexplore.ieee.org/document/8824955>.
- [27] Mark Everingham, Luc Van Gool, C. K. I. Williams, John Winn, and Andrew Zisserman. 2012. The PASCAL Visual Object Classes Challenge 2012 (VOC2012).
- [28] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. [n.d.]. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. <http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html>.
- [29] Li Fei-Fei, Rob Fergus, and Pietro Perona. 2006. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 4 (2006), 594–611.
- [30] Matteo Gadaleta, Michele Rossi, Steven R. Steinhilb, and Giorgio Quer. 2018. Deep learning to detect atrial fibrillation from short noisy ECG segments measured with wireless sensors. *Circulation* 138, Suppl_1 (2018), A16177–A16177.
- [31] 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*. 7329–7346.
- [32] Siddhant Garg, Thuy 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*, Vol. 34. 7780–7788.
- [33] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B. Dolan. 2007. The third Pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*. 1–9.
- [34] Odd Erik Gundersen and Sigbjørn Kjensmo. 2018. State of the art: Reproducibility in artificial intelligence. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
- [35] Tian Guo. 2018. Cloud-based or on-device: An empirical study of mobile deep inference. In *2018 IEEE International Conference on Cloud Engineering (IC2E'18)*. IEEE, 184–190.
- [36] Lav Gupta, Raj Jain, and Gabor Vaszkun. 2015. Survey of important issues in UAV communication networks. *IEEE Communications Surveys & Tutorials* 18, 2 (2015), 1123–1152.
- [37] R. Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szepktor. 2006. The second Pascal recognising textual entailment challenge. In *Proceedings of the 2nd PASCAL Challenges Workshop on Recognising Textual Entailment*.
- [38] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *4th International Conference on Learning Representations*.
- [39] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in Neural Information Processing Systems* 28 (2015), 1135–1143.
- [40] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Sathesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. 2014. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567* (2014).

- [41] Awni Y. Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H. Tison, Codie Bourn, Mintu P. Turakhia, and Andrew Y. Ng. 2019. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature Medicine* 25, 1 (2019), 65–69.
- [42] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*. 2961–2969.
- [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*. 770–778.
- [44] Yihui He, Xiangyu Zhang, and Jian Sun. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 1389–1397.
- [45] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara 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.
- [46] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2014. Distilling the knowledge in a neural network. In *Deep Learning and Representation Learning Workshop: NIPS 2014*.
- [47] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. 2019. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1314–1324.
- [48] 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).
- [49] Diyi Hu and Bhaskar Krishnamachari. 2020. Fast and accurate streaming CNN inference via communication compression on the edge. In *2020 IEEE/ACM 5th International Conference on Internet-of-Things Design and Implementation (IoTDI'20)*. IEEE, 157–163.
- [50] Gao Huang, Danlu Chen, T. Li, Felix Wu, L. V. D. Maaten, and Kilian Q. Weinberger. 2018. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*.
- [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*. 4700–4708.
- [52] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*. PMLR, 448–456.
- [53] Sohei Itahara, Takayuki Nishio, and Koji Yamamoto. 2021. Packet-loss-tolerant split inference for delay-sensitive deep learning in lossy wireless networks. *arXiv preprint arXiv:2104.13629* (2021).
- [54] Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. [n.d.]. First Quora Dataset Release: Question Pairs. Retrieved January 25, 2021, from <https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs>.
- [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*. 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] Mikolaj Jankowski, Deniz Gündüz, and Krystian Mikolajczyk. 2020. Joint device-edge inference over wireless links with pruning. In *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC'20)*. IEEE, 1–5.
- [58] Hyuk-Jin Jeong, InChang Jeong, Hyeon-Jae Lee, and Soo-Mook Moon. 2018. Computation offloading for machine learning web apps in the edge server environment. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS'18)*. 1492–1499.
- [59] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 8018–8025.
- [60] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*. 615–629. <https://doi.org/10.1145/3037697.3037698>
- [61] Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 1317–1327.
- [62] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*.

- [63] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. (2009). <http://www.cs.toronto.edu/~kriz/cifar.html>.
- [64] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), 1097–1105.
- [65] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).
- [66] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. AL-BERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*.
- [67] 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*. 1–15.
- [68] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.
- [69] 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.
- [70] Joo Chan Lee, Yongwoo Kim, SungTae Moon, and Jong Hwan Ko. 2021. A splittable DNN-based object detector for edge-cloud collaborative real-time video inference. In *2021 17th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS'21)*. IEEE, 1–8.
- [71] Hector J. Levesque, Ernest Davis, and Leora Morgenstern. 2012. The Winograd schema challenge. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning*. 552–561.
- [72] Gil Levi and Tal Hassner. 2015. Age and gender classification using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 34–42.
- [73] Guangli Li, Lei Liu, Xueying Wang, Xiao Dong, Peng Zhao, and Xiaobing Feng. 2018. Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge. In *International Conference on Artificial Neural Networks*. 402–411.
- [74] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient ConvNets. In *4th International Conference on Learning Representations*.
- [75] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2017. Pruning filters for efficient ConvNets. In *5th International Conference on Learning Representations*.
- [76] He Li, Kaoru Ota, and Mianxiong Dong. 2018. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE Network* 32, 1 (2018), 96–101.
- [77] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. 2019. Improved techniques for training adaptive deep networks. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV'19)*. 1891–1900.
- [78] Jinyu Li, Rui Zhao, Jui-Ting Huang, and Yifan Gong. 2014. Learning small-size DNN with output-distribution-based criteria. In *15th Annual Conference of the International Speech Communication Association*.
- [79] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. 2020. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on Machine Learning*. PMLR, 5958–5968.
- [80] Min Lin, Qiang Chen, and Shuicheng Yan. 2014. Network in network. In *Second International Conference on Learning Representations*. (2014).
- [81] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2117–2125.
- [82] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*. 2980–2988.
- [83] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft Coco: Common objects in context. In *European Conference on Computer Vision*. Springer, 740–755.
- [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*. 6035–6044.
- [85] 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).
- [86] Zejian Liu, Fanrong Li, Gang Li, and Jian Cheng. 2021. EBERT: Efficient BERT inference with dynamic structured pruning. In *Findings of the Association for Computational Linguistics (ACL-IJCNLP'21)*. 4814–4823.

- [87] 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 *2017 IEEE International Conference on Computer Design (ICCD'17)*. 273–280.
- [88] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B. Letaief. 2017. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials* 19, 4 (2017), 2322–2358.
- [89] Pablo Jiménez Mateo, Claudio Fiandrino, and Joerg Widmer. 2019. Analysis of TCP performance in 5G mm-wave mobile networks. In *2019 IEEE International Conference on Communications (IEEE ICC'19)*. IEEE, 1–7.
- [90] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. 2019. Distilled split deep neural networks for edge-assisted real-time systems. In *Proceedings of the 2019 MobiCom Workshop on Hot Topics in Video Analytics and Intelligent Edges*. 21–26.
- [91] Yoshitomo Matsubara, Davide Callegaro, Sabur Baidya, Marco Levorato, and Sameer Singh. 2020. Head network distillation: Splitting distilled deep neural networks for resource-constrained edge computing systems. *IEEE Access* 8 (2020), 212177–212193. <https://doi.org/10.1109/ACCESS.2020.3039714>
- [92] Yoshitomo Matsubara, Davide Callegaro, Sameer Singh, Marco Levorato, and Francesco Restuccia. 2022. BottleFit: Learning compressed representations in deep neural networks for effective and efficient split computing. *arXiv preprint arXiv:2201.02693* (2022).
- [93] Yoshitomo Matsubara and Marco Levorato. 2020. Split computing for complex object detectors: Challenges and preliminary results. In *Proceedings of the 4th International Workshop on Embedded and Mobile Deep Learning*. 7–12.
- [94] Yoshitomo Matsubara and Marco Levorato. 2021. Neural compression and filtering for edge-assisted real-time object detection in challenged networks. In *2020 25th International Conference on Pattern Recognition (ICPR'21)*. 2272–2279.
- [95] Yoshitomo Matsubara, Thuy Vu, and Alessandro Moschitti. 2020. Reranking for efficient transformer-based answer selection. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1577–1580.
- [96] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. 2022. SC2: Supervised compression for split computing. *arXiv preprint arXiv:2203.08875* (2022).
- [97] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, and Stephan Mandt. 2022. Supervised compression for resource-constrained edge computing systems. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2685–2695.
- [98] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. 2020. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5191–5198.
- [99] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
- [100] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*. 807–814.
- [101] Mutsuki Nakahara, Daisuke Hisano, Mai Nishimura, Yoshitaka Ushiku, Kazuki Maruta, and Yu Nakayama. 2021. Retransmission edge computing system conducting adaptive image compression based on image recognition accuracy. In *2021 IEEE 94rd Vehicular Technology Conference (VTC'21-Fall)*. IEEE, 1–5.
- [102] Katayoun Neshatpour, Farnaz Behnia, Houman Homayoun, and Avesta Sasan. 2019. Exploiting energy-accuracy trade-off through contextual awareness in multi-stage convolutional neural networks. In *20th International Symposium on Quality Electronic Design (ISQED'19)*. 265–270.
- [103] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. [n.d.]. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.
- [104] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. In *CoCo@ NIPS*.
- [105] 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.
- [106] Daniele Jahier Pagliari, Roberta Chiaro, Enrico Macii, and Massimo Poncino. 2020. CRIME: Input-dependent collaborative inference for recurrent neural networks. *IEEE Transactions on Computers* 70, 10 (2020), 1626–1639.
- [107] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. LibriSpeech: An ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'15)*. IEEE, 5206–5210.
- [108] Mary Phuong and Christoph H. Lampert. 2019. Distillation-based training for multi-exit architectures. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV'19)*. 1355–1364.

- [109] Jary Pomponi, Simone Scardapane, and Aurelio Uncini. 2021. A probabilistic re-interpretation of confidence scores in multi-exit models. *Entropy* 24, 1 (2021), 1.
- [110] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. 2018. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)* 51, 5 (2018), 1–36.
- [111] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. 2011. The Kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society.
- [112] Y. Qiu, Hongzheng Li, Shen Li, Yingdi Jiang, Renfen Hu, and L. Yang. 2018. Revisiting correlations between intrinsic and extrinsic evaluations of word embeddings. In *CCL*.
- [113] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10428–10436.
- [114] 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*. 2383–2392.
- [115] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: Better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7263–7271.
- [116] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).
- [117] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*. 91–99.
- [118] 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. <https://doi.org/10.1109/MCOM.001.2000243>
- [119] Gemma Roig, Xavier Boix, Horeh Ben Shitrit, and Pascal Fua. 2011. Conditional random fields for multi-camera object detection. In *2011 International Conference on Computer Vision*. IEEE, 563–570.
- [120] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.
- [121] Farzad Samie, Lars Bauer, and Jörg Henkel. 2016. IoT technologies for embedded computing: A survey. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS'16)*. IEEE, 1–10.
- [122] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.
- [123] Andrew M. Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D. Tracey, and David D. Cox. 2019. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment* 2019, 12 (2019), 124020.
- [124] Marion Sbai, Muhamad Risqi U. Saputra, Niki Trigoni, and Andrew Markham. 2021. Cut, distil and encode (CDE): Split cloud-edge deep inference. In *2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON'21)*. IEEE, 1–9.
- [125] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. 2014. OverFeat: Integrated recognition, localization and detection using convolutional networks. In *2nd International Conference on Learning Representations*.
- [126] Jiawei Shao and Jun Zhang. 2020. BottleNet++: An end-to-end approach for feature compression in device-edge co-inference systems. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops'20)*. IEEE, 1–6.
- [127] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354.
- [128] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations*.
- [129] Amarjot Singh, Devendra Patil, and S. N. Omkar. 2018. Eye in the sky: Real-time drone surveillance system (DSS) for violent individuals identification using ScatterNet hybrid deep learning network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 1629–1637.
- [130] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*. 4077–4087.

- [131] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 1631–1642.
- [132] 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*. 5697–5708.
- [133] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
- [134] Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. 2021. How to train your ViT? Data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270* (2021).
- [135] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1–9.
- [136] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2818–2826.
- [137] Yaniv Taigman, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1701–1708.
- [138] 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 Conference on Computer Vision and Pattern Recognition*. 2820–2828.
- [139] Mingxing Tan, Ruoming Pang, and Quoc V. Le. 2020. EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10781–10790.
- [140] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR’16)*. IEEE, 2464–2469.
- [141] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2017. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS’17)*. 328–339.
- [142] Naftali Tishby, Fernando C. Pereira, and William Bialek. 2000. The information bottleneck method. *arXiv preprint physics/0004057* (2000).
- [143] Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW’15)*. IEEE, 1–5.
- [144] Ultralytics. [n.d.]. YOLOv5. <https://github.com/ultralytics/yolov5>.
- [145] 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 *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc., 5998–6008.
- [146] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.
- [147] Fei Wang, Boyu Diao, Tao Sun, and Yongjun Xu. 2020. Data security and privacy challenges of computing offloading in FINS. *IEEE Network* 34, 2 (2020), 14–20.
- [148] Meiqi Wang, Jianqiao Mo, Jun Lin, Zhongfeng Wang, and Li Du. 2019. DynExit: A dynamic early-exit strategy for deep residual networks. In *2019 IEEE International Workshop on Signal Processing Systems (SiPS’19)*. IEEE, 178–183.
- [149] Mengqiu Wang, Noah A. Smith, and Teruko Mitamura. 2007. What is the jeopardy model? A quasi-synchronous grammar for QA. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL’07)*. 22–32.
- [150] Yue Wang, Jianghao Shen, Ting-Kuei Hu, P. 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 (2020), 623–633.
- [151] Alex Warstadt, Amanpreet Singh, and Samuel Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics* 7 (2019), 625–641.
- [152] Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 1112–1122.

- [153] Maciej Wolczyk, Bartosz Wójcik, Klaudia Bałazy, Igor Podolak, Jacek Tabor, Marek Śmieja, and Tomasz Trzcinski. 2021. Zero time waste: Recycling predictions in early exit neural networks. *Advances in Neural Information Processing Systems* 34 (2021).
- [154] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 38–45.
- [155] Ji Xin, Rodrigo Nogueira, Yaoliang Yu, and Jimmy Lin. 2020. Early exiting BERT for efficient document ranking. In *Proceedings of SustainNLP: Workshop on Simple and Efficient Natural Language Processing*. 83–88.
- [156] Ji Xin, Raphael Tang, Jaeyun 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*. 2246–2251.
- [157] Qunliang Xing, Mai Xu, Tianyi Li, and Zhenyu Guan. 2020. Early exit or not: Resource-efficient blind quality enhancement for compressed images. In *Computer Vision (ECCV'20)*. Springer International Publishing.
- [158] L. Yang, Yizeng Han, X. Chen, Shiji Song, Jifeng Dai, and Gao Huang. 2020. Resolution adaptive networks for efficient inference. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'20)*. 2366–2375.
- [159] Taojiannan Yang, Sijie Zhu, Chen Chen, Shen Yan, Mi Zhang, and Andrew Willis. 2020. MutualNet: Adaptive convnet via mutual learning from network width and resolution. In *European Conference on Computer Vision*. Springer, 299–315.
- [160] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5687–5695.
- [161] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 285–300.
- [162] Yibo Yang, Robert Bamler, and Stephan Mandt. 2020. Variational Bayesian quantization. In *International Conference on Machine Learning*. PMLR, 10670–10680.
- [163] Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. WikiQA: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2013–2018.
- [164] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 476–488.
- [165] Shujian Yu and Jose C. Principe. 2019. Understanding autoencoders with information theoretic concepts. *Neural Networks* 117 (2019), 104–123.
- [166] Shujian Yu, Kristoffer Wickstrøm, Robert Jenssen, and José C. Principe. 2020. Understanding convolutional neural networks with information theory: An initial exploration. *IEEE Transactions on Neural Networks and Learning Systems* Vol. 32, 1 (2020), 435–442.
- [167] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC'16)*. BMVA Press, 87.1–87.12.
- [168] Liekang Zeng, En Li, Zhi Zhou, and X. Chen. 2019. Boomerang: On-demand cooperative deep neural network inference for edge intelligence on the industrial Internet of Things. *IEEE Network* 33 (2019), 96–103.
- [169] 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.
- [170] Shizhou Zhang, Qi Zhang, Yifei Yang, Xing Wei, Peng Wang, Bingliang Jiao, and Yanning Zhang. 2020. Person re-identification in aerial imagery. *IEEE Transactions on Multimedia* 23 (2020), 281–291.
- [171] X. Zhang, J. Zhao, and Y. LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*.
- [172] 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.
- [173] Barret Zoph and Quoc Le. 2017. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.
- [174] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8697–8710.

Received 7 March 2021; revised 15 November 2021; accepted 14 March 2022