Dynamic Neural Networks: A Survey

Yizeng Han*, Gao Huang*, Member, IEEE, Shiji Song, Senior Member, IEEE, Le Yang, Honghui Wang, and Yulin Wang

Abstract—Dynamic neural network is an emerging research topic in deep learning. Compared to static models which have fixed computational graphs and parameters at the inference stage, dynamic networks can adapt their structures or parameters to different inputs, leading to notable advantages in terms of accuracy, computational efficiency, adaptiveness, etc. In this survey, we comprehensively review this rapidly developing area by dividing dynamic networks into three main categories: 1) sample-wise dynamic models that process each sample with data-dependent architectures or parameters; 2) spatial-wise dynamic networks that conduct adaptive computation with respect to different spatial locations of image data; and 3) temporal-wise dynamic models that perform adaptive inference along the temporal dimension for sequential data such as videos and texts. The important research problems of dynamic networks, e.g., architecture design, decision making scheme, optimization technique and applications, are reviewed systematically. Finally, we discuss the open problems in this field together with interesting future research directions.

Index Terms—Dynamic networks, Adaptive inference, Efficient inference, Convolutional neural networks.

1 INTRODUCTION

Deep neural networks (DNNs) are playing an important role in various areas including computer vision (CV) [1, 2, 3, 4, 5] and natural language processing (NLP) [6, 7, 8]. Recent years have witnessed many successful deep models such as AlexNet [1], VGG [2], GoogleNet [3], ResNet [4], DenseNet [5] and Transformer [6]. These architectural innovations have enabled the training of deeper, more accurate and more efficient models. The recent research on neural architecture search (NAS) [9, 10] further speeds up the process of designing more powerful structures. However, most of the prevalent deep learning models perform inference in a static manner, i.e., both the computational graph and the network parameters are fixed once trained, which may limit their representation power, efficiency and interpretability [11, 12, 13, 14].

Dynamic networks, as opposed to static ones, can adapt their structures or parameters to the input during inference, and therefore enjoy favorable properties that are absent in static models. In general, dynamic computation in the context of deep learning has the following advantages:

1) Efficiency. One of the most notable advantages of dynamic networks is that they are able to allocate computations on demand at test time, by selectively activating model components (e.g. layers [12], channels [15] or sub-networks [16]) conditioned on the input. Consequently, less computation is spent on canonical samples that are relatively easy to recognize, or on less informative spatial/temporal locations of an input. In addition to computational efficiency, dynamic models have also shown promising results for improving data efficiency in the scenario of few-shot learning [17, 18].

2) Representation power. Due to the data-dependent network architecture/parameters, dynamic networks have significantly enlarged parameter space and improved representation power. For example, with a minor increase of computation, model capacity can be boosted by applying feature-conditioned attention weights on an ensemble of convolutional kernels [13, 19]. It is worth noting that the popular soft attention mechanism could also be unified in the framework of dynamic networks, as different channels [20], spatial areas [21] or temporal locations [22] of features are dynamically re-weighted at test time.

3) Adaptiveness. Dynamic models are able to achieve a desired trade-off between accuracy and efficiency for dealing with varying computational budgets on the fly. Therefore, they are more adaptable to different hardware platforms and changing environments, compared to static models with a fixed computational cost.

4) Compatibility. Dynamic networks are compatible with most advanced techniques in deep learning, including architecture design [4, 5], optimization algorithms [23, 24] and data preprocessing [25, 26], which ensures that they can benefit from the most recent advances in the field to achieve state-of-the-art performance. For example, dynamic networks can inherit architectural innovations in lightweight models [27], or be designed via NAS approaches [9, 10]. Their efficiency could also be further improved by acceleration methods developed for static models, such as network pruning [28], weight quantization [29], knowledge distillation [30] and low-rank approximation [31].

5) Generality. As a substitute for static deep learning techniques, many dynamic models are general approaches that can be applied seamlessly to a wide range of applications, such as image classification [12], [32], object detection [33] and semantic segmentation [34]. Moreover, the techniques developed in CV tasks are proven to transfer well to language models in NLP tasks [35], [36], and vice versa.

6) Interpretability. We finally note that the research on dynamic networks may bridge the gap between the underlying mechanism of deep models and brains, as it is believed that the brains process information in a dynamic way [37, 38]. It is possible to analyze which components of a dynamic model are activated [32] when processing an input sample, and to observe which parts of the input are ac-
Fig. 1. Overview of the survey. We first review the dynamic networks that perform adaptive computation at three different granularities (i.e. sample-wise, spatial-wise and temporal-wise). Then we summarize the decision making strategy, training technique and applications of dynamic models. Existing open problems in this field together with some future research directions are finally discussed. Best viewed in color.

Fig. 2. Two early-exiting schemes. The dashed lines and shaded modules are not executed, conditioned on the decisions made by the routers.

countable for certain predictions [39]. These properties may shed light on interpreting the decision process of DNNs.

In fact, adaptive inference, the key idea underlying dynamic networks, has been studied before the popularity of modern DNNs. The most classical approach is building a model ensemble through a cascaded [40] or parallel [41] structure, and selectively activating the models conditioned on the input. Spiking neural networks (SNNs) [42], [43] also perform data-dependent inference by propagating pulse signals. However, the training strategy for SNNs is quite different from that of popular convolutional neural networks (CNNs), and they are less used in vision tasks. Therefore, we leave out the work related to SNNs in this survey.

In the context of deep learning, dynamic inference with modern deep architectures, has raised many new research questions and has attracted great research interests in the past three years. Despite the extensive work on designing various types of dynamic networks, a systematic and comprehensive review on this topic is still lacking. This motivates us to write this survey, to review the recent advances in this rapidly developing area, with the purposes of 1) providing an overview as well as new perspectives for researchers who are interested in this topic; 2) pointing out the close relations of different subareas and reducing the risk of reinventing the wheel; and 3) summarizing the key challenges and possible future research directions.

This survey is organized as follows (see Fig. 1 for an overview). In Sec. 2, we introduce the most common sample-wise dynamic networks which adapt their architectures or parameters conditioned on each input sample. Dynamic models working on a finer granularity, i.e., spatially adaptive and temporally adaptive models, are reviewed in Sec. 3 and Sec. 4, respectively. Then we investigate the decision making strategies and the training techniques of dynamic.

1. These two categories can also be viewed as sample-wise dynamic networks as they perform adaptive computation within each sample at a finer granularity, and we adopt such a split for narrative convenience.
networks in Sec. 5. The applications of dynamic models are further summarized in Sec. 6. Finally, we conclude this survey with a discussion on a number of open problems and future research directions in Sec. 7. For better readability, we list the notations that will be used in this survey in Table 1.

2 Sample-wise Dynamic Networks

Aiming at processing different inputs in data-dependent manners, sample-wise dynamic networks are typically designed from two perspectives: 1) adjusting model architectures to allocate appropriate computation based on each sample, and therefore reducing redundant computation for increased efficiency (Sec. 2.1.2), 2) adapting network parameters to every input sample with fixed computational graphs, with the goal of boosting the representation power with minimal increase of computational cost (Sec. 2.2).

2.1 Dynamic Architectures

Considering different inputs may have diverse computational demands, it is natural to perform inference with dynamic architectures conditioned on each sample. Specifically, one can adjust the network depth (Sec. 2.1.1), width (Sec. 2.1.2), or perform dynamic routing within a super network (SuperNet) that includes multiple possible paths (Sec. 2.1.3). Networks with dynamic architectures not only save redundant computation for canonical (“easy”) samples, but also preserve their representation power when recognizing non-canonical (“hard”) samples. Such a property leads to remarkable advantages in efficiency compared to the acceleration techniques for static models [29], [29], [44], which handle “easy” and “hard” inputs with identical computation, and fail to reduce intrinsic computational redundancy.

2.1.1 Dynamic Depth

As modern DNNs are getting increasingly deep for recognizing more “hard” samples, a straightforward solution to reducing redundant computation is performing inference with dynamic depth, which can be realized by 1) early exiting, i.e. allowing “easy” samples to be output at shallow exits without executing deeper layers [12], [45], [46]; or 2) layer skipping, i.e. selectively skipping intermediate layers conditioned on each sample [11], [47], [48].

1) Early exiting. The complexity (or “difficulty”) of input samples varies in most real-world scenarios, and shallow networks are capable of correctly identifying many canonical inputs. Ideally, these samples should be output at certain early exits without executing deeper layers.

For an input sample \( x \), the forward propagation of an \( L \)-layer deep network \( F \) could be represented by

\[
y = F^L \circ F^{L-1} \circ \cdots \circ F^1(x),
\]

where \( F^\ell \) denotes the operational function at layer \( \ell \), \( 1 \leq \ell \leq L \). In contrast, early exiting allows to terminate the inference procedure at an intermediate layer. For the \( i \)-th input sample \( x_i \), the forward propagation can be written as

\[
y_i = F_i^\ell \circ F_i^{\ell-1} \circ \cdots \circ F_i^1(x_i), 1 \leq \ell_i \leq L.
\]

Note that \( \ell_i \) is adaptively determined based on \( x_i \). Extensive architectures have been studied to endow DNNs with such early exiting behaviors, as discussed in the following.

a) Cascading of DNNs. The most intuitive approach to enabling early exiting is cascading multiple models (see Fig. 2 (a)), and adaptively retrieving the prediction of an early network without activating latter ones. For example, Big/little-Net [49] cascades two CNNs with different depths. After obtaining the SoftMax output of the first model, early exiting is conducted when the score margin between the two largest elements exceeds a threshold. Moreover, a number of classic CNNs [1], [3], [4] are cascaded in [46] and [50]. After each model, a decision function is trained to determine whether the obtained feature should be fed to a linear classifier for immediate prediction, or be sent to subsequent models.

b) Intermediate classifiers. The models in the aforementioned cascading structures are mutually independent. Consequently, once a “difficult” sample is decided to be fed to a latter network, a whole inference procedure needs to be executed from scratch without reusing the already learned features. A more compact design is involving intermediate classifiers within one backbone network (see Fig. 2 (b)), so that early features can be propagated to deep layers if needed. Based on such a multi-exit architecture, adaptive early exiting is typically achieved according to confidence-based criteria [45], [51] or learned functions [46], [52], [53].

c) Multi-scale architecture with early exits. Researchers [12] have observed that in chain-structured networks, the multiple classifiers may interfere with each other, which degrades the overall performance. A reasonable interpretation could be that in regular CNNs, the high-resolution features lack the coarse-level information that is essential for classification, leading to unsatisfying results for early exits. Moreover, early classifiers would force the shallow layers to generate task-specialized features, while a part of general information is lost, resulting in degraded performance for deep exits. To tackle this issue, multi-scale dense network (MSDNet) [12] adopts 1) a multi-scale architecture, which consists of multiple sub-networks for processing feature maps with different resolutions (scales), to quickly generate coarse-level features that are suitable for classification; 2) dense connections, to reuse early features and improve the performance of deep classifiers (see Fig. 3 (a)). Such a specially-designed architecture effectively enhances the
Overall accuracy of all the classifiers in the network.

Based on the multi-scale architecture design, researchers have also studied the exiting policies \([54], [55]\) (see Fig. 3(b)) and training schemes \([56]\) of early-exiting dynamic models. More discussion about the inference and training schemes for dynamic models will be presented in Sec. 5.

Previous methods typically achieve the adaptation of network depths. From the perspective of exploiting spatial redundancy in features, resolution adaptive network (RANet, see Fig. 3(c)) \([32]\) first processes each sample with low-resolution features, while high-resolution representations are conditionally utilized based on early predictions.

Adaptive early exiting is also extended to language models (e.g. BERT \([7]\)) for improving their efficiency on NLP tasks \([37], [38], [39], [60]\). In addition, it can be implemented in recurrent neural networks (RNNs) for temporally dynamic inference when processing sequential data such as videos \([61], [62]\) and texts \([63], [64], [65]\) (see Sec. 4).

2) **Layer skipping**. The general idea of the aforementioned early-exiting paradigm is skipping the execution of all the deep layers after a certain classifier. More flexibly, the network depth can also be adapted on the fly by strategically skipping the calculation of intermediate layers without placing extra classifiers. Given the \(i\)-th input sample \(x_i\), dynamic layer skipping could be generally written as

\[
y_i = (1^L \circ \mathcal{F}^L) \circ (1^{L-1} \circ \mathcal{F}^{L-1}) \circ \cdots \circ (1^1 \circ \mathcal{F}^1)(x_i),
\]

where \(1^\ell\) denotes the indicator function determining the execution of layer \(\mathcal{F}^\ell, 1 \leq \ell \leq L\). This scheme is typically implemented on structures with skip connections (e.g. ResNet \([4]\)) to guarantee the continuity of forward propagation, and here we summarize three common approaches.

a) **Halting score** is first proposed in \([11]\), where an accumulated scalar named halting score adaptively decides whether the hidden state of an RNN will be directly fed to the next time step. The halting scheme is extended to vision tasks by viewing residual blocks within a ResNet \([33]\) (see Fig. 4(a)). Rather than skipping the execution of layers with independent parameters, multiple blocks in each ResNet stage could be replaced by one weight-sharing block, leading to a significant reduction of parameters \([66]\). In every stage, the block is executed for an adaptive number of steps according to the halting score.

In addition to RNNs and CNNs, the halting scheme is further implemented on Transformers \([6]\) by \([25]\) and \([36]\) to achieve dynamic network depth on NLP tasks.

b) **Gating function** is also a prevalent option for dynamic layer skipping due to its plug-and-play property. Take ResNet as an example (see Fig. 4(b)), let \(x^\ell\) denote the input feature of the \(\ell\)-th residual block, gating function \(\mathcal{G}^\ell\) generates a binary value to decide the execution of residual block \(\mathcal{F}^\ell\). This procedure could be represented by

\[
x^{\ell+1} = g^\ell(x^\ell) \mathcal{F}^\ell(x^\ell) + x^\ell, g^\ell(x^\ell) \in \{0, 1\},
\]

SkipNet \([47]\) and convolutional network with adaptive inference graph (Conv-AIG) \([48]\) are two typical approaches to enabling dynamic layer skipping. Both methods induce lightweight computational overheads to efficiently produce the binary decisions on whether skipping the calculation of a residual block. Specifically, Conv-AIG utilizes two FC layers in each residual block, while the gating function in SkipNet is implemented as an RNN for parameter sharing.

Rather than skipping layers in classic ResNets, dynamic recursive network \([67]\) iteratively executes one block with shared parameters in each stage. Although the weight-sharing scheme seems similar to the aforementioned IamNN \([66]\), the skipping policy of \([67]\) differs significantly: gating modules are exploited to decide the recursion depth.

Instead of either skipping a layer, or executing it thoroughly with a full numerical precision, a line of work \([68], [69]\) studies adaptive bit-width for different layers conditioned on the resource budget. Furthermore, fractional skipping \([70]\) adaptively selects a bit-width for each residual block by a gating function based on input features.

c) **Policy network** can be built to take in an input sample, and directly produces the skipping decisions for all the layers in a backbone network \([71]\) (see Fig. 4(c)).

2.1.2 **Dynamic Width**

In addition to dynamic network depth (Sec. 2.1.1), a finer-grained form of conditional computation is performing inference with dynamic width: although every layer is executed, its multiple units (e.g. neurons, channels or branches) are selectively activated conditioned on the input.

1) **Skipping neurons in fully-connected (FC) layers**. The computational cost of an FC layer is determined by its input and output dimensions. It is commonly believed that different neuron units are responsible for representing different features, and therefore not all of them need to be activated for every sample. Early studies learn to adaptively control the neuron activations by auxiliary branches \([72], [73], [74]\) or other techniques such as low-rank approximation \([75]\).

2) **Skipping branches in mixture-of-experts (MoE)**. In Sec. 2.1.1 adaptive model ensemble is achieved via a cascading way, and later networks are conditionally executed based on early predictions. An alternative approach to improving the model capacity is the MoE \([41], [76]\) structure, which means that multiple network branches are built as experts in parallel. These experts could be selectively executed, and their outputs are fused with data-dependent weights.

Conventional soft MoEs \([41], [76], [77]\) adopt real-valued weights to dynamically rescale the representations obtained.

2. Here we refer to a stage as a stack of multiple residual blocks with the same feature resolution.

3. For simplicity and without generality, the subscript for sample index will be omitted in the following.
from different experts (Fig. 5 (a)). In this way, all the branches still need to be executed, and thus the computation cannot be reduced at test time. Hard gates with only a fraction of non-zero elements are developed to increase the inference efficiency of the MoE structure (see Fig. 5 (b)) [78], [79], [80]. Let \( G \) denote a gating module whose output is a \( N \)-dimensional vector \( \alpha \) controlling the execution of \( N \) experts \( F_1, F_2, \ldots, F_N \), the final output can be written as

\[
y = \sum_{n=1}^{N} [G(x)]_n F_n(x) = \sum_{n=1}^{N} \alpha_n F_n(x),
\]

and the \( n \)-th expert will not be executed if \( \alpha_n = 0 \).

Hard MoE has been implemented in diverse network structures. For example, HydraNet [78] replaces the convolutional blocks in a CNN by multiple branches, and selectively execute them conditioned on the input. For another example, dynamic routing network (DRNet) [82] performs a branch selection in each cell structure which is commonly used in NAS [10]. On NLP tasks, sparsely gated MoE [16] and switch Transformer [81] embeds hard MoE in a long short-term memory (LSTM) [82] network and a Transformer [6], respectively. Instead of making choice with binary gates as in [80], only the branches corresponding to the top-K elements of the real-valued gates are activated in [16], [78], [81].

**3) Skipping channels in CNNs.** Modern CNNs usually have considerable channel redundancy. Based on the common belief that the same feature channel can be of disparate importance for different samples, adaptive width of CNNs could be realized by dynamically activating convolutional channels. Compared to the static pruning methods [28], [44] which remove “unimportant” filters permanently, such a data-dependent pruning approach improves the inference efficiency without degrading the model capacity.

a) **Multi-stage architectures along the channel dimension.** Recall that the early-exiting networks [12], [32] discussed in Sec. 2.1.1 can be viewed as multi-stage models along the depth dimension, where late stages are conditionally executed based on early predictions. One can also build multi-stage architectures along the width (channel) dimension, and progressively execute these stages on demand.

Along this direction, an optimal architecture is searched among multiple structures with different widths, and any sample can be output at an early stage when a confident prediction is obtained [83]. Channel gating network (CGNet) [84] first executes a subset of convolutional filters in every layer, and the remaining filters are only activated on strategically selected areas.

b) **Dynamic pruning with gating functions.** In the aforementioned progressive activation paradigm, the execution of a later stage is decided based on previous output. As a result, a complete forward propagation is required for every stage, which might be suboptimal for reducing the practical inference latency. Another prevalent solution is to decrease the execution of channels at every layer by gating functions. For example, runtime neural pruning (RNP) [15] models the layer-wise pruning as a Markov decision process, and an RNN is used to select specific channel groups at every layer. Moreover, pooling operations followed by FC layers are utilized to generate channel-wise hard attention (i.e. making discrete decisions on whether to activate each channel) for each sample [85], [86], [87], [88]. The recent work [89] uses a gate module to decide the width for a whole stage of a ResNet. Different reparameterization and optimizing techniques are required for training these gating functions, which will be reviewed in Sec. 5.2.

Rather than placing plug-in gating modules inside a CNN, GaterNet [90] builds an extra network, which takes in the input sample and directly generates all the channel selection decisions for the backbone CNN. This implementation is similar to BlockDrop [71] that exploits an additional policy network for dynamic layer skipping (Sec. 2.1.1).

c) **Dynamic pruning based on feature activations directly** has also been realized without auxiliary branches and computational overheads [91], where a regularization item is induced in training to encourage the sparsity of features.

On basis of the existing literature on dynamically skipping either network layers [47], [48] or convolutional filters [15], [85], [86], [87], recent work [92], [93], [94] has realized dynamic inference with respect to network depth and width simultaneously: only if a layer is determined to be executed, its channels will be selectively activated, leading to a more flexible adaptation of computational graphs.

2.1.3 Dynamic Routing

The aforementioned methods mostly adjust the depth (Sec. 2.1.1) or width (Sec. 2.1.2) of classic architectures by activating their computational units (e.g. layers [47], [48] or channels [15], [87]) conditioned on the input. Another line of work develops different forms of SuperNets with various possible inference paths, and performs dynamic routing inside the SuperNets to adapt the computational graph to each sample.

To achieve dynamic routing, there are typically routing nodes that are responsible for allocating features to different paths. For node \( s \) at layer \( \ell \), let \( \alpha_{s \rightarrow j}^{\ell} \) denote the probability of assigning the reached feature \( x_{s}^{\ell} \) to node \( j \) at layer \( \ell + 1 \), the path \( F_{s \rightarrow j}^{\ell} \) will be activated only when \( \alpha_{s \rightarrow j}^{\ell} > 0 \). The resulting feature reaching node \( j \) is represented by

\[
x_{j}^{\ell+1} = \sum_{s \in \{ \alpha_{s \rightarrow j}^{\ell} > 0 \}} \alpha_{s \rightarrow j}^{\ell} F_{s \rightarrow j}^{\ell}(x_{s}^{\ell}).
\]

The probability \( \alpha_{s \rightarrow j}^{\ell} \) can be obtained in different manners. Note that the dynamic early-exiting networks [12], [32] are a special form of SuperNets, where the routing decisions are only made at intermediate classifiers. The CapsuleNets
also perform dynamic routing between capsules, i.e., groups of neurons, to characterize the relations between (parts of) objects. Here we mainly focus on specific architecture designs of the SuperNets and their routing policies.

1) Path selection in multi-branch structures. The simplest dynamic routing can be implemented by selectively executing one of multiple candidate modules at each layer [96], [97], which is equivalent to producing a one-hot probability distribution $\alpha_{l,s} \sim$ in Eq. 6. The main difference of this approach to hard MoE (Fig. 5 (b)) is that only one branch is activated without any fusion operations.

2) Neural trees and tree-structured networks. As decision trees always perform inference along one forward path that is dependent on input properties, combining tree structure with neural networks can naturally enjoy the adaptive inference paradigm and the representation power of DNNs simultaneously. Note that in a tree structure, the outputs of different nodes are routed to independent paths rather than being fused as in MoE structures (compare Fig. 5(b), (c)).

a) Soft decision tree (SDT) [98], [99], [100] adopts neural units as routing nodes (blue nodes in Fig. 5(c)), which decides the portion that the inputs are assigned to their left/right sub-tree. Each leaf node generates a probability distribution over the output space, and the final prediction is the expectation of the results from all leaf nodes. Although the probability for a sample reaching each leaf node in an SDT is data-dependent, all the paths are still executed, which limits the inference efficiency.

b) Neural trees with deterministic routing policies [101], [102] are designed to make hard routing decisions during inference, avoiding computation on those unselected paths.

c) Tree-structured DNNs. Instead of developing decision trees containing neural units, a line of work builds special network architectures to endow them with the routing behavior of decision trees. For instance, a small CNN is first executed to classify each sample into coarse categories, and specific sub-networks are conditionally activated based on the coarse predictions [103]. A subsequent work [104] only partitions samples to different sub-networks, but also divides and routes the feature channels.

Different to those networks using neural units only in routing nodes [101], [102], or routing each sample to pre-designed sub-networks [103], [104], adaptive neural tree (ANT) [105] adopts CNN modules as feature transformers in a hard neural tree (see lines with arrows in Fig. 5(c)), and learns the tree structure together with the network parameters simultaneously in the training stage.

3) Others. Performing dynamic routing within more general SuperNet architectures is also a recent research trend. Representatively, an architecture distribution with partly shared parameters is searched from a SuperNet containing $\sim 10^{25}$ sub-networks [106]. During inference, every sample is allocated by a controller network to one sub-network with appropriate computational cost. Instead of training a standalone controller network, gating modules are plugged inside the hand-designed SuperNet (see Fig. 3(d)) to decide the routing path based on intermediate features [107].

2.2 Dynamic Parameters

Although the networks with dynamic architectures in Sec. 2.1 can adapt their inference graphs to each sample and achieve an efficient allocation of computation, they usually have special architecture designs, requiring specific training strategies or careful hyper-parameters tuning (Sec. 2).

Another line of work adapts network parameters to different inputs while keeping the architectures fixed, which has been shown effective in improving the representation power of networks with a minor increase of computational cost. Given an input sample $x$, the output of a conventional network (module) with static parameters can be written as $y = F(x, \Theta)$. In contrast, the output of a model with dynamic parameters could be represented by

$$y = F(x, \hat{\Theta}; x) = F(x, \hat{W}(x, \Theta)),$$

where $\hat{W}(\cdot, \Theta)$ is the operation producing input-dependent parameters, and its design has been extensively explored.

In general, the parameter adaptation can be achieved from three aspects (see Fig. 6): 1) adjusting the trained parameters based on the input (Sec. 2.2.1); 2) directly generating the network parameters from the input (Sec. 2.2.2); and 3) rescaling the features with soft attention (Sec. 2.2.3).

2.2.1 Parameter Adjustment

A typical approach to parameter adaptation is adjusting the weights based on their input during inference as presented in Fig. 5(a). This implementation usually evokes little computation to obtain the adjustments, e.g., attention weights [13], [19], [108], [109] or sampling offsets [110], [111], [112].

1) Attention on weights. To improve the representation power without noticeably increasing the computation, soft attention can be performed on multiple convolutional kernels, producing an adaptive ensemble of parameters [13], [19]. Assuming that there are $N$ kernels $W_n$, $n = 1, 2, \cdots, N$, such a dynamic convolution can be formulated as

$$y = x \cdot \hat{W} = x \cdot \left( \sum_{n=1}^{N} \alpha_n W_n \right).$$

This procedure increases the model capacity yet remains high efficiency, as the result obtained through fusing the outputs of $N$ convolutional branches (as in MoE structures, see Fig. 5(a)) is equivalent to that produced by performing once convolution with $\hat{W}$. However, only $\sim 1/N$ times of computation is consumed in the latter approach.

Weight adjustment could also be achieved by performing soft attention over the spatial locations of convolutional weights [108], [109]. For example, segmentation-aware convolutional network [108] applies locally masked convolution to aggregate information with larger weights from similar pixels, which are more likely to belong to the same object. Unlike [108] that requires a sub-network for feature embedding, pixel-adaptive convolution (PAC) [109] adapts the convolutional weights based on the attention mask generated from the input feature at each layer.

Instead of adjusting weights conditioned on every sample itself, meta-neighborhoods [113] adapt the network parameters to each input sample based on its similarity to the neighborhoods stored in a dictionary.

2) Kernel shape adaptation. Apart from adaptively scaling the weight values, parameter adjustment can also be realized to reshape the convolutional kernels and achieve dynamic reception of fields. Towards this direction, deformable convolutions [110], [111] sample feature pixels from adaptive locations when performing convolution on each pixel. Deformable kernels [112] samples weights in the kernel space
to adapt the effective reception field (ERF) while leaving the reception field unchanged. Table 2 summarizes the formulations of the above three methods. Due to their irregular memory access and computation pattern, these kernel shape adaptation approaches typically require customized CUDA kernels for the implementation on GPUs. However, recent literature has shown that the practical efficiency of deformable convolution could be effectively improved by co-designing algorithm and hardware based on embedded devices such as FPGAs [114].

2.2.2 Weight Prediction

Compared to making modifications on model parameters on the fly (Sec. 2.2.1), weight prediction [115] is more straightforward: it directly generates (a subset of) input-adaptive parameters with an independent model at test time (see Fig. 6(b)). This idea was first suggested in [116], where both the weight prediction model and the backbone model were feedforward networks. Recent work has further extended the paradigm to modern network architectures and tasks.

1) General architectures. Dynamic filter networks (DFN) [117] and HyperNetworks [118] are two classic approaches realizing runtime weight prediction for CNNs and RNNs, respectively. Specifically, a filter generation network is built in DFN [117] to produce the filters for a convolutional layer. As for processing sequential data (e.g. a sentence), the weight matrices of the main RNN are predicted by a smaller layer. As for processing sequential data (e.g. a sentence), the weight matrices of the main RNN are predicted by a smaller layer.

LambdaNetworks [119] unifies the dynamic schemes of [113] and [20] by predicting the convolutional weights via simple grouped FC layers, achieving competitive results in terms of the accuracy-FLOPS and accuracy-parameters trade-offs.

Rather than generating standard convolutional weights, LambdaNetworks [120] learns to predict the weights of linear projections based on the contexts of each pixel together with the relative position embeddings, showing advantages in terms of computational cost and memory footprint.

2) Task-specific information has also been exploited to predict model parameters on the fly, enabling dynamic networks to generate task-aware feature embeddings. For example, edge attributes are utilized in [121] to generate filters for graph convolution, and camera perspective is incorporated in [122] to generate weights for image convolution. Such task-aware weight prediction has been shown effective in improving the data efficiency on many tasks, including visual question answering [123, 124] and few-shot learning [17], [18].

2.2.3 Dynamic Features

The main goal of either adjusting (Sec. 2.2.1) or predicting (Sec. 2.2.2) model parameters is producing more dynamic and informative features, and therefore enhancing the representation power of deep networks. A more straightforward solution is rescaling the features with input-dependent soft attention (see Fig. 6(c)), which requires minor modifications on computational graphs. Note that for a linear transformation \( F \), applying attention \( \alpha \) on the output is equivalent to performing computation with re-weighted parameters, i.e.

\[
F(x, \Theta) \otimes \alpha = F(x, \Theta \otimes \alpha).
\]

1) Channel-wise attention is one of the most common soft attention mechanisms. Existing work typically follows the form in squeeze-and-excitation network (SENet) [20]:

\[
\hat{y} = y \otimes \alpha = y \otimes A(y), \alpha \in [0, 1]^C.
\]

In Eq. (10) \( y = x \ast W \) is the output feature of a convolutional layer with \( C \) channels, and \( A(\cdot) \) is a lightweight function composed of pooling and linear layers for producing \( \alpha \). Taking the convolution into account, the procedure can also be written as \( \hat{y} = (x \ast W) \otimes \alpha = x \ast (W \otimes \alpha) \), from which we can observe that applying attention on features is equivalent to performing convolution with dynamic weights.

Other implementations for attention modules have also been developed, including using standard deviation to provide more statistics [125], or replacing FC layers with efficient 1D convolutions [126]. The empirical performance of three computational graphs for soft attention is studied in [127]: 1) \( \hat{y} = y \otimes A(y) \), 2) \( \hat{y} = y \otimes A(x) \) and 3) \( \hat{y} = y \otimes A(\text{Conv}(x)) \). It is found that the three forms yield different performance in different backbone networks.

2) Spatial-wise attention. Spatial locations in features could also be dynamically rescaled with attention to improve the representation power of deep models [128]. Instead of using pooling operations to efficiently gather global information as in channel-wise attention, convolutions are often adopted in spatial-wise attention to encode local information. Moreover, these two types of attention modules can be integrated in one framework [21], [129], [130], [131] (see Fig. 6(c)).

3) Dynamic activation functions. The aforementioned approaches to generating dynamic features usually apply soft attention before static activation functions. A recent line of work has sought to increase the representation power of models with dynamic activation functions [132], [133]. For instance, DY-ReLU [132] replaces ReLU \( (y_c = \max(x_c, 0)) \) with the max value among \( N \) linear transformations \( y_c = \max_n (a_n^c x_c + b_n^c) \), where \( c \) is the channel index, and \( a_n^c, b_n^c \) are linear coefficients calculated from \( x \). On many vision tasks, these dynamic activation functions can effectively improve the performance of different network architectures with negligible computational overhead.

To summarize, soft attention has been exploited in many fields due to its simplicity and effectiveness. Moreover, it can be incorporated with other methods conveniently. E.g., by replacing the weighting scalar \( \alpha_n \) in Eq. 5 with channel-wise [134] or spatial-wise [133] attention, the output of
multiple branches with independent kernel sizes [134] or feature resolutions [135] are adaptively fused.

Note that we leave out the detailed discussion on the self attention mechanism, which is widely studied in both NLP [4], [7] and CV fields [136], [137], [138] to re-weight features based on the similarity between queries and keys at different locations (temporal or spatial). Readers who are interested in this topic may refer to review studies [139], [140], [141].

In this survey, we mainly focus on the feature re-weighting scheme in the framework of dynamic inference.

3 Spatial-wise Dynamic Networks
In visual learning, it has been found that not all locations contribute equally to the final prediction of CNNs [142], which suggests that spatially dynamic computation has great potential for reducing computational redundancy. In other words, making a correct prediction may only require processing a fraction of pixels or regions with an adaptive amount of computation. Moreover, based on the observations that low-resolution representations are sufficient to yield decent performance for most inputs [27], the static CNNs that take in all the input with the same resolution may also induce considerable redundancy.

To this end, spatial-wise dynamic networks are built to perform adaptive inference with respect to different spatial locations of images. According to the granularity of dynamic computation, we further categorize the relevant approaches into three levels: pixel level (Sec. 3.1), region level (Sec. 3.2), and resolution level (Sec. 3.3).

3.1 Pixel-level Dynamic Networks
Commonly seen spatial-wise dynamic networks perform adaptive computation at the pixel level. Similar to the categorization in Sec. 2, pixel-level dynamic networks are grouped into two types: models with pixel-specific dynamic architectures (Sec. 3.1.1) and dynamic parameters (Sec. 3.1.2).

3.1.1 Pixel-wise Dynamic Architectures
Based on the common belief that foreground pixels are more informative and computational demanding than those in the background, some dynamic networks learn to adjust their architectures for each pixel. Existing literature generally achieves this by 1) dynamic sparse convolution, which only performs convolutions on a subset of sampled pixels; 2) additional refinement, which strategically allocates extra computation (e.g., layers or channels) on certain spatial positions.

1) Dynamic sparse convolution. To reduce the unnecessary computation on less informative locations, convolution can be performed only on strategically sampled pixels. Existing sampling strategies include 1) making use of the intrinsic sparsity of the input [143]; 2) predicting the positions of zero elements on the output [144], [145]; and 3) estimating the saliency of pixels [146], [147], [148]. A typical approach is using an extra branch to generate a spatial mask, determining the execution of convolution on each pixel (see Fig. 7).

Pixel-wise dynamic depth could also be achieved based on a halting scheme [33] (see Sec. 2.1.1). These dynamic convolutions usually neglect the unselected positions, which might degrade the network performance. Interpolation is utilized in [148] to efficiently fill those locations, therefore alleviating the aforementioned disadvantage.

2) Dynamic additional refinement. Instead of only sampling certain pixels to perform convolutions, another line of work first conducts relatively cheap computation on the whole feature map, and adaptively activate extra modules on selected pixels for further refinement. Representatively, dynamic capacity network [149] generates coarse features with a shallow model, and salient pixels are sampled based on the gradient information. For these salient pixels, extra layers are applied to extract finer features. Similarly, specific positions are additionally processed by a fraction of convolutional filters in [84]. These methods adapt their network architectures in terms of depth or width at the pixel level, achieving a spatially adaptive allocation of computation.

The aforementioned dynamic additional refinement approaches [84], [149] are mainly developed for image classification. On the semantic segmentation task, pixel-wise early exiting (see also Sec. 2.1.1) is proposed in [34], where the pixels with high prediction confidence are output without being processed by deeper layers. PointRend [150] shares a similar idea, and applies additional FC layers on selected pixels with low prediction confidence, which are more likely to be on borders of objects. All these researches demonstrate that by exploiting the spatial redundancy in image data, dynamic computation at the pixel level beyond sample level significantly increases the model efficiency.

3.1.2 Pixel-wise Dynamic Parameters
In contrast to entirely skipping the convolution operation on a subset of pixels, dynamic networks can also apply data-dependent parameters on different pixels for improved representation power or adaptive reception fields.

1) Dynamic weights. Similar to the sample-wise dynamic parameter methods (Sec. 2.2), pixel-level dynamic weights are achieved by test-time adjustment [108], [109], prediction [151], [152], [153], [154] or dynamic features [21], [129], [130], [135]. Take weight prediction as an example, typical approaches generate an $H \times W \times k^2$ kernel map to produce spatially dynamic weights ($H, W$ are the spatial size of the
output feature and $k$ is the kernel size). Considering the pixels belonging to the same object may share identical weights, dynamic region-aware convolution (DRCConv) [153] generates a segmentation mask for an input image, dividing it into $m$ regions, for each of which a weight generation network is responsible for producing a data-dependent kernel.

2) **Dynamic reception fields.** Traditional convolution operations usually have a fixed shape and size of kernels (e.g., the commonly used $3\times3$ 2D convolution). The resulting uniform receptive field across all the layers may have limitations for recognizing objects with varying shapes and sizes. To tackle this, a line of work learns to adapt the receptive field for different feature pixels [110], [111], [112], as discussed in Sec. 2.2.1. Instead of adapting the sampling location of features or kernels, adaptive connected network [156] realizes a dynamic trade-off among self transformation (e.g., $1\times1$ convolution), local inference (e.g., $3\times3$ convolution) and global inference (e.g., FC layer). The three branches of outputs are fused with data-dependent weighted summation. Besides images, the local and global information in non-Euclidean data, such as graphs, could also be adaptively aggregated.

### 3.2 Region-level Dynamic Networks

Pixel-level dynamic networks mentioned in Sec. 3.1 often require specific implementations for sparse computation, and consequently may face challenges in terms of achieving real acceleration on hardware [148]. An alternative approach is performing adaptive inference on regions/patches of input images. There mainly exists two lines of work along this direction (see Fig. 5): one performs parameterized transformations on a region of feature maps for more accurate prediction (Sec. 3.2.1), and the other learns patch-level hard attention, with the goal of improving the effectiveness and/or efficiency of models (Sec. 3.2.2).

#### 3.2.1 Dynamic Transformations

Dynamic transformations (e.g., affine/projective/thin plate spline transformation) can be performed on images under certain variations [157] for better generalization ability, or to exaggerate the salient regions [158] for discriminative feature representation. For example, spatial transformer [157] adopts a localization network to generate the transformation parameters, and then applies the parameterized transformation to recover the input from the corresponding variations. Moreover, transformations are learned to adaptively zoom-in the salient regions on some tasks where the model performance is sensitive to a small portion of regions.

#### 3.2.2 Hard Attention on Selected Patches

Inspired by the fact that informative features may only be contained in certain regions of an image, dynamic networks with hard spatial attention are explored to strategically select patches from the input for improved efficiency.

1) **Hard attention with RNNs.** The most typical approach is formulating a classification task as a sequential decision process, and adopting RNNs to make iterative predictions based on selected patches [159], [160]. For example, images are classified within a fixed number of steps, and at each step, the classifier RNN only sees a cropped patch, deciding the next attentional location until the last step is reached [159]. An adaptive step number is further achieved by including early stopping in the action space [160]. Glance-and-focus network (GFNet) [39] builds a general framework of region-level adaptive inference by sequentially focusing on a series of selected patches, and is compatible with most existing CNN architectures. The recurrent attention mechanism together with the early exiting paradigm enables both spatially and temporallyadaptive inference [39], [160].

2) **Hard attention with other implementations.** Rather than using an RNN to predict the region position that the model should pay attention to, class activation mapping (CAM) [142] is leveraged in [161] to iteratively focus on salient patches. At each iteration, the selection is performed on the previously cropped input, leading to a progressive refinement procedure. A multi-scale CNN is built in [162], where the sub-network in each scale takes in the cropped patch from the previous scale, and is responsible for simultaneously producing 1) the feature representations for classification and 2) the attention map for the next scale. Without an iterative manner, the recent differentiable patch selection [163] adopts a differentiable top-K module to select a fixed number of patches in one step.

### 3.3 Resolution-level Dynamic Networks

The researches discussed above typically divide feature maps into different areas (pixel-level or region-level) for adaptive inference. On a coarser granularity, some dynamic networks could treat each image as a whole by processing feature representations with adaptive resolutions. Although it has been observed that a low resolution might be sufficient for recognizing most "easy" samples [27], conventional CNNs mostly process all the inputs with the same resolution, inducing considerable redundancy. Therefore, resolution-level dynamic networks exploit spatial redundancy from the perspective of feature resolution rather than the saliency of different locations. Existing approaches mainly include 1) scaling the inputs with adaptive ratios (Sec. 3.3.1) and 2) selectively activating the sub-networks with different resolutions in a multi-scale architecture (Sec. 3.3.2).

#### 3.3.1 Adaptive Scaling Ratios

Dynamic resolution can be achieved by scaling features with adaptive ratios. For example, a small sub-network is first executed to predict a scale distribution of faces on the face detection task, then the input images are adaptively zoomed, so that all the faces fall in a suitable range for recognition [164]. A plug-in module is used by [163] to predict the stride for the first convolution block in each ResNet stage, producing features with dynamic resolution.

#### 3.3.2 Dynamic Resolution in Multi-scale Architectures

An alternative approach to achieving dynamic resolution is building multiple sub-networks in a parallel or cascading way. These sub-networks with different feature resolutions are selectively activated conditioned on
the input during inference. For instance, Elastic [166] realizes a soft selection from multiple branches at every layer, where each branch performs a downsample-convolution-upsample procedure with an independent scaling ratio. To practically avoid redundant computation, a hard selection is realized by [32], which allows each sample to conditionally activate sub-networks that process feature representations with resolution from low to high (see Fig. 3(c) in Sec. 2.1.1).

4 Temporal-wise Dynamic Networks

Apart from the spatial dimension (Sec. 3), adaptive computation could also be performed along the temporal dimension of sequential data, such as texts (Sec. 4.1) and videos (Sec. 4.2). In general, network efficiency can be improved by dynamically allocating less/no computation to the inputs at unimportant temporal locations.

4.1 RNN-based Dynamic Text Processing

Traditional RNNs mostly follow a static inference paradigm, i.e. input tokens are read sequentially to update a hidden state at each time step, which could be written as

$$h_t = \mathcal{F}(x_t, h_{t-1}), \quad t = 1, 2, \cdots, T. \quad (11)$$

Such a static inference paradigm induces significant redundant computation, as different tokens usually have different contributions to the downstream tasks. A type of dynamic RNN is developed for allocating appropriate computational cost at each step. Some learn to "skim" unimportant tokens by dynamic update of hidden states (Sec. 4.1.1), and others conduct adaptive reading to avoid processing task-irrelevant tokens. Specifically, such adaptive reading can be achieved by early exiting (Sec. 4.1.2) or dynamic jumping (Sec. 4.1.3).

4.1.1 Dynamic Update of Hidden States

Since not all the tokens are essential for capturing the task-relevant information in a sequence, dynamic RNNs can be built to adaptively update their hidden states at each time step. Less informative tokens will be coarsely skinned, i.e. the states are updated with cheaper computation.

1) Skipping the update. For unimportant inputs at certain temporal locations, dynamic models can learn to entirely skip the update of hidden states (see Fig. 9(a)), i.e.

$$h_t = \alpha_t \mathcal{F}(x_t, h_{t-1}) + (1 - \alpha_t) h_{t-1}, \quad \alpha_t \in \{0, 1\}. \quad (12)$$

For instance, Skip-RNN [167] updates a controlling signal in every step to determine whether to update or copy the hidden state from the previous step. An extra agent is adopted by Structural-Jump-LSTM [168] to make the skipping decision conditioned on the previous state and the current input. Without training the RNNs and the controllers jointly as in [167] and [168], a predictor is trained in [169] to estimate whether each input will make a "significant change" on the hidden state. The update is identified worthy to be executed only when the predicted change is greater than a threshold.

2) Coarse update. As directly skipping the update may be too aggressive, dynamic models could also update the hidden states with adaptively allocated operations. In specific, a network can adapt its architecture in every step, i.e.

$$h_t = \mathcal{F}_t(x_t, h_{t-1}), \quad t = 1, 2, \cdots, T, \quad (13)$$

where $\mathcal{F}_t$ is determined based on the input $x_t$. One implementation is selecting a subset of dimensions of the hidden state to calculate, and copying the remaining from the previous step [170], [171], as shown in Fig. 9(b). To achieve the partial update, a subset of rows in weight matrices of the RNN is dynamically activated in [170], while Skim-RNN [171] makes a choice between two independent RNNs. When the hidden states are generated by a multi-layer network, the update could be interrupted at an intermediate layer based on an accumulated halting score [11].

To summarize, a coarse update can be realized by data-dependent network depth [11] or width [170], [171].

3) Selective updates in hierarchical RNNs: Considering the intrinsic hierarchical structure of texts (e.g. sentence-word-character), researchers have developed hierarchical RNNs to encode the temporal dependencies with different timescales using a dynamic update mechanism [172], [173]. During inference, the RNNs at higher levels will selectively update their states conditioned on the output of low-level ones (see Fig. 9(c)). For example, when a character-level model in [172] detects that the input satisfies certain conditions, it will "flush" (reset) its states and feed them to a word-level network. Similar operations have also been realized by a gating module on question answering tasks [173].

4.1.2 Temporally Early Exiting in RNNs

Despite that the dynamic RNNs in Sec. 4.1.1 are able to update their states with data-dependent computational costs at each step, all the tokens still must be read, leading to inefficiency in scenarios where the task-relevant results can be obtained before reading the entire sequence.

Ideally, an efficient model should adaptively stop reading before the last step $T$ in Eq. (11) is reached, once the captured information is satisfactory to solve the task. For instance, reasoning network (ReasonNet) [63] terminates its reading procedure when sufficient evidence has been found for question answering. Similarly, early stopping is implemented for sentence-level [174] and paragraph-level [65] text classification, respectively. Note that the approaches discussed here focus on making early predictions with respect to the temporal dimension of sequential input, rather than along the depth dimension of networks as in Sec. 2.1.1.

4.1.3 Jumping in Texts

Although early exiting in Sec. 4.1.2 can largely reduce redundant computation, all the tokens must still be fed to the model one by one. More aggressively, dynamic RNNs could further learn to decide "where to read" by strategically skipping some tokens without reading them, and directly jumping to an arbitrary temporal location (see Fig. 9(d)).

Such dynamic jumping, together with early exiting, is realized in [175] and [64]. Specifically, LSTM-Jump [175] implements an auxiliary unit to predict the jumping stride within a defined range, and the reading process ends when the unit outputs zero. The model in [64] first decides whether to stop at each step. If not, it will further choose to re-read the current input, or to skip a flexible number of words. Moreover, structural information is exploited by Structural-Jump-LSTM [168], which utilizes an agent to decide whether to jump to the next punctuation. Apart from looking ahead, LSTM-Shuttle [176] also allows backward jumping to supplement the missed history information.

4.2 Temporal-wise Dynamic Video Recognition

For video recognition, where a video could be seen as a sequential input of frames, temporal-wise dynamic networks
are designed to allocate adaptive computational resources for different frames. This can generally be achieved by two approaches: 1) dynamically updating the hidden states in each time step of recurrent models (Sec. 4.2.1), and 2) performing adaptive pre-sampling for key frames (Sec. 4.2.2).

### 4.2.1 Video Recognition with Dynamic RNNs

Video recognition is often conducted via a recurrent procedure, where the video frames are first encoded by a 2D CNN, and the obtained frame features are fed to an RNN sequentially for updating its hidden state. Similar to the approaches introduced in Sec. 4.1, RNN-based adaptive video recognition is typically realized by 1) treating unimportant frames with relatively cheap computation (“glimpse”) [177], 2) early exiting [61], [62]; and 3) performing dynamic jumping to decide “where to see” [61], [179], [180], [181].

1. **Dynamic update of hidden states.** To reduce redundant computation at each time step, LiteEval [177] makes a choice between two LSTMs with different computational costs. ActionSpotter [178] decides whether to update the hidden state according to each input frame. AdaFuse [182] selectively reuses certain feature channels from the previous step to efficiently make use of historical information. Recent work has also proposed to adaptively decide the numerical precision [183] or modalities [184], [185] when processing the sequential input frames. Such a glimpse procedure (i.e. allocating cheap operations to unimportant frames) is similar to the aforementioned text skimming [167], [168].

2. **Temporally early exiting.** Humans are able to comprehend the contents easily before watching an entire video. Such early stopping is also implemented in dynamic networks to make predictions only based on a portion of video frames [61], [62], [186]. Together with the temporal dimension, the model in [62] further achieves early exiting from the aspect of network depth as discussed in Sec. 2.1.1.

3. **Jumping in videos.** Considering encoding those unimportant frames with a CNN still requires considerable computation, a more efficient solution could be dynamically skipping some frames without watching them. Existing arts [179], [180], [187] typically learn to predict the location that the network should jump to at each time step. Furthermore, both early stopping and dynamic jumping are allowed in [61], where the jumping stride is limited in a discrete range. Adaptive frame (AdaFrame) [181] generates a continuous scalar within the range of [0, 1] as the relative location.

### 4.2.2 Dynamic Key Frame Sampling

Rather than processing video frames recurrently as in Sec. 4.2.1 another line of work first performs an adaptive pre-sampling procedure, and then makes prediction by processing the selected subset of key frames or clips.

1. **Temporal attention** is a common technique for networks to focus on salient frames. For face recognition, neural aggregate network [22] uses soft attention to adaptively aggregate frame features. To improve the inference efficiency, hard attention is realized to remove unimportant frames iteratively with RL for efficient video face verification [188].

2. **Sampling module** is also a prevalent option for dynamically selecting the key frames/clips in a video. For example, the frames are first sampled uniformly in [189], [190], and discrete decisions are made for each selected frame to go forward or backward step by step. As for clip-level sampling, SCsample [191] is designed based on a trained classifier to find the most informative clips for prediction. Moreover, dynamic sampling network (DSN) [192] segments each video into multiple sections, and a sampling module with shared weights across the sections is exploited to sample one clip from each section.

Adjusting multiple factors of deep models simultaneously has attracted researches in both static [193], [194] and dynamic networks [195], [196], [197], [198]. For example, together with temporal-wise frame sampling, spatially adaptive computation can be achieved by spatial [196]/temporal [199] resolution adaptation and patch selection [197], [200]. It would be promising to exploit the redundancy in both input data and network structure for further improving the efficiency of deep networks.

## 5 INFERENCe AND TRAINING

In previous sections, we have reviewed three different types of dynamic networks (sample-wise (Sec. 2), spatial-wise (Sec. 3) and temporal-wise (Sec. 4)). It can be observed that making data-dependent decisions at the inference stage is essential to achieve high efficiency and effectiveness. Moreover, training dynamic models is usually more challenging than optimizing static networks.

Note that since parameter adaptation (Sec. 2.2) could be conveniently achieved by differentiable operations, models with dynamic parameters [13], [20], [119] can be directly trained by stochastic gradient descent (SGD) without specific techniques. Therefore, in this section we mainly focus on discrete decision making (Sec. 5.1) and its training strategies (Sec. 5.2), which are absent in most static models.

### 5.1 Decision Making of Dynamic Networks

As described above, dynamic networks are capable of making data-dependent decisions during inference to transform their architectures, parameters, or to select salient spatial/temporal locations in the input. Here we summarize three commonly seen decision making schemes as follows.

#### 5.1.1 Confidence-based Criteria

Many dynamic networks [12], [32], [45] are able to output “easy” samples at early exits if a certain confidence-based criterion is satisfied. These methods generally require estimating the confidence of intermediate predictions, which is...
compared to a predefined threshold for decision making. In classification tasks, the confidence is usually represented by the maximum element of the SoftMax output \([12, 22]\). Alternative criteria include the entropy \([45, 58]\) and the score margin \([49]\). On NLP tasks, a model patience is proposed in \([60]\) when the predictions for one sample stay unchanged after a number of classifiers, the inference procedure stops.

In addition, the halting score in \([11, 33, 35, 36]\) could also be viewed as confidence for whether the current feature could be output to the next time step or calculation stage. Empirically, the confidence-based criteria are easy to implement, and generally require no specific training techniques. A trade-off between accuracy and efficiency is controlled by manipulating the thresholds, which are usually tuned on a validation dataset. Note that the overconfidence issue in deep models \([201, 202]\) might affect the effectiveness of such decision paradigm, when the incorrectly classified samples could obtain a high confidence at early exits.

5.1.2 Policy Networks
It is a common option to build an additional policy network learning to adapt the network topology based on different samples. Specifically, each input sample is first processed by the policy network, whose output directly determines which parts of the main network should be activated. For example, BlockDrop \([71]\) and GaterNet \([90]\) use a policy network to adaptively decide the depth and width of a backbone network. More generally, dynamic routing in a SuperNet can also be controlled by a policy network \([106]\).

One possible limitation of this scheme is that the architectures and the training process of some policy networks are developed for a specific backbone \([71, 90]\), and may not be easily adapted to different architectures.

5.1.3 Gating Functions
Gating function is a general and flexible approach to decision making in dynamic networks. It can be conveniently adopted as a plug-in module at arbitrary locations in any backbone network. During inference, each module is responsible for controlling the local inference graph of a layer or block. The gating functions take in intermediate features and efficiently produce binary-valued gate vectors to decide: 1) which channels need to be activated \([15, 85, 86, 87, 88]\), 2) which layers need to be skipped \([47, 48, 92, 93]\), 3) which paths should be selected in a SuperNet \([107]\), or 4) what locations of the input should be allocated computations \([146, 147, 148, 182]\).

Compared to the aforementioned decision policies, the gating functions demonstrate notable generality and applicability. However, due to their lack of differentiability, these gating functions usually need specific training techniques, which will be introduced in the following Section 5.2.

5.2 Training of Dynamic Networks
Besides architecture design, training is also essential for dynamic networks. Here we summarize the existing training strategies for dynamic models from the perspectives of objectives and optimization.

5.2.1 Training Objectives for Efficient Inference
1) Training multi-exit networks. We first notice that early-exiting dynamic networks \([12, 32]\) are generally trained by minimizing a weighted cumulative loss of intermediate classifiers. One challenge for training such models is the joint optimization of multiple classifiers, which may interfere with each other. MSDNet \([12]\) alleviates the problem through its special architecture design. Several improved training techniques \([58]\) are proposed for multi-exit networks, including a gradient equilibrium algorithm to stabilize the training process, and a bi-directional knowledge transfer approach to boost the collaboration of classifiers. For temporal-wise early exiting, the training of the policy network in FrameExit \([86]\) is supervised by pseudo labels.

2) Encouraging sparsity. Many dynamic networks adapt their inference procedure by conditionally activating their computational units \([47, 87]\) or strategically sampling locations from the input \([148]\). Training these models without additional constraints would result in superfluous computational redundancy, as a network could tend to activate all the candidate units for minimizing the task-specific loss.

The overall objective function for restraining such redundancy are typically written as \(L = L_{\text{task}} + \gamma L_{\text{sparsity}}\), where \(\gamma\) is the hyper-parameter balancing the two items for the trade-off between accuracy and efficiency. In real-world applications, the second item can be designed based on the gate/mask values of candidate units (e.g. channels \([86, 87]\), layers \([47, 48]\) or spatial locations \([148]\)). Specifically, one may set a target activation rate \([48, 86]\) or minimizing the \(L_1\) norm of the gates/masks \([148]\). It is also practical to directly optimize a resource-aware loss (e.g. FLOPs) \([92, 107, 147]\), which can be estimated according to the input and output feature dimension for every candidate unit.

3) Others. Note that extra loss items are mostly designed for but not limited to improving efficiency. Take \([162]\) as an example, the model progressively focuses on a selected region, and is trained with an additional inter-scale pairwise ranking loss for proposing more discriminative regions. Moreover, knowledge distilling is utilized to boost the co-training of multiple sub-networks in \([84]\) and \([56]\).

5.2.2 Optimization of Non-differentiable Functions
A variety of dynamic networks contain non-differentiable functions that make discrete decisions to modify their architectures or sampling spatial/temporal locations from the input. These functions can not be trained directly with back-propagation. Therefore, specific techniques are studied to enable the end-to-end training as follows.

1) Gradient estimation is proposed to approximate the gradients for those non-differentiable functions and enable back-propagation. In \([72, 172]\), straight-through estimator (STE) is exploited to heuristically copy the gradient with respect to the stochastic output directly as an estimator of the gradient with respect to the Sigmoid argument.

2) Reparameterization is also a popular technique to optimize the discrete decision functions. For instance, the gating functions controlling the network width \([86]\) or depth \([48]\) can both be trained with Gumbel SoftMax \([259, 260]\), which is also used for pixel-level dynamic convolution \([147, 148]\). An alternative technique is Improved SentHash \([261]\) adopted in \([88]\) and \([90]\) to train their hard gating modules.

Note that although these reparameterization techniques enable joint optimizing dynamic models together with gating modules in an end-to-end fashion, they usually lead to
a longer training process for the decision functions to converge into a stable situation \cite{144}. Moreover, the model performance might be sensitive to some extra hyper-parameters (e.g., temperature in Gumbel SoftMax), which might also increase the training cost for these dynamic networks.

3) **Reinforcement learning (RL)** is widely exploited for training non-differentiable decision functions. In specific, the backbones are trained by standard SGD, while the agents (either policy networks in Sec. 5.1.2 or gating functions in Sec. 5.1.3) are trained with RL to take discrete actions for dynamic inference graphs \cite{15, 47}, \cite{71} or spatial/temporal sampling strategies \cite{39}, \cite{190}.

One challenge for RL-based training is the design of reward functions, which is important to the accuracy-efficiency tradeoff of dynamic models. Commonly seen reward signals are usually constructed to minimize a penalty item of the computational cost \cite{15}, \cite{47}. Moreover, the training could be costly due to a multi-stage procedure: a pre-training process may be required for the backbone networks before the optimization of decision \cite{71} or sampling \cite{39} modules, and joint finetuning may be indispensable finally.

### 6 APPLICATION OF DYNAMIC NETWORKS

In this section, we summarize the applications of dynamic networks. Representative methods are listed in Table 3 based on the input data modality.

For image recognition, most dynamic CNNs are designed to conduct sample-wise or spatial-wise adaptive inference on classification tasks, and many inference paradigms can be generalized to other applications. Note that as mentioned in Sec. 5.2, the object recognition could be formulated as a sequential decision problem \cite{39}, \cite{160}. By allowing early exiting in these approaches, temporally adaptive inference procedure could also be enabled.

For text data, reducing its intrinsic temporal redundancy has attracted great research interests, and the inference paradigm of temporal-wise dynamic RNNs (see Sec. 4.3) is also general enough to process audios \cite{262}. Based on large language models such as Transformer \cite{6} and BERT \cite{7}, adaptive depths \cite{57}, \cite{58}, \cite{59}, \cite{60} are extensively studied to reduce redundant computation in network architectures.

For video-related tasks, the three types of dynamic inference can be implemented simultaneously \cite{160}, \cite{197}, \cite{234}. However, for the networks that do not process videos recurrently, e.g., 3D CNNs \cite{263}, \cite{264}, \cite{265}, most of them still follow a static inference scheme. Few researches have been committed to building dynamic 3D CNNs \cite{195}, which might be an interesting future research direction.

Moreover, dynamic networks (especially the attention mechanisms) have also been applied to dynamically fuse the features from different modalities in some multi-modal learning tasks, e.g. RGB-D image segmentation \cite{212} and image/video captioning \cite{130}, \cite{243}, \cite{244}, \cite{245}, \cite{246}, \cite{247}.

Finally, dynamic networks have also been exploited to tackle some fundamental problems in deep learning. For example, multi-exit models can be used to: 1) alleviate the overthinking issue while reducing the overall computation \cite{50}, \cite{266}; 2) perform long-tailed classification \cite{267} by inducing early exiting in the training stage; and 3) improve the model robustness \cite{268}. For another example, the idea of dynamic routing is implemented for: 1) reducing the training cost under a multi-task setting \cite{269} and 2) finding the optimal fine-tuning strategy for per example in transfer learning \cite{270}.

### 7 CHALLENGES AND FUTURE DIRECTIONS

Though recent years have witnessed significant progress in the research of dynamic neural networks, there still exist many open problems that are worth exploring. In this section, we summarize a few challenges together with possible future directions in this field.

#### 7.1 Theories for Dynamic Networks

Despite the success of dynamic neural networks, relatively few researches has been committed to analyze them from the theoretical perspective. In fact, theories for a deep understanding of current dynamic learning models and further improving them in principled ways are highly valuable. Notably, it has been proven that a dynamic network with an adaptive width can preserve the representation power of an unsupervised model \cite{79}. However, there are more theoretical problems that are fundamental for dynamic networks. Here we list several of them as follows.

1) **Optimal decision in dynamic networks.** An essential operation in most dynamic networks (especially those designed for improving computational efficiency) is making data-dependent decisions, e.g., determining whether a module should be evaluated or skipped. Existing solutions either...
use confidence-based criteria, or introduce policy networks and gating functions. Although being effective in practice (as mentioned in Sec. 5), they may not be optimal and lack theoretical justifications. Take early exiting as an example, the current heuristic methods [12], [32] might face the issues of overconfidence, high sensitivity for threshold setting and poor transferability. As for policy networks or gating modules, runtime decisions can be made based on a learned function. However, they often introduce extra computations, and usually require a long and unstable training procedure. Therefore, principled approaches with theoretical guarantees for decision function design in dynamic networks is a valuable research topic.

2) Generalization issues. In a dynamic model, a sub-network might be activated for a set of test samples that are not uniformly sampled from the data distribution, e.g., smaller sub-networks tend to handle “easy” samples, while larger sub-networks are used for “hard” inputs [12]. This brings a divergence between the training data distribution and that of the inference stage, and thus violates the common i.i.d. assumption in classical machine learning. Therefore, it would be interesting to develop new theories to analyze the generalization properties of dynamic networks under such distribution mismatch. Note that transfer learning also aims to address the issue of distributional shift at test time, but the samples of the target domain are assumed to be accessible in advance. In contrast, for dynamic models, the test distribution is not available until the training process is finished, when the network architecture and parameters are finalized. This poses greater challenges than analyzing the generalization issues in transfer learning.

7.2 Architecture Design for Dynamic Networks

Architecture design has been proven to be essential for deep networks. Existing researches on architectural innovations are mainly proposed for static models [4], [5], [27], while relatively few are dedicated to developing architectures specially for dynamic networks. It is expected that architectures developed specifically for dynamic networks may further improve their effectiveness and efficiency. For example, the interference among multiple classifiers in an early-exiting network could be mitigated by a carefully designed multi-scale architecture with dense connections [12].

Possible research direction include designing dynamic network structures either by hand (as in [12], [32], [35], [67]), or by leveraging the NAS techniques (as in [83], [106]). Moreover, considering the popularity of Transformers [138], recent work has proposed dynamic vision Transformers with adaptive early exiting [271] or token sparsification [272], [273]. Developing a dynamic version of this family of models could also be an interesting direction.

Note that the research on dynamic networks differs from a seemingly close topic, i.e. model compression [28], [29], [31]. One common goal of them is improving the network efficiency with minimal accuracy drop. However, model compression may focus on reducing the size of deep networks, while dynamic networks pay more attention to the computation, even at the price of slightly increasing model size [15], [47]. Moreover, model compression typically adopts pruning [25] or quantization [29] techniques to produce compact static models, which treat all the inputs in the same way. In contrast, dynamic networks perform data-dependent computation on different inputs, which can effectively reduce the intrinsic redundancy in static models.

7.3 Applicability for More Diverse Tasks

Many existing dynamic networks (e.g., most of the sample-wise adaptive networks) are designed specially for classification tasks, and cannot be applied to other vision tasks such as object detection and semantic segmentation. The difficulty arises from the fact that for these tasks there is no simple criterion to assert whether an input image is easy or hard, as it usually contains multiple objects and pixels that have different levels of difficulty. Although many efforts, e.g., spatially adaptive models [33], [39], [148] and soft attention based models [13], [20], [21] have been made to address this issue, it remains a challenging problem to develop a unified and elegant dynamic network that can serve as an off-the-shelf backbone for a variety of tasks.

7.4 Gap between Theoretical & Practical Efficiency

The current deep learning hardware and libraries are mostly optimized for static models, and they may not be friendly to dynamic networks. Therefore, we usually observe that the practical runtime of dynamic models lags behind the theoretical efficiency. For example, some spatially adaptive networks involve sparse computation, which is known to be inefficient on modern computing devices due to the memory access bottleneck [148].

A recent line of work focuses on the co-design of algorithm and hardware for accelerating deep models on platforms with more flexibility such as FPGA [274]. Many input-dependent operations, including pixel-level dynamic computation [114], [273], [276], adaptive channel pruning [277], [278] and early exiting [279], have also been tailored together with hardware for further improving their practical efficiency. It is an interesting research direction to simultaneously optimize the algorithm, hardware and deep learning libraries to harvest the theoretical efficiency gains of dynamic networks.

In addition, a data-dependent inference procedure, especially for the dynamic architectures, usually requires a model to handle input samples sequentially, which also poses a challenge for parallel computation. Although inference with batches has been enabled for early-exiting networks [271], the conflict between adaptive computational graph and parallel computation still exists for other types of dynamic architectures. This issue is mitigated in the scenario of mobile/edge computing, where the input signal by itself is sequential and the computing hardware is less powerful than high-end platforms. However, designing dynamic networks that are more compatible with existing hardware and software is still a valuable and challenging topic.

7.5 Robustness Against Adversarial Attack

Dynamic models may provide new perspectives for the research on adversarial robustness of deep neural networks. For example, recent work [268] has leveraged the multi-exit structure to improve the robustness against adversarial attacks. Moreover, traditional attacks are usually aimed at causing misclassification. For dynamic networks, it is possible to launch attacks on efficiency [250], [281]. Specifically, by
adjusting the objective function of the adversarial attack, input-adaptive models could be fooled to activate all their intermediate layers \cite{26} or yielding confusing predictions at early exits \cite{27} even for “easy” samples. It has also been observed that the commonly used adversarial training is not effective to defend such attacks. The robustness of dynamic network is an interesting yet understudied topic.

7.6 Interpretabilit

Dynamic networks inherit the black-box nature of deep learning models, and thus also invite research on interpreting their working mechanism. What is special here is that the adaptive inference paradigm, e.g., spatial/temporal adaptiveness, conforms well with that of the human visual system, and may provide new possibilities for making the model more transparent to humans. In a dynamic network, it is usually convenient to analyze which part of the model is activated for a given input or to locate which part of the input the model mostly relies on in making its prediction. It is expected that the research on dynamic network will inspire new work on the interpretability of deep learning.

Acknowledgments

This work is supported in part by the National Science and Technology Major Project of the Ministry of Science and Technology of China under Grants 2018AAA0100701, the National Natural Science Foundation of China under Grants 61906106 and 62022048, the Institute for Guo Qiang of Tsinghua University and Beijing Academy of Artificial Intelligence.

References

\cite{1} Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Ima-genet classification with deep convolutional neural networks. In NeurIPS, 2012.

\cite{2} Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.

\cite{3} Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In CVPR, 2015.

\cite{4} Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In CVPR, 2016.

\cite{5} Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In CVPR, 2017.

\cite{6} Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illian Polosukhin. Attention is all you need. In NeurIPS, 2017.

\cite{7} Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In ACL, 2019.

\cite{8} Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In NeurIPS, 2020.

\cite{9} Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In ICLR, 2017.

\cite{10} Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In ICLR, 2018.

\cite{11} Alex Graves. Adaptive computation time for recurrent neural networks. arXiv preprint arXiv:1603.08983, 2016.

\cite{12} Gao Huang, Dulu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In ICLR, 2018.

\cite{13} Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. In NeurIPS, 2019.

\cite{14} Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In NeurIPS, 2017.

\cite{15} Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In NeurIPS, 2017.

\cite{16} Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In ICLR, 2017.

\cite{17} Luca Bertinetto, João H Henriques, Jack Valmadre, Philip HS Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. In NeurIPS, 2016.

\cite{18} Xin Wang, Fisher Yu, Ruth Wang, Trevor Darrell, and Joseph E Gonzalez. TaFe-net: Task-aware feature embeddings for low shot learning. In CVPR, 2019.

\cite{19} Yingpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In CVPR, 2020.

\cite{20} Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In CVPR, 2018.

\cite{21} Sanghyun Woo, Jongchun Park, Juno-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In ECCV, 2018.

\cite{22} Jialong Yang, Peiran Ren, Dongqing Zhang, Dong Chen, Fang Wen, Hongdong Li, and Gang Hu. Neural aggregation network for video face recognition. In CVPR, 2017.

\cite{23} Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR, 2015.

\cite{24} Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.

\cite{25} Yulin Wang, Xuran Pan, Shiji Song, Hong Zhang, Gao Huang, and Cheng Wu. Implicit semantic data augmentation for deep networks. In NeurIPS, 2019.

\cite{26} Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In CVPR, 2019.

\cite{27} Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

\cite{28} Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In CVPR, 2018.

\cite{29} Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In NeurIPS, 2016.

\cite{30} Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In NeurIPS Workshop, 2014.

\cite{31} Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In BMVC, 2014.

\cite{32} Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution Adaptive Networks for Efficient Inference. In CVPR, 2020.

\cite{33} Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In CVPR, 2017.

\cite{34} Xianxiao Li, Ziwei Liu, Ping Luo, Chen Change Loy, and Xiaoou Tang. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In CVPR, 2017.

\cite{35} Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal Transformers. In ICLR, 2019.

\cite{36} Maha Elbayad, Jiajia Gu, Edouard Grave, and Michael Auli. Depth-Adaptive Transformer. In ICLR, 2020.

\cite{37} David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex. The journal of physiology, 1962.

\cite{38} Akira Murata, Vittorio Gallese, Giuseppe Luppino, Masakazu Kaseda, and Hideo Sakata. Selectivity for the shape, size, and orientation of objects for grasping in neurons of monkey parietal area aip. Journal of neurophysiology, 2000.

\cite{39} Yulin Wang, Kangchen Lv, Rui Huang, Shiji Song, Le Yang, and Gao Huang. Glance and focus: a dynamic approach to reducing spatial redundancy in image classification. In NeurIPS, 2020.

\cite{40} Paul Viola and Michael J. Jones. Robust real-time face detection. IJCV, 2004.
Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. Neural networks, 1997.

Eugene M Izhikevich. Simple model of spiking neurons. TNN, 2003.

Zhuang Liu, Jiang Li, Zhiquiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In ICCV, 2017.

Surat Teerapittayanon, Bradley McDaniel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In ICPP, 2016.

Pavel Molchanov and Ekin Celebi. Dynamic inference paths in residual networks. In AAAI, 2020.

Jiapeng Wang, Xiangnan Kong, and Tian Guo. Epnet: Learning to exit early. In IJCAI, 2019.

Sam Leroux, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. In ICLR Workshop, 2013.

Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In KDD, 2018.

Jingjin Yu, and Joseph E Gonzalez. Idk cascades: Fast deep learning by learning not to overthink. In AUAI, 2017.

Andrew Davis and Itamar Arel. Low-rank approximations for conditional feedforward computation in deep neural networks. In arXiv preprint arXiv:1312.4461, 2013.

David Eigen, Marc’Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. In ICLR Workshop, 2013.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 1997.

Zhihuan Yuan, Bingzhe Wu, Zheng Liang, Shiwang Zhao, Weichen Bi, and Guangyu Sun. S2DNN: Transforming static cnn models for dynamic inference via neural architecture search. In ECCV, 2020.

Weizhe Hua, Yuan Zhou, Christopher M De Sa, Zhihuan Zhang, and G Edward Suh. Channel gating neural networks. In NeurIPS, 2019.

Xitong Gao, Yiren Zhao, Lukasz Dudziak, Robert Mullins, and Chengzhong Xu. Dynamic channel pruning: Feature boosting and suppression. In ICLR, 2019.

Charles Herrmann, Richard Strong Bowen, and Ramin Zabih. Channel selection using gumbel softmax. In ECCV, 2020.

Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. Batch-encoding for learning conditional channel gated networks. In ICLR, 2020.

Jinting Chen, Zhaocheng Zhu, Cheng Li, and Yuming Zhao. Self-adaptive network pruning. In ICONIP, 2019.

Changlin Li, Guanrong Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In CVPR, 2021.

Zhourong Chen, Yang Li, Samy Bengio, and Si Si. You look twice: Gatednet for dynamic filter selection in cnns. In CVPR, 2019.

Chuanjian Liu, Yunhe Yin, Kai Han, Chunjing Xu, and Chang X. Learning instance-wise sparsity for accelerating deep models. In IJCAI, 2019.

Yue Wang, Jianghao Shen, Ting-Kuei Hu, Pengfei Xu, Tan Nguyen, Richard G. Baraniuk, Zhangyang Wang, and Yingyan Lin. Dual dynamic inference: Enabling more efficient, adaptive and controllable deep inference. In JSSTA, 2020.

Weizhe Hua, Yuan Zhou, Christopher M De Sa, Zhihuan Zhang, and G Edward Suh. Channel gating neural networks. In NeurIPS, 2019.
[254] Zhenhua Huang, Xin Xu, Honghao Zhu, and MengChu Zhou. An efficient group recommendation model with multi-attention-based neural networks. *IEEE TNNLS*, 2020.

[255] Giannis Nikolentzos, Antoine Tixier, and Michalis Vazirgiannis. Message passing attention networks for document understanding. In *AAAI*, 2020.

[256] Gihyeon Choi, Shinhyeok Oh, and Harksoo Kim. Improving document-level sentiment classification using importance of sentences. *Entropy*, 2020.

[257] Haopeng Zhang and Jiawei Zhang. Text graph transformer for document classification. In *EMNLP*, 2020.

[258] Sunok Kim, Seungryong Kim, Dongbo Min, and Kwanghoon Sohn. Laf-net: Locally adaptive fusion networks for stereo confidence estimation. In *CVPR*, 2019.

[259] Emil Julius Gumbel. Statistical theory of extreme values and some practical applications. *NBS Applied Mathematics Series*, 1954.

[260] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

[261] Łukasz Kaiser and Samy Bengio. Discrete autoencoders for sequence models. *arXiv preprint arXiv:1801.09797*, 2018.

[262] Raffaele Tavarone and Leonardo Badino. Conditional-Computation-Based Recurrent Neural Networks for Computationally Efficient Acoustic Modelling. In *Interspeech*, 2018.

[263] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 2015.

[264] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *CVPR*, 2017.

[265] Yulin Wang, Rui Huang, Shiji Song, Zeyi Huang, and Gao Huang. Not all images are worth 16x16 words: Dynamic vision transformers with adaptive sequence length, 2021.

[266] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. *arXiv preprint arXiv:2106.02034*, 2021.

[267] Bowen Pan, Yifan Jiang, Rameswar Panda, Zhangyang Wang, Rogerio Feris, and Aude Oliva. Ia-red ˆ 2: Interpretability-aware redundancy reduction for vision transformers. *arXiv preprint arXiv:2106.12620*, 2021.