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) *instance-wise* dynamic models that process each instance 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]. In recent years, we have witnessed many successful deep models such as AlexNet [1], VGG [2], GoogleNet [3], ResNet [4], DenseNet [5] and Transformers [6]. These architecture 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 strategically 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.

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], [17]. It is worth noting that the popular soft attention mechanism could also be unified in the framework of dynamic networks, as different channels [18], spatial areas [19] or temporal locations [20] 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 [21], [22] and data preprocessing [23], [24], 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 architecture innovations in lightweight models [25], or be designed via NAS approaches [9], [10]. Their efficiency could also be further improved by using acceleration methods developed for static models, such as network pruning [26], weight quantization [27], knowledge distillation [28] and low-rank approximation [29].

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], [30], object detection [31] and semantic segmentation [32]. Moreover, the techniques developed in CV tasks are proven to transfer well to language models in NLP tasks [33], [34], and vice versa.

6) Interpretability. We finally note that the research on dynamic networks may potentially 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 [35], [36]. With dynamic neural networks, it

- *The authors are with the Department of Automation, Tsinghua University, Beijing 100084, China.*
  E-mail: {hanyz18, yangle15, wanghh20, wang-yl19}@mails.tsinghua.edu.cn; {gaohuang, shijis}@tsinghua.edu.cn.
  Corresponding author: Gao Huang.

*. *Equal contribution.*

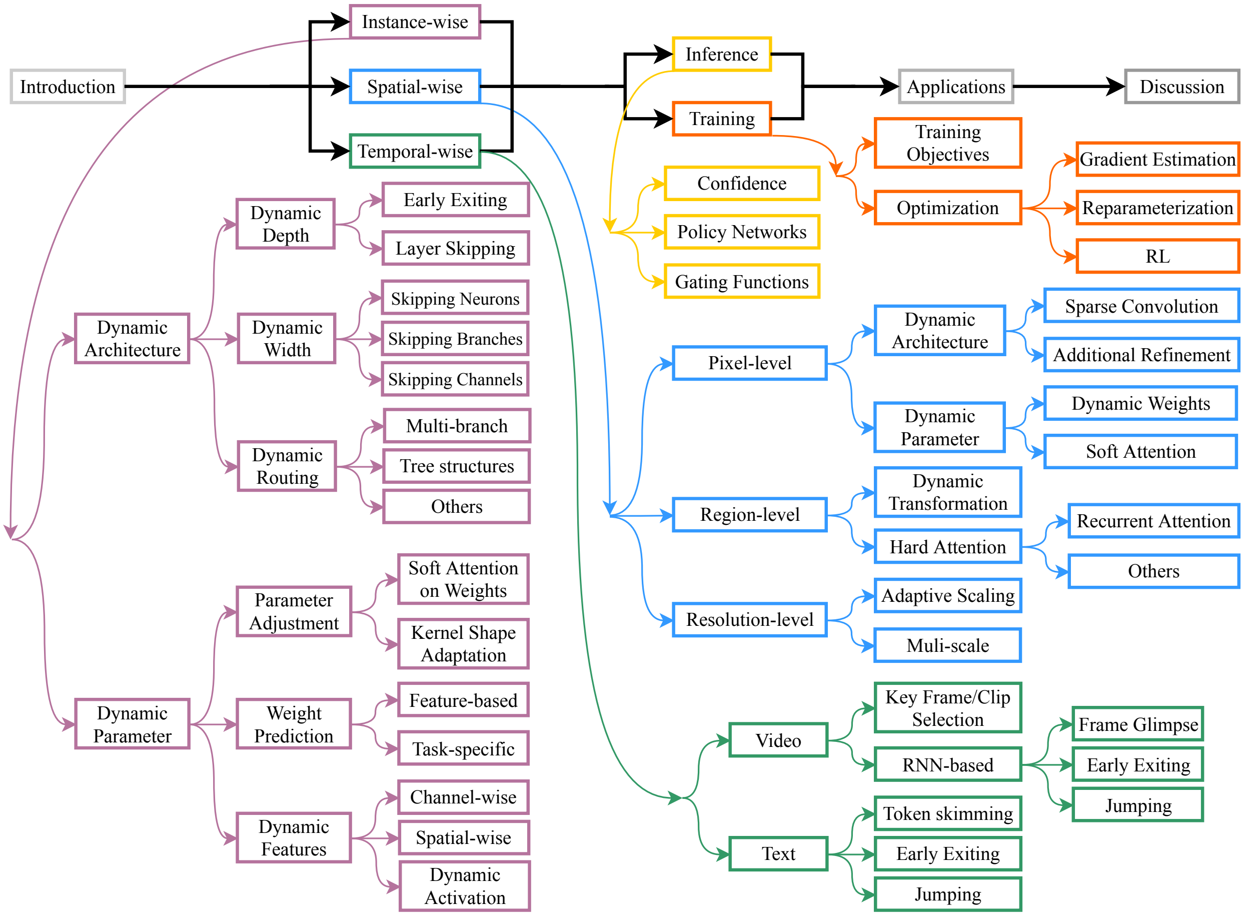


Fig. 1. Overview of the survey.

is possible to analyze which components of a deep model are activated [30] when processing an input instance, and to observe which parts of the input are accountable for certain predictions [37]. These properties may shed light on interpreting the decision process of DNNs.

In fact, adaptive inference, the key idea underlying dynamic neural networks, has been studied before the popularity of modern DNNs. The most classical approach is building an adaptive ensemble of multiple models through a cascaded [38] or parallel [39] structure, and selectively activating them conditioned on the input. The spiking neural network (SNN) [40], [41] also performs data-dependent inference by propagating pulse signals in the model. However, the training strategy for SNN is quite different from that of popular convolutional neural networks (CNNs), and it is not commonly used in vision tasks. Therefore, we leave out the work related to SNN 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.

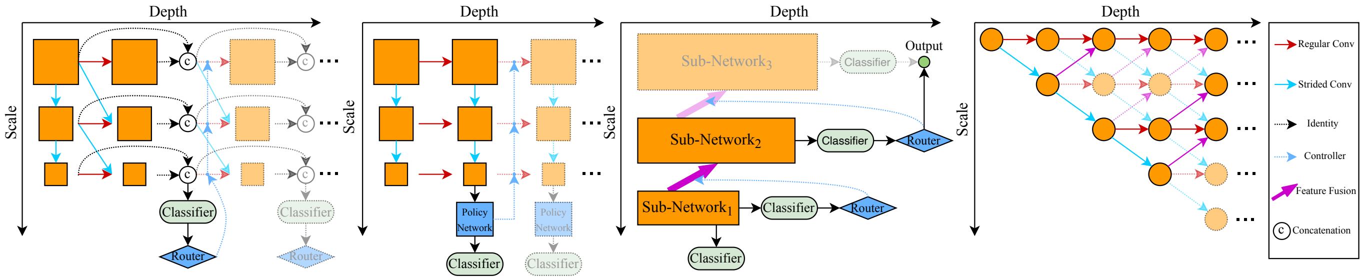
TABLE 1
Notations used in this paper.

Notations	Descriptions
$\mathbb{R}^m$	m-dimensional real number domain
$a, \mathbf{a}$	Scalar, vector/matrix/tensor
$\mathbf{x}, \mathbf{y}$	Input, output feature
$\mathbf{x}^\ell$	Feature at layer ℓ
$\mathbf{h}_t$	Hidden state at time step t
$\mathbf{x}(\mathbf{p})$	Feature at spatial location $\mathbf{p}$ on $\mathbf{x}$
$\boldsymbol{\Theta}$	Learnable parameter
$\hat{\boldsymbol{\Theta}}\|\mathbf{x}$	Dynamic parameter conditioned on $\mathbf{x}$
$\mathbf{x} \star \mathbf{W}$	Convolution of feature $\mathbf{x}$ and weight $\mathbf{W}$
$\otimes$	Channel-wise or element-wise multiplication
$\mathcal{F}(\cdot, \boldsymbol{\Theta})$	Functional Operation parameterized by $\boldsymbol{\Theta}$
$\mathcal{F} \circ \mathcal{G}$	Composition of function $\mathcal{F}$ and $\mathcal{G}$

This survey is organized as follows. In Sec. 2, we introduce the most common *instance-wise* dynamic networks which adapt their architectures or parameters conditioned on each input instance. Then, dynamic models working on a finer granularity, i.e., *spatially* adaptive or *temporally* adaptive models, are reviewed in Sec. 3 and Sec.4, respectively. Then we review the decision making strategies and the training techniques of dynamic networks in Sec. 5. We further summarize the applications of dynamic models in Sec. 6. Finally, a number of open problems and future research directions are discussed in Sec. 7. For better readability, we list the notations that will be used in this survey in Table 1.

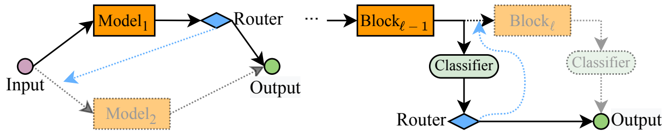
2 Instance-wise Dynamic Networks

Aiming at processing different samples in data-dependent manners, instance-wise dynamic networks are typically de-



(a) Multi-scale DenseNet. (b) Early exiting with policy networks. (c) Resolution Adaptive Network. (d) Dynamic Routing inside a SuperNet.

Fig. 2. Multi-scale architectures with dynamic inference graphs.



(a) Cascading of models. (b) Network with intermediate classifiers.

Fig. 3. The early-exiting scheme. The dashed lines and shaded modules are not executed, conditioned on the decisions made by the routers.

signed from two perspectives: 1) adjusting the model *architectures* to allocate appropriate computation based on each instance, and therefore reducing the redundant computation on those "easy" samples to improve the inference efficiency (Sec. 2.1); 2) adapting the network *parameters* to every instance while keeping the computational graphs fixed, with the goal of boosting the representation power with minimal increase of computational cost (Sec. 2.2).

2.1 Dynamic Architectures

Given that 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") instances, 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 [26], [27], [42], 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 network depths, which can be realized by 1) *early exiting*, i.e. allowing "easy" samples to be output at shallow exits without executing deeper layers [12], [43], [44]; or 2) *layer skipping*, i.e. selectively skipping intermediate network layers conditioned on each instance [11], [45], [46]. Because of the layer-wise sequential execution procedure of deep networks, models with dynamic depths usually enjoy favorable runtime efficiency in practice.

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

For an input sample $\mathbf{x}$, the forward propagation of an L-layer deep network $\mathcal{F}$ could be represented by

$$\mathbf{y} = \mathcal{F}^L \circ \mathcal{F}^{L-1} \circ \cdots \circ \mathcal{F}^1(\mathbf{x}), \tag{1}$$

where $\mathcal{F}^\ell$ 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 $\mathbf{x}_i$, the forward propagation can be written as

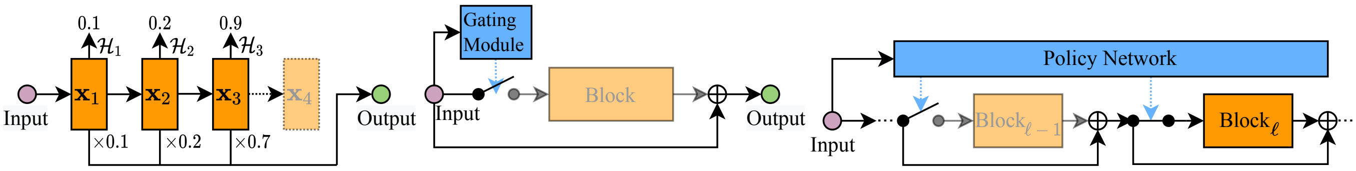
$$\mathbf{y}_i = \mathcal{F}^{\ell_i} \circ \mathcal{F}^{\ell_i - 1} \circ \cdots \circ \mathcal{F}^1(\mathbf{x}_i), 1 \leq \ell_i \leq L. \tag{2}$$

Note that ℓ_i is adaptively determined based on $\mathbf{x}_i$. 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. 3 (a)), and adaptively retrieving the prediction of an early network without activating latter ones. For example, Big/little-Net [47] 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 [44]. 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 classifiers.

b) *Intermediate classifiers.* The models in the aforementioned cascading structures are mutually independent. Consequently, once a "difficult" instance is decided to be fed to a latter network, a whole inference procedure needs to be executed from scratch, and thus the already learned features are not efficiently reused. A more compact design is to involve intermediate classifiers within one backbone network (see Fig. 3 (b)), where early features can be propagated to deep layers if needed. Based on this such architecture design, early exiting can be achieved according to confidence-based criteria [43], [48] or learned decision functions [44], [49], [50], [51]. Note that the confidence-based exiting policy consumes no extra computation during inference, while usually requiring tuning the threshold(s) on the validation set. In the second learned scheme, gating functions that directly make discrete decisions might face some training issues, which will be further discussed in Sec. 5.

Adaptive early-exiting can also be extended to language models (e.g. BERT [7]) for improving their efficiency on NLP tasks [52], [53], [54], [55]. It has also been implemented in recurrent neural networks (RNNs) for *temporally* dynamic inference when processing sequential data such as videos [56], [57] and texts [58], [59], [60] (Sec. 4).



(a) Layer skipping based on halting score. (b) Layer skipping based on a gating function. (c) Layer skipping based on a policy network.

Fig. 4. Dynamic layer skipping. The dashed features in (a) are not calculated conditioned on the halting score, and the gating module in (b) decides whether to execute the layer/block. The extra policy network in (c) directly generates the skipping decisions for all layers in the main network.

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 global 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, leading to degraded performance for deep exits. To address this issue, multi-scale dense network (MSDNet) [12] adopts 1) a *multi-scale* architecture, 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. 2 (a)). Such a specially-designed architecture effectively enhances the overall accuracy of all the classifiers in the network.

Besides the architecture design, the exiting policies and training techniques are also important for the model performance. Apart from the confidence-based criteria in [12], policy networks are built for the multi-scale dynamic models with early classifiers (see Fig. 2 (b)) [61], [62] to make decisions on whether each instance should exit. As for training, specific techniques are studied in [63] for multi-exit networks. More discussion about the inference and training schemes for dynamic models will be reviewed in Sec. 5.

The methods discussed above mostly implement the early-exiting scheme via *depth adaptation*. From the perspective of exploiting spatial redundancy in features, resolution adaptive network (RANet, see Fig. 2 (c)) [30] further achieves *resolution adaptation* with depth adaptation simultaneously. Specifically, the network first processes each instance with low-resolution features, while high-resolution representations are utilized conditioned on the prediction confidence of early classifiers.

2) Layer skipping. In the aforementioned early-exiting paradigm, the general idea 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 instance $\mathbf{x}_i$, dynamic layer skipping could be generally written as

$$\mathbf{y}_i = (\mathbb{1}^L \circ \mathcal{F}^L) \circ (\mathbb{1}^{L-1} \circ \mathcal{F}^{L-1}) \circ \cdots \circ (\mathbb{1}^1 \circ \mathcal{F}^1)(\mathbf{x}_i), \quad (3)$$

where $\mathbb{1}^\ell$ 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 representative approaches.

a) *The halting score.* Adaptive computation time (ACT) [11] is achieved based on an RNN, where a scalar named halting score is accumulated as multiple layers are sequentially executed within a time step, and the hidden state of the RNN will be directly fed to the next step if the score exceeds a threshold. The ACT method is further extended to ResNet for vision tasks [31] by viewing residual blocks within a stage [1] as linear layers within a step of RNN (see Fig. 4 (a)). Moreover, the halting score in [31] is allowed to vary across spatial locations. Rather than skipping the execution of layers with independent parameters, iterative and adaptive mobile neural network (IamNN) [64] replaces multiple residual blocks in each ResNet stage by one block with shared weights, leading to a significant reduction of parameters. 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 [33] and [34] to achieve dynamic network depth on NLP tasks.

b) *Gating function.* Apart from comparing the calculated halting scores with certain thresholds as in aforementioned approaches, gating function is also a prevalent option for making discrete decisions due to its plug-and-play property. By generating binary values based on intermediate features, a gating function can determine the skipping/execution of a layer (block) on the fly (see Fig. 4 (b)).

Take the layer skipping in ResNet as an example, let $\mathbf{x}^\ell$ denote the input feature of the ℓ-th residual block, gating function $\mathcal{G}^\ell$ generates a binary value to determine the execution of $\mathcal{F}^\ell$. This procedure could be represented by[2]

$$\mathbf{x}^{\ell+1} = \mathcal{G}^\ell(\mathbf{x}^\ell)\mathcal{F}^\ell(\mathbf{x}^\ell) + \mathbf{x}^\ell. \quad (4)$$

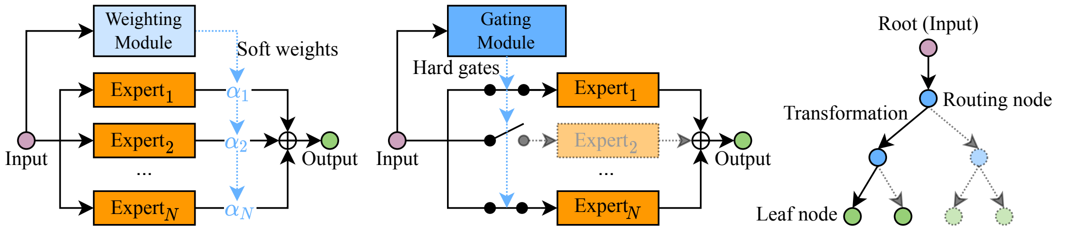
SkipNet [45] and convolutional network with adaptive inference graph (conv-AIG) [46] are two representative approaches to enabling dynamic layer skipping. Both methods induce lightweight computational overheads to efficiently produce the binary decisions on whether skipping the calculation of a residual 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 [65] iteratively executes one block with shared parameters in each residual stage. Although being seemingly similar to the aforementioned IamNN [64], its decision policies differs significantly. Without tuning the threshold for halting scores as IamNN, gating modules are exploited by [65] to decide the recursion depth.

Instead of either skipping a layer, or executing it thoroughly with a full numerical precision, a line of work [66], [67] studies adaptive *bit-width* for different layers conditioned on the *resource budget*. Furthermore, fractional skip-

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

[2] 2. For simplicity and without generality, the subscript for sample index will be omitted in the following.



(a) Soft weights for adaptive fusion. (b) Selective execution of MoE branches. (c) Dynamic routing in a tree structure.

Fig. 5. MoE structure with soft weighting (a) and hard gating (b) schemes both adopt an auxiliary module to generate the weights or gates. In the tree structure (c), nodes and transformations (paths) are represented as circles and lines with arrows respectively. Only the full lines are activated.

ping [68] adaptively selects a bit-width for each residual block by a gating function based on *input features*.

c) *Policy networks.* Besides making sequential decisions based on *intermediate features*, another implementation is using an extra model to directly decide which layers in a network need to be executed based on each *instance*. For example, BlockDrop [69] builds a policy network to take each instance as input, producing the binary gates for all layers in a pre-trained ResNet, as illustrated in Fig. 4 (c).

2.1.2 Dynamic Width

An alternative to adapting the network *depths* (Sec. 2.1.1) is performing inference with dynamic *widths*: although every layer is still executed, its multiple components (e.g. neurons, branches or channels) are selectively activated conditioned on the input. Therefore, this approach can be viewed as a finer-grained form of conditional computation.

1) Dynamic width of fully-connected (FC) layers. The computational cost of a 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 instance. Early studies learn to adaptively control the neuron activations by auxiliary branches [70], [71], [72] or other techniques such as low-rank approximation [73].

2) 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 capacity of networks without making them deeper is the MoE [39], [74] 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* MoE approaches [39], [74] adopt real-valued weights to dynamically rescale the representations obtained from different experts (shown in Fig. 5 (a)). In this way, all the branches still need to be executed, and the computation cannot be reduced at test time. To increase the inference efficiency, *hard* gates with only a fraction of non-zero elements allow the models to adaptively allocate the computation dependent on the input (see Fig. 5 (b)). Let $\mathcal{G}$ denote a gating module whose output is a N-dimensional vector $\boldsymbol{\alpha}$ controlling the execution of these N experts $\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_N$, the final output can be written as

$$\mathbf{y} = \sum\nolimits_{n=1}^{N} [\mathcal{G}(\mathbf{x})]_n \mathcal{F}_n(\mathbf{x}) = \sum\nolimits_{n=1}^{N} \alpha_n \mathcal{F}_n(\mathbf{x}), \qquad (5)$$

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 [75] replaces the convolutional blocks in the last stage of a CNN by multiple branches, and selectively execute these branches at test time. For another example, dynamic routing network (DRNet) [76] implements a hard branch selection in each cell structure commonly used in the NAS framework [10]. On NLP tasks, sparely gated MoE [16] and the recent switch Transformer [77] embeds hard MoE in a long short-term memory (LSTM) [78] network and a Transformer [6], respectively. In place of making choice with *binary* gates as in [76], only the branches corresponding to the *top-K* elements of the real-valued gates are activated in [16], [75], [77].

3) Dynamic channel pruning in CNNs. Modern CNNs usually have considerable redundancy in the large number of feature channels. Based on the common belief that the same channel can be of disparate importance for different instances, dynamic width of CNNs could be realized by adapting the channel numbers at runtime. Compared to the static pruning methods [26], [42] that remove certain filters permanently, such a dynamic pruning approach selectively skips the calculation of channels in a data-dependent manner. The capacity of a CNN is not degraded, while the overall efficiency could be improved.

a) *Multi-stage architectures along the channel dimension.* Recall that the early-exiting networks [12], [30] 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, channel gating network (CGNet) [79] is an example that uses a subset of convolutional filters in every layer, and activate the remaining filters only on certain strategically selected areas. The recent static-to-dynamic neural architecture search (S2DNAS) [80] searches for an optimal architecture among multiple structures with different widths, and any instance can be output at an early stage when a confident prediction is obtained.

b) *Dynamic pruning based on gating functions.* The aforementioned progressive activation paradigm decides the execution of a later stage based on previous output. As a result, a complete forward propagation is required to be performed for every stage, which might be suboptimal for reducing the practical inference latency. Another prevalent solution is to decide the execution of channels at every layer based on gating functions. For example, runtime neural pruning (RNP) [15] models the layer-wise pruning as a Markov decision process, and uses an RNN to select specific channel groups. Moreover, pooling operations followed by FC layers are utilized to generate channel-wise hard attention for each instance [81], [82], [83], [84]. Different reparameterization and optimizing techniques are adopted at the training stage, which will be discussed in Sec. 5.2.

The approaches mentioned above have managed to skip the execution of either network *layers* [45], [46] (see Sec. 2.1.1) or convolutional *filters* [15], [81], [82], [83]. On basis of these existing literature, recent work [85], [86], [87] 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.

It is worth noting that rather than placing plug-in gating modules inside a CNN, GaterNet [88] builds an individual network with the same architecture as the backbone. This additional network takes in the input instance and directly generating all the channel selection decisions for the backbone CNN. This implementation is similar to BlockDrop [69] discussed in Sec. 2.1.1 that exploits an extra policy network for dynamic layer skipping.

c) *Dynamic pruning based on feature activations.* Without auxiliary branches and computational overheads, dynamic pruning can be conducted directly based on *feature activation* values [89], and a regularization item is induced in training to encourage the sparsity of intermediate features.

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 [45], [46] or channels [15], [83]) conditioned on the input. The computational graph can be adapted by performing dynamic routing inside a SuperNet with various possible inference paths.

To achieve dynamic routing, there are typically routing nodes in a SuperNet that are responsible for allocating the features/samples to different paths. For node s at the ℓ-th layer, let $\alpha_{s\rightarrow j}^{\ell}$ denote the probability of assigning the reached feature $\mathbf{x}_s^{\ell}$ to node j at layer $\ell+1$, the path from node s to node j will be activated only when $\alpha_{s\rightarrow j}^{\ell}>0$. The resulting feature that reaches node j could be obtained by

$$\mathbf{x}_j^{\ell+1}=\sum\nolimits_{s\in\left\{s:\alpha_{s\rightarrow j}^{\ell}>0\right\}}\alpha_{s\rightarrow j}^{\ell}\mathcal{F}_{s\rightarrow j}^{\ell}(\mathbf{x}_s^{\ell}). \tag{6}$$

One can generate the probability $\alpha_{s\rightarrow j}^{\ell}$ in different manners, and extra constraints could be imposed at the training stage to improve efficiency. Note that the dynamic early-exiting networks [12], [30] are a special form of SuperNets, where the routing decisions are only made at intermediate classifiers. The CapsuleNet series [14], [90] also performs dynamic routing between capsules, i.e. groups of neurons, to character the relations between (parts of) objects. Here we mainly focus on different architecture designs for the SuperNets and their routing policies.

1) Path selection in multi-branch structures. The simplest SuperNet can be established by setting a number of candidate modules at each layer, and dynamically selecting one of them to execute [91], [92]. This is equivalent to having the probability distribution $\alpha_{s\rightarrow\cdot}^{\ell}$ in Eq. 6 being one-hot, and can be viewed as a special form of hard MoE (see Fig. 5 (b)). The main difference is that only one branch is selected without any fusion operations. Various implementations have been explored. For example, the branch selection is realized with RNN-based gating functions in [91]. Different topologies of branches have also been enabled by [92].

2) Neural trees and tree-structured networks. As decision trees perform inference along one forward path that is dependent on input properties, combining tree structure with neural networks can 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* together as in MoE structures (compare Fig. 5 (b), (c)).

Early work develops *soft* decision tree (SDT) [93], [94], [95] that performs differentiable operations in both training and inference stages yet are unable to achieve conditional computation. The end-to-end training for neural trees that make *hard* decisions has been enabled with specific techniques [96], [97]. Moreover, tree-structured CNNs are developed [98], [99], [100] to endow modern CNN architectures with dynamic routing behaviors.

a) *SDT* [93], [94], [95] adopts neural units as its routing nodes (blue nodes in Fig. 5 (c)), and the output of a routing node is a real-valued portion that the inputs are assigned to its left/right sub-tree. Each leaf node of an SDT generates a probability distribution over the output space, and the final prediction is the expectation of the results from all leaf nodes. In an SDT, the probability for an instance reaching each leaf node is data-dependent, while all the paths are still executed, which limits the inference efficiency.

b) *Neural trees with deterministic routing policies* are designed to make hard routing decisions during inference, avoiding computation on those unselected paths, and therefore practically improve the efficiency. The end-to-end training of hard neural trees has been enabled in [96] and [97].

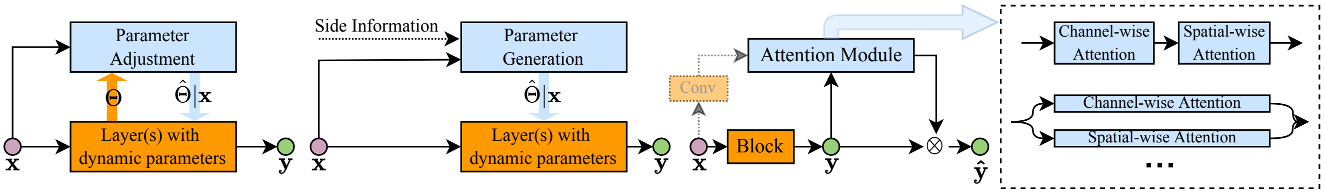
c) *Tree-structured DNNs.* Apart from developing decision trees containing neural units, a line of work builds special network architectures to endow them with the routing behavior of decision trees. For instance, hierarchical deep convolutional neural network (HD-CNN) [98] first activates a small CNN to classify each sample into coarse categories, and then conditionally executes specific sub-networks based on the coarse predictions. A subsequent work [99] not 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 [96], [97], or routing each sample to pre-designed sub-networks [98], [99], adaptive neural tree (ANT) [100] 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 other forms of SuperNet is also a recent research trend. Representatively, one can design the SuperNet by hand [101] or by NAS [102], and the routing policies for each instance are determined either by an extra network [102] or by plug-in modules [101].

For example, instance-aware neural architecture search (InstaNAS) [102] *searches* for an architecture distribution with partly shared parameters from a SuperNet containing $\sim 10^{25}$ sub-networks. During inference, every instance is allocated by a controller network to one sub-network with appropriate computational cost.

For another example, a *hand-designed* multi-scale SuperNet (see Fig. 2 (d)) is developed in [101]. Instead of training a standalone controller network with reinforcement



(a) Dynamic weight adjustment. (b) Dynamic weight prediction. (c) Soft attention for dynamic features.

Fig. 6. Adaptive inference with dynamic parameters.

learning (RL) as InstaNAS, gating modules are plugged inside the SuperNet to decide the routing path for each sample. Moreover, unlike the soft routing functions that only produce non-zero values [93], [94], [95], or many gating functions that require reparameterization techniques to produce binary values [45], [46], [83], the routing modules in [101] utilizes $\max(0, \mathrm{Tanh}(\cdot))$ as their activation function to directly generate zero values, leading to a conditional activation of different paths.

2.2 Dynamic Parameters

Although the dynamic *architectures* in Sec. 2.1 can adapt their inference graphs to each instance and achieve an efficient allocation of computation, they usually have special architecture designs, requiring specific training strategies or careful hyper-parameters tuning (Sec. 7).

Another line of work performs adaptive inference with dynamic *parameters*, while keeping the architectures fixed. Existing arts have been shown effective in improving the representation power of networks with a minor increase of computational cost. Given an input sample $\mathbf{x}$, the output of a conventional network (module) with static parameters can be written as $\mathbf{y} = \mathcal{F}(\mathbf{x}, \mathbf{\Theta})$. In contrast, the output of a model with dynamic parameters is

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \hat{\mathbf{\Theta}}|\mathbf{x}) = \mathcal{F}(\mathbf{x}, \mathcal{W}(\mathbf{x}, \mathbf{\Theta})), \tag{7}$$

where $\mathcal{W}(\cdot, \mathbf{\Theta})$ is the operation for producing the dynamic parameters, and different choices of $\mathcal{W}$ have been extensively explored.

In general, the parameter adaptation can be achieved from three aspects: 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. This implementation usually consumes little computation to obtain the adjustments, e.g., attention weights [13], [17], [103], [104] or sampling offsets [105], [106], [107] (see Fig. 6 (a)).

1) Attention on weights. The amount of trainable parameters is a key factor to the representation power. A type of dynamic networks, e.g. conditionally parameterized convolution (CondConv) [13] and dynamic convolutional neural network (DY-CNN) [17], perform soft attention on multiple convolutional kernels to produce an adaptive ensemble of parameters without noticeably increasing the computational cost. Assuming that there are N kernels $\mathbf{W}_n, n = 1, 2, \cdots, N$, such a dynamic convolution can be formulated as

$$\mathbf{y} = \mathbf{x} \star \tilde{\mathbf{W}} = \mathbf{x} \star (\sum\nolimits_{n=1}^{N} \alpha_n \mathbf{W}_n). \tag{8}$$

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 $\tilde{\mathbf{W}}$. However, the latter approach only consumes approximate $1/N$ times of computation.

Weight adjustment could also be achieved by performing soft attention over the *spatial locations* of convolutional weights [103], [104]. For example, segmentation-aware convolutional network [103] applies locally masked convolution to aggregate information with larger weights from similar pixels, which are more likely to belong to the same object. Unlike [103] that requires a sub-network for feature embedding, pixel-adaptive convolution (PAC) [104] adapts the convolutional weights based on the attention mask generated from the input feature at each layer.

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, when performing convolution on each pixel, deformable convolutions [105], [106] sample pixels from adaptive locations in the feature maps. Deformable kernels [107] samples the weights in the kernel space to adapt the *effective* reception field (ERF) while leaving the reception field unchanged. Table 2 summarizes the formulations of these three methods. Note that the main difference between [105] and [106] is that the latter version introduces a dynamic modulation mechanism by learning a spatial mask. Though customized CUDA kernels are required for implementation, these kernel shape adaptation approaches all lead to significant improvements in accuracy on image classification and object detection tasks.

2.2.2 Weight prediction

Compared to making modifications on model parameters on the fly (Sec. 2.2.1), weight prediction [108] is more straightforward: it directly generates (a subset of) instance-wise parameters with an independent model at test time (see Fig. 6 (b)). This idea was first suggested in [109], 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) [110] and HyperNetworks [111] are two classic approaches realizing runtime weight prediction for CNNs and RNNs, respectively. Specifically, a filter generation network is built in DFN [110] 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 one at each time step conditioned on the input (e.g. a word) [111]. The recent WeightNet [112] unifies the dynamic schemes of CondConv [13] and squeeze-and-excitation network (SENet) [18], and directly predicts the convolutional

TABLE 2
Deformation for convolutional kernels.

Method	Formulation	Sampled Target	Dynamic Mask
Regular Convolution	$\mathbf{y}(\mathbf{p}) = \sum_{k=1}^{K} \mathbf{W}(\mathbf{p}_k)\mathbf{x}(\mathbf{p}+\mathbf{p}_k)$	-	-
Deformable ConvNet-v1 [105]	$\mathbf{y}(\mathbf{p}) = \sum_{k=1}^{K} \mathbf{W}(\mathbf{p}_k)\mathbf{x}(\mathbf{p}+\mathbf{p}_k+\Delta\mathbf{p}_k)$	Feature map	No
Deformable ConvNet-v2 [106]	$\mathbf{y}(\mathbf{p}) = \sum_{k=1}^{K} \mathbf{W}(\mathbf{p}_k)\mathbf{x}(\mathbf{p}+\mathbf{p}_k+\Delta\mathbf{p}_k)\Delta\mathbf{m}_k$	Feature map	Yes
Deformable Kernels [107]	$\mathbf{y}(\mathbf{p}) = \sum_{k=1}^{K} \mathbf{W}(\mathbf{p}_k+\Delta\mathbf{p}_k)\mathbf{x}(\mathbf{p}+\mathbf{p}_k)$	Conv kernel	No

weights via adding a grouped FC layer after the attention activation layer, achieving competitive results in terms of the accuracy-FLOPs[3] and accuracy-parameters trade-offs.

2) Task-specific information has also been exploited to predict model parameters on the fly. For example, edge attributes are utilized in [113] to generate filters for graph convolution, and camera perspective is incorporated in [114] to generate weights for image convolution.

2.2.3 Dynamic Features

The main effect of performing inference with *adjusted* (Sec. 2.2.1) or *predicted* (Sec. 2.2.2) parameters is producing more dynamic and informative features, and therefore enhancing the representation power of deep models. A more straightforward solution is rescaling the features with input-dependent soft attention (see Fig. 6 (c)). Such dynamic features are easier to obtain, as minor modifications on computational graphs are required. Note that for a linear transformation $\mathcal{F}$, applying attention $\boldsymbol{\alpha}$ on the output features is equivalent to performing computation with re-weighted parameters, i.e.

$$\mathcal{F}(\mathbf{x}, \boldsymbol{\Theta}) \otimes \boldsymbol{\alpha} = \mathcal{F}(\mathbf{x}, \boldsymbol{\Theta} \otimes \boldsymbol{\alpha}). \tag{9}$$

1) Channel-wise attention. One common soft attention mechanism is dynamically rescaling different feature channels, following the form in SENet [18]:

$$\tilde{\mathbf{y}} = \mathbf{y} \otimes \boldsymbol{\alpha} = \mathbf{y} \otimes \mathcal{A}(\mathbf{y}). \tag{10}$$

In Eq. 10, $\mathbf{y} = \mathbf{x} \star \mathbf{W}$ is the output feature of a convolutional layer with C channels, and $\mathcal{A}(\cdot)$ is a parameterized function that contains pooling and linear layers, producing the attention $\boldsymbol{\alpha} \in [0,1]^C$ with relatively cheap computation. Taking the convolution into account, the procedure can also be written as $\tilde{\mathbf{y}} = (\mathbf{x} \star \mathbf{W}) \otimes \boldsymbol{\alpha} = \mathbf{x} \star (\mathbf{W} \otimes \boldsymbol{\alpha})$, from which we can see 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 [115], or replacing FC layers with more efficient 1D convolutions [116]. The empirical performance of three computational graphs for soft attention is studied in [117]: 1) $\tilde{\mathbf{y}} = \mathbf{y} \otimes \mathcal{A}(\mathbf{y})$, 2) $\tilde{\mathbf{y}} = \mathbf{y} \otimes \mathcal{A}(\mathbf{x})$ and 3) $\tilde{\mathbf{y}} = \mathbf{y} \otimes \mathcal{A}(\mathrm{Conv}(\mathbf{x}))$. 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 [118]. 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 [19], [119], [120], [121] (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 [122], [123]. For instance, DY-ReLU [122] replaces ReLU $(\mathbf{y}_c = \max(\mathbf{x}_c, 0))$ with the max value among N linear transformations $\mathbf{y}_c = \max_n \{a_c^n \mathbf{x}_c + b_c^n\}$, where c is the channel index, and a_c^n, b_c^n are linear coefficients calculated from $\mathbf{x}$. The dynamic activation functions are compatible with different network architectures, and have been shown effective in vision tasks.

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. For example, by replacing the weighting scalar α_n in Eq. 5 with channel-wise [124] or spatial-wise [125] attention, the output of multiple branches with independent kernel sizes [124] or feature resolutions [125] in a *soft MoE* structure (see Fig. 5 (a)) are fused with more flexibility.

Note that we leave out the detailed discussion on the self attention mechanism, which is widely studied in both NLP [6], [7] and CV fields [126], [127], [128] 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 [129], [130], [131]. 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 [132], 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 [25], 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: 1) *pixel level*, where each pixel in features is treated adaptively (Sec. 3.1); 2) *region level*, where the model only attends to strategically selected regions (Sec. 3.2) and 3) *resolution level*, while each input image is processed with adaptive resolutions (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, there are two types of pixel-level

[3] Floating point operations.

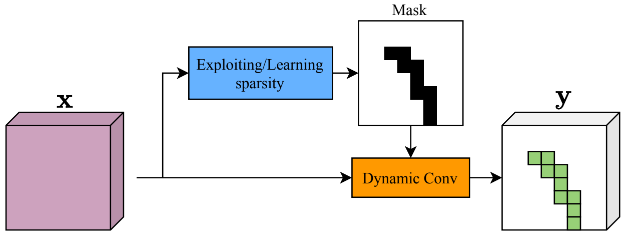


Fig. 7. Dynamic convolution on selected spatial locations.

dynamic networks: 1) models with *dynamic architectures* that adapt their depth or width when processing each pixel of features (Sec. 3.1.1); 2) networks with *dynamic parameters* that perform convolutions with pixel-specific weights for improved flexibility of feature representation (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 work generally achieves this by 1) *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. The quality of the sampled feature locations largely determines the accuracy and efficiency of the network.

Existing sampling strategies include 1) making use of the intrinsic sparsity of the input [133]; 2) predicting the positions of zero elements on the output [134], [135] and 3) estimating the saliency of pixels [136], [137], [138]. A typical implementation is adopting an extra branch to generate a spatial mask, determining the execution of convolution on each pixel (see Fig. 7). Moreover, as mentioned in Sec. 2.1.1, spatially adaptive computation time (SACT) [31] achieves dynamic network depth at each pixel based on a calculated halting score. In these dynamic convolutions, the unselected positions are usually neglected, which might degrade the network performance. The recent stochastic feature sampling and interpolation (SFSI) [138] utilizes interpolation to efficiently fill those locations, therefore alleviating the aforementioned disadvantage.

2) Dynamic additional refinement. Instead of sampling a subset of pixels to perform convolutions, another line of work first conducts relatively cheap computation on the whole feature map, and adaptively activate extra modules on certain pixels for further refinement. Representatively, dynamic capacity network [139] generates coarse features with a shallow model, and utilizes the gradient information to predict sensitive spatial locations for the network output. For these locations, extra layers are applied to extract finer features. Similarly, specific positions are additionally processed by a fraction of convolutional filters in channel gating network (CGNet) [79]. These methods adapt their network architectures in terms of *depth* or *width* at the pixel level, achieving a spatially adaptive allocation of computation.

In semantic segmentation, pixel-wise *early exiting* (see Sec. 2.1.1) is proposed in [32], where the pixels with high prediction confidence are output without being processed by deeper layers. PointRend [140] 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 instance 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 weights on different pixels for improved representation power or adaptive reception fields.

1) Dynamic weights. The trained convolutional weights could be rescaled by pixel-wise attention on the fly. For example, pixel-adaptive convolution [104] rescales the weights based on the distance between pairs of pixels. Apart from making dynamic modifications, *weight prediction* (Sec. 2.2.2) is also adopted to directly generate location-specific convolution kernels. Most existing arts [141], [142], [143], [144] 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 (DRConv) [145] 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-dependant kernel.

2) Dynamic reception fields. Traditional convolution operations usually have a fixed shape and size of kernels (e.g. the commonly used 3×3 square for 2D convolution). The resulting uniform reception field across all the layers may have limitations for recognizing objects with varying shapes and sizes. To tackle this, a line of work learns adaptive reception field for different feature positions. As we have introduced in Sec. 2.2.1, the deformable convolution series [105], [106] dynamically samples pixels from the whole feature map when performing convolutions. Moreover, an adaptive sampling can also be conducted in the kernel space rather than the feature space to achieve adaptive ERF [107].

Instead of adapting the sampling location of features or kernels, adaptive connected network [146] realizes a dynamic trade-off among self transformation (e.g. 1×1 convolution), local inference (e.g. 3×3 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) Pixel-wise dynamic feature. Applying spatial-wise soft attention on features can effectively increase the representation power of models [19], [119], [120], [125]. Though being equivalent to performing convolution with dynamic weights, directly rescaling the features is usually easier to implement in practice.

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. An alternative approach is performing adaptive inference on *regions or patches* of the

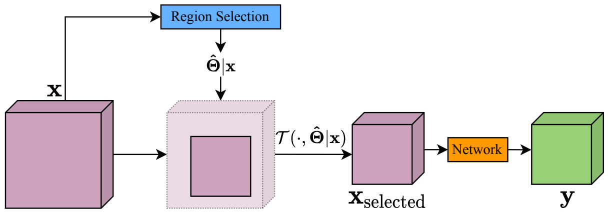

Region Selection
$\hat{\mathbf{\Theta}}|\mathbf{x}$
$\mathbf{x}$
$\mathcal{T}(\cdot, \hat{\mathbf{\Theta}}|\mathbf{x})$
Network
$\mathbf{x}_{\text{selected}}$
$\mathbf{y}$

Fig. 8. Region-level dynamic inference.

input images. There mainly exists two lines of work along this direction. One performs parameterized *transformations* on a region from input feature maps for more accurate prediction [147], [148] (Sec. 3.2.1). The other one learns *hard attention* on selected patches [37], [149], [150], with the goal of improving the effectiveness and/or efficiency of models (Sec. 3.2.2). The general procedure is illustrated in Fig. 8, where the region selection module generates the transformation parameters or the location of the attended region, and the subsequent network performs inference on the transformed/cropped input.

3.2.1 Dynamic transformations

Dynamic transformations (e.g. affine/projective/thin plate spline transformation) can be performed on images to undo certain variations [147] for better generalization ability, or to exaggerate the salient regions [148] for effective visual attention. For example, spatial transformer [147] 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 of images on tasks that the model performance is sensitive to a small portion of regions, e.g. gaze tracking and fine-grained image classification [148].

3.2.2 Hard attention on selected patches

Inspired by the fact that informative features may only correspond to certain regions of an image, dynamic networks with hard attention are proposed to strategically select patches from the input for higher efficiency. Extensive implementations have been explored as follows.

1) Hard attention with RNNs. The most typical approach to region-level hard attention is formulating a classification task as a sequential decision process. Along this direction, RNNs are adopted to focus on one patch at a time, and predictions are made iteratively [149], [151]. For example, recurrent attention model (RAM) [149] classifies images within a fixed number of steps. At each step, the classifier RNN only sees a cropped patch, deciding the next attentional location until the last step is reached. An adaptive step number is further achieved by including early stopping in the action space [151]. The recent glance-and-focus network (GFNet) [37] builds a general framework of adaptive inference by sequentially focusing on a series of selected patches, and is compatible with all existing backbone architectures. By allowing early exiting, both spatially and temporally adaptive inference can be realized [37], [151].

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) [132] is adopted to select salient patches iteratively [152]. At each iteration, the selection is performed on the previously cropped input, leading to a progressive refinement procedure. Recurrent attention CNN (RA-CNN) [150] adopts a multi-scale architecture to implement hard attention, in which each scale takes the cropped patch from the previous scale as input, and is responsible for simultaneously learning 1) the feature representations for classification and 2) the attention map for the next scale.

3.3 Resolution-level Dynamic Networks

The researches discussed above typically divide feature maps into different areas, and treat them in an adaptive manner. A downside of these approaches is that the sparse sampling (Sec. 3.1) or cropping (Sec. 3.2) operations might degrade the practical efficiency. Alternatively, dynamic networks could treat each image as a whole with representations of adaptive resolutions. It has been observed that a low resolution might be sufficient for recognizing ”easy” samples [25]. Traditional CNNs mostly process all the inputs with the same resolution, inducing considerable redundancy. Therefore, resolution-level dynamic networks exploits spatial redundancy from the perspective of feature resolution rather than the saliency of different locations. Existing arts mainly include 1) downsampling/upsampling with adaptive scaling ratios [153], [154] (Sec. 3.3.1); 2) selectively activating the sub-networks with different resolutions in a multi-scale architecture [30], [155] (Sec. 3.3.2).

3.3.1 Adaptive scaling ratios

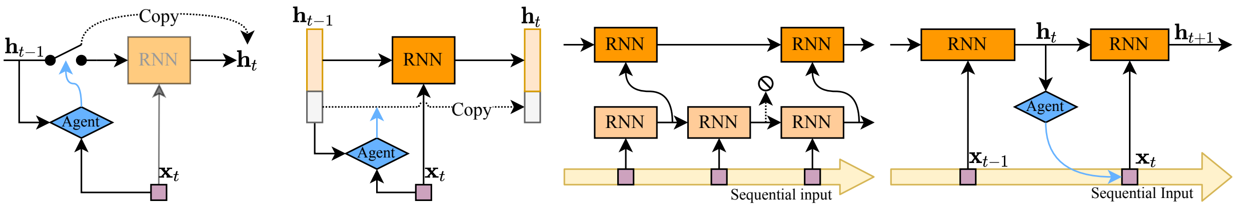
Features with dynamic resolution can be produced by performing downsample/upsample with adaptive scaling ratios. For example, a small sub-network is first executed to predict a scale distribution of faces on the face detection task [153]. Then the input images are adaptively zoomed-in or zoomed-out, so that all the faces fall in a suitable range for recognition. A subsequent work further exploits a plug-in module to predict the stride for the first convolution layer in each ResNet stage, producing features with dynamic resolution [154].

3.3.2 Dynamic resolution in multi-scale architectures

An alternative approach to achieving dynamic resolution is building multiple sub-networks in a parallel [155] or cascading [30] way. These sub-networks with different feature resolutions are selectively activated conditioned on the input during inference. For instance, Elastic [155] 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 resolution adaptive network (RANet) [30], which allows each instance to conditionally activate sub-networks that process feature representations with resolution from low to high (see also 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 particular, redundant computation can be generally saved from two aspects: 1) allocating cheap operations for the input at certain locations; 2) selectively conducting computation only on a subset of temporal locations.



(a) Skip update of hidden state. (b) Partial update of hidden state. (c) Multi-scale RNN architecture. (d) Temporal dynamic jumping.

Fig. 9. Temporally adaptive inference.

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

$$\mathbf{h}_t = \mathcal{F}(\mathbf{x}_t, \mathbf{h}_{t-1}), t = 1, 2, \cdots, T. \tag{11}$$

The final state $\mathbf{h}_T$ is utilized for solving the task. Such a static inference paradigm induces significant redundant computation, as different tokens usually have different contributions to the downstream tasks. A type of dynamic RNNs is developed for allocating appropriate computational cost at each step. Some of them read in all the tokens while learning to *"skim"* unimportant tokens by dynamic update of hidden states (Sec. 4.1.1), and others conduct an *adaptive reading* procedure to avoid reading in task-irrelevant tokens at test time. Specifically, such adaptive reading can be achieved by *early exiting* (Sec. 4.1.2) or by *jumping* in texts with adaptive strides (Sec. 4.1.3). Note that the input of these RNNs at each step is free to the level of text, which could be characters [11], words [156] or even sentences [60].

4.1.1 Dynamic Update of Hidden States

Since not all the words or sentences 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 *skimmed*, i.e. the states are updated with cheap operations to reduce unnecessary computation. After reading in a token, the dynamic update could be achieved by: 1) directly skipping the update [157], [158], [159]; 2) conducting a coarse update [11], [160], [161]and 3) performing selective update in multi-scale structures [162], [163].

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.

$$\mathbf{h}_t = \alpha_t \mathcal{F}(\mathbf{x}_t, \mathbf{h}_{t-1}) + (1 - \alpha_t)\mathbf{h}_{t-1}, \alpha_t \in \{0, 1\}\,. \tag{12}$$

For instance, Skip-RNN [157] uses a controlling signal to determine whether to update or *copy* the state, where the signal is regarded as a special hidden state and is updated in every step with negligible computation. An extra agent is adopted by Structural-Jump-LSTM [158] to make the skipping decision conditioned on the previous state and the current input. Compared to [157] and [158] that require training the RNNs and the controllers jointly, a predictor is trained in [159] 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 change brought by the update 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.

$$\mathbf{h}_t = \mathcal{F}_t(\mathbf{x}_t, \mathbf{h}_{t-1}), \tag{13}$$

where $\mathcal{F}_t$ is determined based on the input $\mathbf{x}_t$. One implementation is selecting a subset of state dimensions to calculate, and copying the remaining from the previous step [160], [161], as shown in Fig. 9 (b). To achieve partial update, a subset of rows in weight matrices of the RNN is dynamically activated in [160], while Skim-RNN [161] 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 a calculated halting score (see Fig. 4 (a)).

To summarize, a coarse update can be realized by with data-dependent network *depth* [11] or *width* [160], [161].

3) Selective updates in multi-scale RNNs. Dynamic multi-scale RNNs [162], [163] are built to capture the hierarchical structure in long sequences. 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, in hierarchical multi-scale recurrent neural network (HM-RNN) [162], when the low-level (character-level) model detects that the input satisfies certain conditions, it will *"flush"* (reset) its states and feed them to a higher-level network (word-level). Focused hierarchical RNN [163] builds a multi-scale architecture in question answering (QA) tasks. A gating module is applied to decide whether the state should be fed to the higher-level RNN (sentence-level) based on each word in the asked question.

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. For instance, one could capture the main idea of a paper by reading only its title and abstract.

Ideally, an efficient model should adaptively stop reading once the captured information is sufficient to yield a confident result, i.e. performing early exiting before the last step T in Eq. 11 is reached. For instance, reasoning network (ReasoNet) [58] terminates its reading procedure when sufficient evidence has been found for answering a question. Similarly, length adaptive recurrent model (LARM) [164] and Jumper [60] implement early stopping for sentence-level and paragraph-level classification respectively. Note that the dynamic models 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 discussed in Sec. 2.1.1.

4.1.3 Jumping in Texts

Although the early exiting mechanism in Sec. 4.1.2 largely reduces 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 certain tokens without reading them, and directly jumping to a temporal location with an adaptive stride (see Fig. 9 (d)).

Such dynamic jumping, together with early exiting, is presented in [156] and [59]. Specifically, the network in [156] implements an auxiliary unit to predict the jumping stride within a defined range, and the reading process ends when the unit outputs zero. Differently, the model in [59] first decides whether to stop at each step. If not, it will further choose to re-read the current input, or skip a flexible number of words. Moreover, structural information is exploited by Structural-Jump-LSTM [158], which utilizes an agent to decide whether to jump to the next punctuation. Apart from looking ahead, LSTM-Shuttle [165] 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 conditioned on the input. This can generally be achieved by two approaches as follows.

First (Sec. 4.2.1), a line of work performs recognition by processing frames sequentially with RNNs. Similar to the approaches introduced in Sec. 4.1, RNN-based adaptive video recognition can be realized by 1) treating unimportant frames with relatively cheap operations (a *"glimpse"*) [166], [167]; 2) *early exiting* [56], [57] and 3) strategically decide *"what to see when"* in each time step [56], [168], [169], [170].

The second line of work (Sec. 4.2.2) adopts a dynamic pre-sampling procedure for key frames (or clips, i.e. a small part of a long video) [171], [172], [173], and the selected frames or clips are processed by a task-specific model.

4.2.1 Video Recognition with Dynamic RNNs

Video recognition is often conducted via a recurrent procedure, where the frames of a video are first encoded by a 2D CNN, and the obtained frame features are fed to an RNN sequentially for updating a hidden state, which is finally used for the prediction. Such a procedure is similar to text processing mentioned in Sec. 4.1. Due to the temporal redundancy in videos, task-irrelevant frames could be processed coarsely, or even be neglected.

1) Dynamic update of hidden states. To avoid redundant computation in each step, LiteEval [166] makes a choice between two LSTM models with different computational costs. ActionSpotter [167] adaptively decides whether the current input should be used to update the hidden state. Such a *glimpse* procedure (i.e. allocating cheap operations on relatively unimportant frames) is similar to the *skimming* operation for texts [157], [158] (Sec. 4.1.1).

2) Temporally early exiting. Humans are able to comprehend the contents easily before watching an entire video. Such early stopping is also implemented to make predictions only based on a portion of video frames [56], [57]. Together with the *temporal* dimension, the model in [57] 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 inference paradigm could be achieved by dynamically skipping some frames without watching them. Existing arts [168], [169], [174] 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 [56], where the jumping stride is limited in a discrete range. Adaptive frame (AdaFrame) [170] generates a continuous scalar within the range of $[0, 1]$ as the relative location. Recent work [175] realizes the adaptation of frame resolution together with the dynamic jumping scheme, which further improves the recognition efficiency by considering the redundancy in both *spatial* and *temporal* dimensions.

4.2.2 Dynamic Key Frame/Clip Sampling

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

1) Temporal attention. Both soft and hard attention have been exploited for networks to focus on salient frames in videos. For face recognition, neural aggregation network [20] uses *soft* attention to adaptively aggregate frame features. With the goal of improving inference efficiency, *hard* attention is realized in [171] to remove unimportant frames iteratively with RL for efficient video face verification.

2) Dynamic sampling strategies. In addition to the attention mechanism, sampling module is also an alternative option. For example, one can train the frame sampling agent(s) with RL [172], [176]. Both approaches first sample frames uniformly, and then make decisions for each selected frame to go forward or backward step by step. As for clip-level sampling, salient clips sampler (SCSample) [173] is designed based on a trained classifier to find the most informative clips for prediction. Moreover, dynamic sampling network (DSN) [177] 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.

Beyond frame/clip selection, adaptive 3D convolution (Ada3D) [178] performs a choice between 2D and 3D convolution after obtaining the selected frames. By simply taking the center channel of a 3D filter along its temporal dimension, a 3D convolution could be transformed into a 2D one. By doing so, the inference efficiency is improved by exploiting the redundancy in both *data* and *network structure*, which is a recent research trend [30], [37].

5 Inference and Training

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

Note that since parameter adaptation (Sec. 2.2) could be conveniently achieved by differentiable operations, models with dynamic parameters [13], [18], [112] 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], [30], [43] 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], [30]. Alternative criteria include the entropy [43], [53] and the score margin [47]. On NLP tasks, a *model patience* is proposed in [55]: when the predictions for one instance stay unchanged after a number of classifiers, the inference procedure stops.

In addition, the halting score in [11], [31], [33], [34] 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 is usually tuned on a validation dataset. It is worth noting that the *overconfidence* issue in deep models [179], [180] might affect the effectiveness of such decision paradigm, which means that the samples that are classified incorrectly with high confidence could be output at early exits.

5.1.2 Policy Networks

To adapt the network topology based on different instances, it is a common option to build an additional policy network learning a decision function for execution of multiple units in a model. 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 [69] and GaterNet [88] use a policy network to adaptively control the *depth* (Sec. 2.1.1) and *width* (Sec. 2.1.2) of a backbone network. More generally, the dynamic routing decisions in a *SuperNet* can also be controlled by a policy network [102] (Sec. 2.1.3).

It is worth noting that the architecture design and the training process of such policy networks are typically developed for a specific backbone. This is considered as a limitation of this decision scheme, because it cannot be easily adapted to different backbone 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 in any backbone network at arbitrary locations. 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], [81], [82], [83], [84], 2) which layers need to be skipped [45], [46], [85], [86], 3) which paths should be selected in a SuperNet [101], or 4) what locations of the input should be allocated computations [136], [137], [138].

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 subsection.

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. First, we notice that dynamic networks with early exists [12], [30] 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 multi-scale architecture and dense connections. Several training techniques are proposed in [63] to further improve the training efficacy of multi-exit networks, including a gradient equilibrium algorithm to stable the training process, and a bi-directional knowledge transfer approach to boost the collaboration of classifiers.

2) Encouraging sparsity. Some dynamic networks adapt their inference procedure by conditionally activating their computational units [45], [83] or strategically sampling locations from the input [138]. 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 $\mathfrak{L} = \mathfrak{L}_{\text{task}} + \gamma\mathfrak{L}_{\text{sparse}}$. where γ 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 [82], [83], layers [45], [46] or spatial locations [138]). Specifically, one may set a target activation rate for these units [46], [82] or directly minimizing the $\mathcal{L}_1$ norm of the gates/masks as a regularization item [138]. Moreover, it is also practical to optimize a resource-aware loss (e.g. FLOPs) [85], [101], [137], which can be estimated according to the input and output feature dimension for every candidate unit.

3) Other techniques. Note that extra loss items are mostly designed for but not limited to improving efficiency. Take [150] as an example, the multi-scale model progressively focuses on a selected region, and is trained with an additional *inter-scale pairwise ranking loss*, which is designed for better region proposals with representative features. Moreover, knowledge distilling is also utilized to boost the co-training of multiple sub-networks in [79] and [63].

TABLE 3
Applications of Dynamic Networks. For the type column, I, S, T stand for instance-wise, spatial-wise and temporal-wise respectively.

Fields	Data	Type	Subfields & references
Computer Vision	Image	I	Object detection (face [38], [181], [182], facial point [183], pedestrian [184], general [31], [185], [186], [187], [188]) Image segmentation [101], [189], Super resolution [190], Style transfer [191], Coarse-to-fine classification [192]
		I & S	Image segmentation [32], [119], [136], [138], [140], [144], [146], [193], [194], [195], [196], [197], Image-to-image translation [198], Object detection [105], [106], [137], [138], [153], Semantic image synthesis [199], [200], [201], Image denoising [202], Fine-grained classification [148], [150], [203], [204] Eye tracking [148], Super resolution [141], [143], [205]
		I & S & T	General classification [37], [149], [152], Multi-object classification [206], [207], Fine-grained classification [151]
	Video	I	Multi-task learning (human action recognition and frame prediction) [208]
		I & T	Classification (action recognition) [56], [166], [170], [172], [173], [175], [176], [177], [209], Semantic segmentation [210] Video face recognition [20], [171], Action detection [168], [169], Action spotting [167], [174]
		I & S & T	Frame interpolation [211], [212], Video super resolution [213], Video deblurring [214], [215], Action prediction [216]
	Point Cloud	I & S	3D Shape classification and segmentation, 3D scene segmentation [217], 3D semantic scene completion [218]
Natural Language Processing	Text	I	Neural language inference, Text classification, Paraphrase similarity matching, and Sentiment analysis [54], [55]
		I & T	Language modeling [11], [16], [111], [160], [162], Machine translation [16], [33], [34], Classification [59], [60], [164], Sentiment analysis [156], [158], [159], [161], [165], Question answering [33], [58], [158], [161], [163]
Cross-Field	Image & Text	I & S & T	Image captioning [120], [219], visual question answering [220]
Others			Document classification [146], Link prediction [221], Graph classification [113], Stereo confidence estimation [222], Recommendation systems [223]

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 have been proposed to enable the end-to-end training, including estimating the gradients of non-differentiable variables [70], [162], or adopting reparameterization techniques [46], [88], [137], [138]. Other work also exploits reinforcement learning (RL) to train such discrete actions [15], [45], [61], [169].

1) Gradient estimation is proposed to approximate the gradients for those non-differentiable functions and enable back-propagation. In [70], [162], 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 techniques. Apart from STE, reparameterization techniques are also proposed to deal with the training problem of non-differentiable functions. For instance, the gating functions in [46], [82] are both trained with the *Gumbel SoftMax* technique [224], [225] to control the network width or depth. For reducing the spatial redundancy in CNNs, [138] and [137] also use Gumbel SoftMax to sample feature pixels for dynamic convolution. In addition, *Improved SemHash* [226] is utilized by [84] and [88] to enable end-to-end training of hard gating modules.

3) Reinforcement learning. The non-differentiable decision functions can also be trained with RL. In specific, the backbones are trained by standard SGD, while the agents (either policy networks as introduced in Sec. 5.1.2 or gating functions as discussed in Sec. 5.1.3) are trained with RL to take discrete actions for dynamic inference graphs [15], [45], [69] or spatial/temporal sampling strategies [149], [151], [168], [176]. The reward signal is usually constructed to minimize a penalty item of the computational cost for efficiency.

6 Application of Dynamic Networks

In this section, we summarize the typical applications of dynamic DNNs. Based on the input data modality, we list representative methods and their corresponding adaptive modes in Table 3.

For image recognition, most dynamic CNNs are designed to conduct *instance-wise* or *spatial-wise* adaptive inference on classification task, and many inference paradigms can be generalized to other tasks. Note that as mentioned in Sec. 3.2, the object recognition could be formulated as a sequential decision problem [37], [151]. By allowing early exiting in these approaches, *temporally* adaptive inference procedure could also be realized.

For text data, reducing the intrinsic temporal redundancy of the sequential input has attracted great research interests. The inference paradigm of *temporal-wise* dynamic RNNs (see Sec. 4.1) is also general enough to process audios (e.g. multi-scale RNN for speech recognition [227]). Based on large language models such as Transformer [6] and BERT [7], data-dependent model depths [52], [53], [54], [55] are extensively studied to reduce the structure redundancy for efficient inference.

For video-related tasks, the three types of dynamic inference (*instance-wise, spatial-wise and temporal-wise*) can be implemented simultaneously [151], [211], [212]. However, for networks that do not process videos recurrently, e.g. 3D CNNs [228], [229], [230], most of them still follow a static inference scheme, and few researches have been committed to building dynamic 3D CNNs [178], which might be an interesting future research direction.

Dynamic networks can also be exploited to tackle some fundamental problems in deep learning. For example, multi-exit models have been used to: 1) alleviate the *over-thinking* issue while reducing the overall computation [49], [231]; 2) perform *long-tailed classification* [232] by inducing early exiting in the training stage and 3) improve the model *robustness* [233]. For another example, the idea of dynamic routing is implemented for: 1) reducing the training cost under a *multi-task* setting [234] and 2) finding the optimal fine-tuning strategy for per example in *transfer learning* [235].

7 Discussions

Though significant advances have been made in the research of dynamic deep 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. Here we list several theoretical problems that are fundamental for dynamic networks.

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], [30] 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 architecture innovations are mainly proposed for static models [4], [5], [25], while relatively few are dedicated to developing architectures specially for dynamic networks. Most current approaches simply adopt structures designed for static models, which may lead to suboptimal solutions and degraded performance. For example, it is observed that intermediate classifiers tend to interfere with each other in an early-exiting network, while the problem can be solved by a carefully designed multi-scale architecture with dense connections [12].

It is expected that architectures developed specifically for dynamic networks may further improve their effectiveness and efficiency. Possible research direction include designing dynamic network structures either by hand (as in [12], [30], [33], [65]), or by leveraging the NAS techniques (as in [80], [102]). Moreover, considering the popularity of Transformers [128], developing a dynamic version of this family of models could be an interesting direction.

7.3 Applicability for More Diverse Tasks

Many existing dynamic networks (e.g., most of the instance-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 level of difficulty. Although many efforts, e.g., spatially adaptive models [31], [37], [138] and soft attention based models [13], [18], [19], have been made to address this issue, it remains to be a challenging problem to develop an 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 GPUs. In addition, dynamic inference usually requires the model to handle input samples in sequential, which also poses challenge for parallel computation. 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 GPUs. However, designing dynamic networks that are more compatible with existing hardware and software is still a valuable and challenging topic. Moreover, we note that it is also an interesting research direction to optimize the hardware and deep learning libraries to harvest the theoretical efficiency gains of dynamic networks.

7.5 Robustness Against Adversarial Attack

Dynamic models may provide new perspectives for the research of adversarial robustness on deep neural networks, as shown in the recent work [233]. In addition, traditional attacks are usually aimed at reducing the accuracy of models. For dynamic networks, it is possible to launch adversarial attacks to decrease the efficiency and accuracy simultaneously [236]. The robustness of dynamic network is an interesting yet understudied topic.

7.6 Interpretability

Dynamic networks inherit the black-box nature of deep neural networks, 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 feature the model mostly relies on in making its prediction. We expect that the research on dynamic network will inspire new work on the interpretability of deep learning.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[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.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[5] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

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

[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.

[9] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

[10] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *ICLR*, 2018.

[11] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[12] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *ICLR*, 2018.

[13] Brandon Yang, Gabriel Bender, Quoc V Le, and Jiquan Ngiam. Condconv: Conditionally parameterized convolutions for efficient inference. In *NeurIPS*, 2019.

[14] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *NeurIPs*, 2017.

[15] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *NeurIPS*, 2017.

[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.

[17] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *CVPR*, 2020.

[18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

[19] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *ECCV*, 2018.

[20] Jiaolong Yang, Peiran Ren, Dongqing Zhang, Dong Chen, Fang Wen, Hongdong Li, and Gang Hua. Neural aggregation network for video face recognition. In *CVPR*, 2017.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[23] Yulin Wang, Xuran Pan, Shiji Song, Hong Zhang, Gao Huang, and Cheng Wu. Implicit semantic data augmentation for deep networks. In *NeurIPS*, 2019.

[24] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, 2018.

[25] 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.

[26] Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *CVPR*, 2018.

[27] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NeurIPS*, 2016.

[28] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *NeurIPS Workshop*, 2014.

[29] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[30] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution Adaptive Networks for Efficient Inference. In *CVPR*, 2020.

[31] 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.

[32] Xiaoxiao 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.

[33] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal Transformers. In *ICLR*, 2019.

[34] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-Adaptive Transformer. In *ICLR*, 2020.

[35] 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.

[36] 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.

[37] 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.

[38] Paul Viola and Michael J. Jones. Robust real-time face detection. *IJCV*, 2004.

[39] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 1991.

[40] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 1997.

[41] Eugene M Izhikevich. Simple model of spiking neurons. *TNN*, 2003.

[42] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[43] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *ICPR*, 2016.

[44] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *ICML*, 2017.

[45] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *ECCV*, 2018.

[46] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *ECCV*, 2018.

[47] Eunhyeok Park, Dongyoung Kim, Soobeom Kim, Yong-Deok Kim, Gunhee Kim, Sungroh Yoon, and Sungjoo Yoo. Big/little deep neural network for ultra low power inference. In *CODES+ISSS*, 2015.

[48] Sam Leroux, Steven Bohez, Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. The cascading neural network: building the internet of smart things. *KAIS*, 2017.

[49] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, Fisher Yu, and Joseph E Gonzalez. Idk cascades: Fast deep learning by learning not to overthink. In *AUAI*, 2017.

[50] Jiaqi Guan, Yang Liu, Qiang Liu, and Jian Peng. Energy-efficient amortized inference with cascaded deep classifiers. In *IJCAI*, 2018.

[51] Xin Dai, Xiangnan Kong, and Tian Guo. Epnet: Learning to exit with flexible multi-branch network. In *CIKM*, 2020.

[52] Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and QI JU. FastBERT: a Self-distilling BERT with Adaptive Inference Time. In *ACL*, 2020.

[53] Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference. In *ACL*, 2020.

[54] Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. The Right Tool for the Job: Matching Model and Instance Complexities. In *ACL*, 2020.

[55] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. BERT Loses Patience: Fast and Robust Inference with Early Exit. *arXiv:2006.04152 [cs]*, 2020.

[56] Hehe Fan, Zhongwen Xu, Linchao Zhu, Chenggang Yan, Jianjun Ge, and Yi Yang. Watching a small portion could be as good as watching all: Towards efficient video classification. In *JICAI*, 2018.

[57] Wenhao Wu, Dongliang He, Xiao Tan, Shifeng Chen, Yi Yang, and Shilei Wen. Dynamic Inference: A New Approach Toward Efficient Video Action Recognition. In *CVPR Workshop*, 2020.

[58] Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. In *KDD*, 2017.

[59] Keyi Yu, Yang Liu, Alexander G. Schwing, and Jian Peng. Fast and accurate text classification: Skimming, rereading and early stopping. In *ICLR Workshop*, 2018.

[60] Xianggen Liu, Lili Mou, Haotian Cui, Zhengdong Lu, and Sen Song. Finding decision jumps in text classification. *Neurocomputing*, 2020.

[61] Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *ICML*, 2017.

[62] Zequn Jie, Peng Sun, Xin Li, Jiashi Feng, and Wei Liu. Anytime recognition with routing convolutional networks. *TPAMI*, 2019.

[63] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks. In *ICCV*, 2019.

[64] Sam Leroux, Pavlo Molchanov, Pieter Simoens, Bart Dhoedt, Thomas Breuel, and Jan Kautz. IamNN: Iterative and Adaptive Mobile Neural Network for Efficient Image Classification. In *ICML Workshop*, 2018.

[65] Qiushan Guo, Zhipeng Yu, Yichao Wu, Ding Liang, Haoyu Qin, and Junjie Yan. Dynamic recursive neural network. In *CVPR*, 2019.

[66] Haichao Yu, Haoxiang Li, Honghui Shi, Thomas S Huang, and Gang Hua. Any-precision deep neural networks. *arXiv preprint arXiv:1911.07346*, 2019.

[67] Qing Jin, Linjie Yang, and Zhenyu Liao. Adabits: Neural network quantization with adaptive bit-widths. In *CVPR*, 2020.

[68] Jianghao Shen, Yonggan Fu, Yue Wang, Pengfei Xu, Zhangyang Wang, and Yingyan Lin. Fractional skipping: Towards finer-grained dynamic cnn inference. In *AAAI*, 2020.

[69] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *CVPR*, 2018.

[70] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[71] Kyunghyun Cho and Yoshua Bengio. Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning. *arXiv preprint arXiv:1406.7362*, 2014.

[72] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *ICLR Workshop*, 2016.

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

[74] David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. In *ICLR Workshop*, 2013.

[75] Ravi Teja Mullapudi, William R Mark, Noam Shazeer, and Kayvon Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. In *CVPR*, 2018.

[76] Shaofeng Cai, Yao Shu, and Wei Wang. Dynamic routing networks. In *WACV*, 2021.

[77] William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *arXiv e-prints*, 2021.

[78] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[79] Weizhe Hua, Yuan Zhou, Christopher M De Sa, Zhiru Zhang, and G Edward Suh. Channel gating neural networks. In *NeurIPS*, 2019.

[80] Zhihang Yuan, Bingzhe Wu, Zheng Liang, Shiwan Zhao, Weichen Bi, and Guangyu Sun. S2dnas: Transforming static cnn model for dynamic inference via neural architecture search. In *ECCV*, 2020.

[81] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng zhong Xu. Dynamic channel pruning: Feature boosting and suppression. In *ICLR*, 2019.

[82] Charles Herrmann, Richard Strong Bowen, and Ramin Zabih. An end-to-end approach for speeding up neural network inference. *arXiv preprint arXiv:1812.04180*, 2018.

[83] Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. Batch-shaping for learning conditional channel gated networks. In *ICLR*, 2020.

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

[85] 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. *JSTSP*, 2020.

[86] Wenhan Xia, Hongxu Yin, Xiaoliang Dai, and Niraj K Jha. Fully dynamic inference with deep neural networks. *arXiv preprint arXiv:2007.15151*, 2020.

[87] Ali Ehteshami Bejnordi and Ralf Krestel. Dynamic channel and layer gating in convolutional neural networks. In *KI*, 2020.

[88] Zhourong Chen, Yang Li, Samy Bengio, and Si Si. You look twice: Gaternet for dynamic filter selection in cnns. In *CVPR*, 2019.

[89] Chuanjian Liu, Yunhe Wang, Kai Han, Chunjing Xu, and Chang Xu. Learning instance-wise sparsity for accelerating deep models. In *IJCAI*, 2019.

[90] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *ICLR*, 2018.

[91] Augustus Odena, Dieterich Lawson, and Christopher Olah. Changing model behavior at test-time using reinforcement learning. In *ICLR Workshop*, 2017.

[92] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *AAAI*, 2018.

[93] Samuel Rota Bulo and Peter Kontschieder. Neural decision forests for semantic image labelling. In *CVPR*, 2014.

[94] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *ICCV*, 2015.

[95] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

[96] Thomas M Hehn, Julian FP Kooij, and Fred A Hamprecht. End-to-end learning of decision trees and forests. *IJCV*, 2019.

[97] Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The Tree Ensemble Layer: Differentiability meets Conditional Computation. *arXiv preprint arXiv:2002.07772*, 2020.

[98] Zhicheng Yan, Hao Zhang, Robinson Piramuthu, Vignesh Jagadeesh, Dennis DeCoste, Wei Di, and Yizhou Yu. Hd-cnn: hierarchical deep convolutional neural networks for large scale visual recognition. In *ICCV*, 2015.

[99] Yani Ioannou, Duncan Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*, 2016.

[100] Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *ICML*, 2019.

[101] Yanwei Li, Lin Song, Yukang Chen, Zeming Li, Xiangyu Zhang, Xingang Wang, and Jian Sun. Learning Dynamic Routing for Semantic Segmentation. In *CVPR*, 2020.

[102] An-Chieh Cheng, Chieh Hubert Lin, Da-Cheng Juan, Wei Wei, and Min Sun. Instanas: Instance-aware neural architecture search. In *AAAI*, 2020.

[103] Adam W. Harley, Konstantinos G. Derpanis, and Iasonas Kokkinos. Segmentation-aware convolutional networks using local attention masks. In *ICCV*, 2017.

[104] Hang Su, Varun Jampani, Deqing Sun, Orazio Gallo, Erik Learned-Miller, and Jan Kautz. Pixel-adaptive convolutional neural networks. In *CVPR*, 2019.

[105] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017.

[106] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *CVPR*, 2019.

[107] Hang Gao, Xizhou Zhu, Stephen Lin, and Jifeng Dai. Deformable Kernels: Adapting Effective Receptive Fields for Object Deformation. In *ICLR*, 2019.

[108] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. In *NeurIPS*, 2013.

[109] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 1992.

[110] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *NeurIPS*, 2016.

[111] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *ICLR*, 2016.

[112] Ningning Ma, Xiangyu Zhang, Jiawei Huang, and Jian Sun. WeightNet: Revisiting the Design Space of Weight Networks. In *ECCV*, 2020.

[113] Martin Simonovsky and Nikos Komodakis. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *CVPR*, 2017.

[114] Di Kang, Debarun Dhar, and Antoni Chan. Incorporating side information by adaptive convolution. In *NeurIPS*, 2017.

[115] HyunJae Lee, Hyo-Eun Kim, and Hyeonseob Nam. Srm: A style-based recalibration module for convolutional neural networks. In *ICCV*, 2019.

[116] Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. ECA-net: Efficient channel attention for deep convolutional neural networks. In *CVPR*, 2020.

[117] Jingda Guo, Xu Ma, Andrew Sansom, Mara McGuire, Andrew Kalaani, Qi Chen, Sihai Tang, Qing Yang, and Song Fu. Spanet: Spatial Pyramid Attention Network for Enhanced Image Recognition. In *ICME*, 2020.

[118] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. In *CVPR*, 2017.

[119] Abhijit Guha Roy, Nassir Navab, and Christian Wachinger. Concurrent spatial and channel 'squeeze & excitation'in fully convolutional networks. In *MICCAI*, 2018.

[120] Long Chen, Hanwang Zhang, Jun Xiao, Liqiang Nie, Jian Shao, Wei Liu, and Tat-Seng Chua. Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. In *CVPR*, 2017.

[121] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Andrea Vedaldi. Gather-excite: Exploiting feature context in convolutional neural networks. In *NeurIPS*, 2018.

[122] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic relu. In *ECCV*, 2020.

[123] Ningning Ma, Xiangyu Zhang, and Jian Sun. Funnel activation for visual recognition. In *ECCV*, 2020.

[124] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. In *CVPR*, 2019.

[125] Shenlong Wang, Linjie Luo, Ning Zhang, and Li-Jia Li. Autoscaler: Scale-attention networks for visual correspondence. In *BMVC*, 2017.

[126] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *CVPR*, 2018.

[127] Kaiyu Yue, Ming Sun, Yuchen Yuan, Feng Zhou, Errui Ding, and Fuxin Xu. Compact generalized non-local network. In *NeurIPS*, 2018.

[128] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[129] Sneha Chaudhari, Gungor Polatkan, Rohan Ramanath, and Varun Mithal. An attentive survey of attention models. *arXiv preprint arXiv:1904.02874*, 2019.

[130] Xizhou Zhu, Dazhi Cheng, Zheng Zhang, Stephen Lin, and Jifeng Dai. An empirical study of spatial attention mechanisms in deep networks. In *ICCV*, 2019.

[131] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *arXiv preprint arXiv:2101.01169*, 2021.

[132] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *CVPR*, 2016.

[133] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. SBNet: Sparse Blocks Network for Fast Inference. *CVPR*, 2018.

[134] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *CVPR*, 2017.

[135] Shijie Cao, Lingxiao Ma, Wencong Xiao, Chen Zhang, Yunxin Liu, Lintao Zhang, Lanshun Nie, and Zhi Yang. Seernet: Predicting convolutional neural network feature-map sparsity through low-bit quantization. In *CVPR*, 2019.

[136] Shu Kong and Charless Fowlkes. Pixel-wise attentional gating for scene parsing. In *WACV*, 2019.

[137] Thomas Verelst and Tinne Tuytelaars. Dynamic Convolutions: Exploiting Spatial Sparsity for Faster Inference. In *CVPR*, 2020.

[138] Zhenda Xie, Zheng Zhang, Xizhou Zhu, Gao Huang, and Stephen Lin. Spatially Adaptive Inference with Stochastic Feature Sampling and Interpolation. In *ECCV*, 2020.

[139] Amjad Almahairi, Nicolas Ballas, Tim Cooijmans, Yin Zheng, Hugo Larochelle, and Aaron Courville. Dynamic capacity networks. In *ICML*, 2016.

[140] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *CVPR*, 2020.

[141] Aritra Bhowmik, Suprosanna Shit, and Chandra Sekhar Seelamantula. Training-free, single-image super-resolution using a dynamic convolutional network. *IEEE Signal Processing Letters*, 2017.

[142] Jialin Wu, Dai Li, Yu Yang, Chandrajit Bajaj, and Xiangyang Ji. Dynamic filtering with large sampling field for convnets. In *ECCV*, 2018.

[143] Xuecai Hu, Haoyuan Mu, Xiangyu Zhang, Zilei Wang, Tieniu Tan, and Jian Sun. Meta-SR: A magnification-arbitrary network for super-resolution. In *CVPR*, 2019.

[144] Jiaqi Wang, Kai Chen, Rui Xu, Ziwei Liu, Chen Change Loy, and Dahua Lin. CARAFE: Content-Aware ReAssembly of FEatures. In *ICCV*, 2019.

[145] Jin Chen, Xijun Wang, Zichao Guo, Xiangyu Zhang, and Jian Sun. Dynamic region-aware convolution. *arXiv preprint arXiv:2003.12243*, 2020.

[146] Guangrun Wang, Keze Wang, and Liang Lin. Adaptively connected neural networks. In *CVPR*, 2019.

[147] Max Jaderberg, Karen Simonyan, and Andrew Zisserman. Spatial transformer networks. In *NeurIPS*, 2015.

[148] Adria Recasens, Petr Kellnhofer, Simon Stent, Wojciech Matusik, and Antonio Torralba. Learning to zoom: a saliency-based sampling layer for neural networks. In *ECCV*, 2018.

[149] Volodymyr Mnih, Nicolas Heess, and Alex Graves. Recurrent models of visual attention. In *NeurIPS*, 2014.

[150] Jianlong Fu, Heliang Zheng, and Tao Mei. Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition. In *CVPR*, 2017.

[151] Zhichao Li, Yi Yang, Xiao Liu, Feng Zhou, Shilei Wen, and Wei Xu. Dynamic computational time for visual attention. In *ICCV Workshop*, 2017.

[152] Amir Rosenfeld and Shimon Ullman. Visual concept recognition and localization via iterative introspection. In *ACCV*, 2016.

[153] Zekun Hao, Yu Liu, Hongwei Qin, Junjie Yan, Xiu Li, and Xiaolin Hu. Scale-aware face detection. In *CVPR*, 2017.

[154] Zerui Yang, Yuhui Xu, Wenrui Dai, and Hongkai Xiong. Dynamic-stride-net: deep convolutional neural network with dynamic stride. In *SPIE Optoelectronic Imaging and Multimedia Technology*, 2019.

[155] Huiyu Wang, Aniruddha Kembhavi, Ali Farhadi, Alan L. Yuille, and Mohammad Rastegari. Elastic: Improving cnns with dynamic scaling policies. In *CVPR*, 2019.

[156] Adams Wei Yu, Hongrae Lee, and Quoc Le. Learning to Skim Text. In *ACL*, 2017.

[157] Víctor Campos, Brendan Jou, Xavier Giró-I-Nieto, Jordi Torres, and Shih Fu Chang. Skip RNN: Learning to skip state updates in recurrent neural networks. In *ICLR*, 2018.

[158] Christian Hansen, Casper Hansen, Stephen Alstrup, Jakob Grue Simonsen, and Christina Lioma. Neural Speed Reading with Structural-Jump-LSTM. In *ICLR*, 2019.

[159] Jin Tao, Urmish Thakker, Ganesh Dasika, and Jesse Beu. Skipping RNN State Updates without Retraining the Original Model. In *SenSys-ML*, 2019.

[160] Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. Variable computation in recurrent neural networks. In *ICLR*, 2017.

[161] Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Neural Speed Reading via Skim-RNN. In *ICLR*, 2018.

[162] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. In *ICLR*, 2017.

[163] Nan Rosemary Ke, Konrad Żołna, Alessandro Sordoni, Zhouhan Lin, Adam Trischler, Yoshua Bengio, Joelle Pineau, Laurent Charlin, and Christopher Pal. Focused Hierarchical RNNs for Conditional Sequence Processing. In *ICML*, 2018.

[164] Zhengjie Huang, Zi Ye, Shuangyin Li, and Rong Pan. Length adaptive recurrent model for text classification. In *CIKM*, 2017.

[165] Tsu-Jui Fu and Wei-Yun Ma. Speed Reading: Learning to Read ForBackward via Shuttle. In *EMNLP*, 2018.

[166] Zuxuan Wu, Caiming Xiong, Yu-Gang Jiang, and Larry S. Davis. Liteeval: A coarse-to-fine framework for resource efficient video recognition. In *NeurIPS*, 2019.

[167] Guillaume Vaudaux-Ruth, Adrien Chan-Hon-Tong, and Catherine Achard. ActionSpotter: Deep Reinforcement Learning Framework for Temporal Action Spotting in Videos. *arXiv:2004.06971 [cs]*, 2020.

[168] Serena Yeung, Olga Russakovsky, Greg Mori, and Li Fei-Fei. End-to-end learning of action detection from frame glimpses in videos. In *CVPR*, 2016.

[169] Yu-Chuan Su and Kristen Grauman. Leaving some stones unturned: dynamic feature prioritization for activity detection in streaming video. In *ECCV*, 2016.

[170] Zuxuan Wu, Caiming Xiong, Chih-Yao Ma, Richard Socher, and Larry S. Davis. AdaFrame: Adaptive Frame Selection for Fast Video Recognition. In *CVPR*, 2019.

[171] Yongming Rao, Jiwen Lu, and Jie Zhou. Attention-aware deep reinforcement learning for video face recognition. In *ICCV*, 2017.

[172] Wenhao Wu, Dongliang He, Xiao Tan, Shifeng Chen, and Shilei Wen. Multi-agent reinforcement learning based frame sampling for effective untrimmed video recognition. In *ICCV*, 2019.

[173] Bruno Korbar, Du Tran, and Lorenzo Torresani. Scsampler: Sampling salient clips from video for efficient action recognition. In *ICCV*, 2019.

[174] Humam Alwassel, Fabian Caba Heilbron, and Bernard Ghanem. Action search: Spotting actions in videos and its application to temporal action localization. In *ECCV*, 2018.

[175] Yue Meng, Chung-Ching Lin, Rameswar Panda, Prasanna Sattigeri, Leonid Karlinsky, Aude Oliva, Kate Saenko, and Rogerio Feris. Ar-net: Adaptive frame resolution for efficient action recognition. In *ECCV*, 2020.

[176] Yansong Tang, Yi Tian, Jiwen Lu, Peiyang Li, and Jie Zhou. Deep Progressive Reinforcement Learning for Skeleton-Based Action Recognition. In *CVPR*, 2018.

[177] Yin-Dong Zheng, Zhaoyang Liu, Tong Lu, and Limin Wang. Dynamic Sampling Networks for Efficient Action Recognition in Videos. *TIP*, 2020.

[178] Hengduo Li, Zuxuan Wu, Abhinav Shrivastava, and Larry S Davis. 2d or not 2d? adaptive 3d convolution selection for efficient video recognition. *arXiv preprint arXiv:2012.14950*, 2020.

[179] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *ICML*, 2017.

[180] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *CVPR*, 2019.

[181] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *TPAMI*, 1998.

[182] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *CVPR*, 2015.

[183] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *CVPR*, 2013.

[184] Anelia Angelova, Alex Krizhevsky, Vincent Vanhoucke, Abhijit Ogale, and Dave Ferguson. Real-Time Pedestrian Detection with Deep Network Cascades. In *BMVC*, 2015.

[185] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In *CVPR*, 2016.

[186] Hong-Yu Zhou, Bin-Bin Gao, and Jianxin Wu. Adaptive feeding: Achieving fast and accurate detections by adaptively combining object detectors. In *ICCV*, 2017.

[187] Tong Yang, Xiangyu Zhang, Zeming Li, Wenqiang Zhang, and Jian Sun. Metaanchor: Learning to detect objects with customized anchors. In *NeurIPS*, 2018.

[188] Chunlin Chen and Qiang Ling. Adaptive Convolution for Object Detection. *IEEE Transactions on Multimedia*, 2019.

[189] Hiroki Tokunaga, Yuki Teramoto, Akihiko Yoshizawa, and Ryoma Bise. Adaptive weighting multi-field-of-view cnn for semantic segmentation in pathology. In *CVPR*, 2019.

[190] Gernot Riegler, Samuel Schulter, Matthias Ruther, and Horst Bischof. Conditioned regression models for non-blind single image super-resolution. In *ICCV*, 2015.

[191] Falong Shen, Shuicheng Yan, and Gang Zeng. Neural style transfer via meta networks. In *CVPR*, 2018.

[192] Yu-Gang Jiang, Changmao Cheng, Hangyu Lin, and Yanwei Fu. Learning layer-skippable inference network. *TIP*, 2020.

[193] Junjun He, Zhongying Deng, and Yu Qiao. Dynamic multi-scale filters for semantic segmentation. In *ICCV*, 2019.

[194] Dmitrii Marin, Zijian He, Peter Vajda, Priyam Chatterjee, Sam Tsai, Fei Yang, and Yuri Boykov. Efficient segmentation: Learning downsampling near semantic boundaries. In *ICCV*, 2019.

[195] Jun Li, Yongjun Chen, Lei Cai, Ian Davidson, and Shuiwang Ji. Dense transformer networks for brain electron microscopy image segmentation. In *IJCAI*, 2019.

[196] Fei Wu, Feng Chen, Xiao-Yuan Jing, Chang-Hui Hu, Qi Ge, and Yimu Ji. Dynamic attention network for semantic segmentation. *Neurocomputing*, 2020.

[197] Zilong Zhong, Zhong Qiu Lin, Rene Bidart, Xiaodan Hu, Ibrahim Ben Daya, Zhifeng Li, Wei-Shi Zheng, Jonathan Li, and Alexander Wong. Squeeze-and-Attention Networks for Semantic Segmentation. In *CVPR*, 2020.

[198] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *ECCV*, 2018.

[199] Xihui Liu, Guojun Yin, Jing Shao, Xiaogang Wang, and hongsheng Li. Learning to Predict Layout-to-image Conditional Convolutions for Semantic Image Synthesis. In *NeurIPS*, 2019.

[200] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019.

[201] Peihao Zhu, Rameen Abdal, Yipeng Qin, and Peter Wonka. SEAN: Image Synthesis with Semantic Region-Adaptive Normalization. In *CVPR*, 2020.

[202] Meng Chang, Qi Li, Huajun Feng, and Zhihai Xu. Spatial-adaptive network for single image denoising. In *ECCV*, 2020.

[203] Tianjun Xiao, Yichong Xu, Kuiyuan Yang, Jiaxing Zhang, Yuxin Peng, and Zheng Zhang. The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In *CVPR*, 2015.

[204] Heliang Zheng, Jianlong Fu, Tao Mei, and Jiebo Luo. Learning multi-attention convolutional neural network for fine-grained image recognition. In *ICCV*, 2017.

[205] Wanjie Sun and Zhenzhong Chen. Learned image downscaling for upscaling using content adaptive resampler. *TIP*, 2020.

[206] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. In *ICLR*, 2015.

[207] SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, and Geoffrey E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models. In *NeurIPS*, 2016.

[208] Ali Diba, Vivek Sharma, Luc Van Gool, and Rainer Stiefelhagen. Dynamonet: Dynamic action and motion network. In *ICCV*, 2019.

[209] Ruohan Gao, Tae-Hyun Oh, Kristen Grauman, and Lorenzo Torresani. Listen to look: Action recognition by previewing audio. In *CVPR*, 2020.

[210] Yu-Syuan Xu, Tsu-Jui Fu, Hsuan-Kung Yang, and Chun-Yi Lee. Dynamic video segmentation network. In *CVPR*, 2018.

[211] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive separable convolution. In *ICCV*, 2017.

[212] Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. In *CVPR*, 2017.

[213] Younghyun Jo, Seoung Wug Oh, Jaeyeon Kang, and Seon Joo Kim. Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation. In *CVPR*, 2018.

[214] Tae Hyun Kim, Kyoung Mu Lee, Bernhard Scholkopf, and Michael Hirsch. Online video deblurring via dynamic temporal blending network. In *CVPR*, 2017.

[215] Shangchen Zhou, Jiawei Zhang, Jinshan Pan, Haozhe Xie, Wangmeng Zuo, and Jimmy Ren. Spatio-temporal filter adaptive network for video deblurring. In *ICCV*, 2019.

[216] Lei Chen, Jiwen Lu, Zhanjie Song, and Jie Zhou. Part-activated deep reinforcement learning for action prediction. In *ECCV*, 2018.

[217] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *ICCV*, 2019.

[218] Jie Li, Kai Han, Peng Wang, Yu Liu, and Xia Yuan. Anisotropic convolutional networks for 3d semantic scene completion. In *CVPR*, 2020.

[219] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

[220] Peng Gao, Hongsheng Li, Shuang Li, Pan Lu, Yikang Li, Steven CH Hoi, and Xiaogang Wang. Question-guided hybrid convolution for visual question answering. In *ECCV*, 2018.

[221] Xiaotian Jiang, Quan Wang, and Bin Wang. Adaptive convolution for multi-relational learning. In *NAACL*, 2019.

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

[223] Weiping Song, Zhiping Xiao, Yifan Wang, Laurent Charlin, Ming Zhang, and Jian Tang. Session-based social recommendation via dynamic graph attention networks. In *WSDM*, 2019.

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

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

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

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

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

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

[230] Dongliang He, Zhichao Zhou, Chuang Gan, Fu Li, Xiao Liu, Yandong Li, Limin Wang, and Shilei Wen. Stnet: Local and global spatial-temporal modeling for action recognition. In *AAAI*, 2019.

[231] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *ICML*, 2019.

[232] Rahul Duggal, Scott Freitas, Sunny Dhamnani, Duen Horng, Chau, and Jimeng Sun. ELF: An Early-Exiting Framework for Long-Tailed Classification. *arXiv:2006.11979 [cs, stat]*, 2020.

[233] Ting-Kuei Hu, Tianlong Chen, Haotao Wang, and Zhangyang Wang. Triple Wins: Boosting Accuracy, Robustness and Efficiency Together by Enabling Input-Adaptive Inference. In *ICLR*, 2020.

[234] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *ICLR*, 2018.

[235] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: Transfer learning through adaptive fine-tuning. In *CVPR*, 2019.

[236] Sanghyun Hong, Yiğitcan Kaya, Ionuţ-Vlad Modoranu, and Tudor Dumitraş. A panda? no, it's a sloth: Slowdown attacks on adaptive multi-exit neural network inference. *arXiv preprint arXiv:2010.02432*, 2020.