Bringing AI To Edge: From Deep Learning's Perspective

Di Liu, Hao Kong, Xiangzhong Luo, Weichen Liu, Ravi Subramaniam

***Abstract*—Edge computing and artificial intelligence (AI), especially deep learning for nowadays, are gradually intersecting to build a novel system, called *edge intelligence*. However, the development of edge intelligence systems encounters some challenges, and one of these challenges is the *computational gap* between computation-intensive deep learning algorithms and less-capable edge systems. Due to the computational gap, many edge intelligence systems cannot meet the expected performance requirements. To bridge the gap, a plethora of deep learning techniques and optimization methods are proposed in the past years: light-weight deep learning models, network compression, and efficient neural architecture search. Although some reviews or surveys have partially covered this large body of literature, we lack a systematic and comprehensive review to discuss all aspects of these deep learning techniques which are critical for edge intelligence implementation. As various and diverse methods which are applicable to edge systems are proposed intensively, a holistic review would enable edge computing engineers and community to know the state-of-the-art deep learning techniques which are instrumental for edge intelligence and to facilitate the development of edge intelligence systems. This paper surveys the representative and latest deep learning techniques that are useful for edge intelligence systems, including hand-crafted models, model compression, hardware-aware neural architecture search and adaptive deep learning models. Finally, based on observations and simple experiments we conducted, we discuss some future directions.**

***Index Terms*—Deep learning, model optimization, edge computing, neural architecture search**

I. Introduction

In 2012, AlexNet [1] broke the record of ImageNet LSVRC contest [2], improving the prediction accuracy by a large margin of 10%, which marked the milestone for the booming development of deep learning (DL) techniques, or more specifically deep neural network (DNN) [3]. In the subsequent years, as the increasing number of DNN applications [4] emerges, designing DNN-based systems attracts a lot of attention and efforts from academia and industry, spanning from advanced image manipulation and enhancement and convenient voice assistant on mobile phones to Level-4 autonomous driving systems on automotives [5]. AlphaGo [6] is another prominent example which defeated the top professional player in the ancient Chinese game GO, deemed impossible before due to its extremely high complexity.

The successful application of DNN models relies on two stages: **training** and **inference/deployment**. Training indicates the procedure of learning a predictive model from a huge amount of labeled data, while inference denotes the procedure of using the trained model upon new data to make an accurate prediction. The training procedure involves a complex learning process requiring powerful computational units and a huge amount of data and spending considerable time. For instance, training ResNet50 [7] with ImageNet on 8 Nvidia Tesla P100 takes 29 hours [8], where ImageNet dataset [2] has 1.28 M images of 1000 classes for training. Recently, researchers also measure the environmental effect of the training procedure and indicate that training DNN consumes excessive energy and emits substantial CO_2 [9] and environment-sustainable DNN models should be considered.

The trained DNN model is exploited to make prediction on new data, and this procedure is usually called DNN *inference*. Many companies implement their DNN inference on servers to provide convenient services for their customers such as voice-assistant, machine translation, image retrieval, etc [10]. However, inference from cloud suffers from *responsiveness* (e.g., latency) and *privacy* issues which are critical in some scenarios. For instance, DNNs are seen to be a pivotal technique for self-driving cars, in which most of executions are subject to highly rigorous real-time constraints [5], [11], but the high and uncertain communication overhead makes it difficult to satisfy the temporal requirement. In addition, some inference is conducted on confidential data, e.g., manufacturing and production data, and uploading these data to cloud may risk data leakage which will severely hurt their business. To address these concerns, *edge computing* is proposed [12].

Edge computing systems are placed at the proximity of data producer, like sensors, end-users, etc, to rapidly and locally process data. The emergence of edge computing paves the way for pervasive intelligent systems [13]. IEEE Computer Society identifies 'Artificial Intelligence at the edge' as one of the top 12 technology trends to reach adoption in 2020 [14]. Edge systems have a broad spectrum of hardware features, from powerful computation units in communication stations and self-driving cars to small and battery-supplied smart phones and wearable devices. Fig. 1 plots the overview of edge

Di Liu, Hao Kong, and Xiangzhong Luo contribute equally to this work.

This research was conducted in collaboration with HP Inc. and supported by National Research Foundation (NRF) Singapore and the Singapore Government through the Industry Alignment Fund-Industry Collaboration Projects Grant (I1801E0028). This work is also partially supported by NTU NAP M4082282 and SUG M4082087, Singapore.

Di Liu is with HP-NTU Digital Manufacturing Corporate Lab, Nanyang Technological University, Singapore. (Email: liu.di@ntu.edu.sg)

Hao Kong and Weichen Liu are with HP-NTU Digital Manufacturing Corporate Lab, Nanyang Technological University, Singapore, and School of Computer Science and Engineering, Nanyang Technological University, Singapore. (Email: kong.hao@ntu.edu.sg, liu@ntu.edu.sg)

Xiangzhong Luo is with School of Computer Science and Engineering, Nanyang Technological University, Singapore. (Email: xiangzho001@e.ntu.edu.sg)

Ravi Subramaniam is with Innovations and Experiences–Business Personal Systems, HP Inc., USA. (Email: ravi.subramaniam@hp.com)

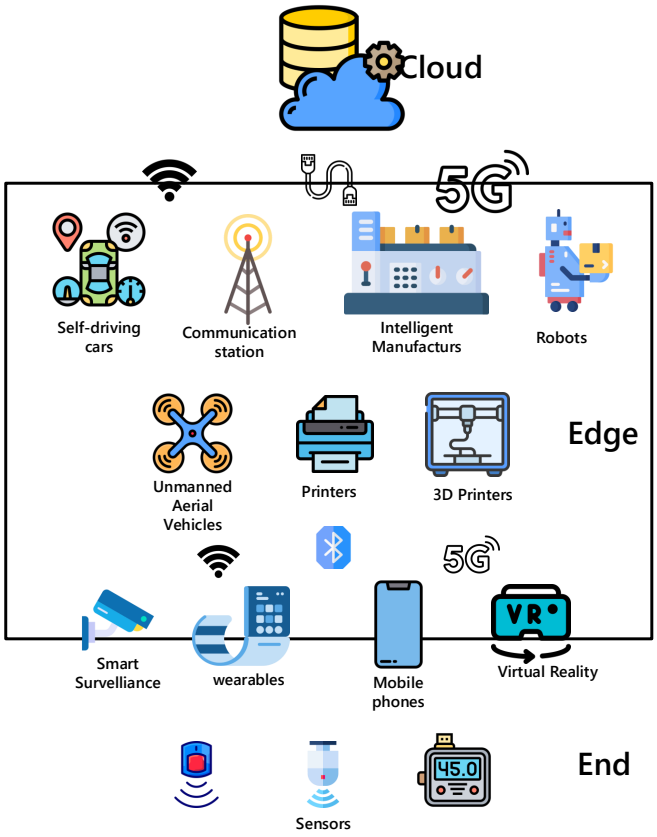

Fig. 1. The wide spectrum of Edge Computing

computing. In different scenarios, edge systems are subject to diverse performance and physical constraints, e.g., latency and power constraints for self-driving cars, computational capability and power constraints for UAVs, etc. However, DNN models, the current core component of artificial intelligence, are highly computation-intensive and are even growing '*wider*' (more filters within a layer) and '*deeper*' (more layers) with billions of parameters and millions of float point operations (FLOPs) (Some preliminaries about DNN are given in Section II). Such '*deeper*' models provide high accuracy at the cost of highly computational complexity. This unavoidably leads to *'computational gap'* between DNN models and less-capable edge systems.

Many approaches are proposed to bridge the '*computational gap*' between the complex models and resource-constrained edge systems. From hardware perspective, different accelerators are proposed and many companies are dedicated to develop specific hardware architectures to speed up the execution of DNN models, such as application-specific integrated circuits (ASICs) Tensor Processing Unit (TPU) [15] and DianNao family [16], and FPGA-based accelerators ESE [17] [18]. However, developing chips is a costly process, 30-80 million dollars and 2-3 years to develop [19]. Besides, the benefit of porting DNN applications to these new computing units is still under doubt. The research from Facebook shows the performance gain by porting DNN applications to specific hardware accelerators may not be able to compensate the significant engineering effort spent due to the poor ecosystems and program-ability [20]. In addition, some customized edge accelerators achieve high efficiency at the cost of generality, e.g., edge TPU only supports a limited number of operators [21] and some new DNN architectures cannot be well-supported.

On the other hand, software is known to be flexible and less costly to develop in comparison with hardware. For efficient edge intelligence systems, from software perspective, the goal is to design proper DNN models which can be fit on edge systems while guaranteeing the required performance and maximizing accuracy. Different methodologies are devised to approach this goal, like novel light-weight DNN model design, model compression, and the emergent neural architecture search (NAS) [22], all of which facilitate the development and application of DNN models on edge systems. In the past decade, many DNN breakthroughs have significantly improved DNNs' accuracy and performance and we believe the innovation from deep learning techniques will still be the key ingredient for the emergent edge intelligence.

A. Motivation Behind This Paper

There exist several excellent reviews about edge computing and edge intelligence. Zhou *et al.* [13] discuss the great potential of edge intelligence and point out some future research directions. Wang *et al.* [23] mainly review the edge intelligence systems from communication systems' perspective. Chen and Ran in [24] discuss the diverse applications when deep learning techniques are integrated into edge computing. All of these reviews only have a small fraction to discuss the edge DNN design which is the key element of edge intelligence systems.

The vast DNN design methods applicable to edge systems can be broadly classified into three categories: *hand-crafted models*, *model compression* and *hardware-aware NAS*. Some prior DNN reviews cover one of the three categories, like [25] [26] for model compression, [27] for early works on efficient DNN processing and [28] for NAS[1] but there is no review discussing all three categories which in future will be integrated seamlessly to design effective and efficient DNNs in the new edge intelligence era. Considering the anticipated emergence of numerous edge intelligence systems in the next years [13], this motivates us to comprehensively review the relevant works about edge DNN design in a holistic fashion.

In addition, except the static DNN design for edge systems, we also discuss adaptive DNN models which are also not reviewed systematically in previous articles. Edge systems installed in real world operate in highly dynamic environment, while it still needs to guarantee a certain degree of quality of service (QoS) and performance, like real-time requirement. However, static DNN models are unable to achieve such guarantee under dynamic environment [20]. Hence, we need models which are able to adapt their computation to achieve a good trade-off between accuracy and efficiency.

[1]This survey is for the general NAS, not the hardware-aware NAS.

B. Structure of This Paper

In general, we consider this paper as a good complementary for other related surveys or reviews in the field of edge computing and hope this paper facilitates a profound understanding from deep learning's perspective, which plays a pivotal role in implementing edge intelligence systems. As shown in [29], around 50 papers were uploaded to Arxiv per day in 2018, discussing DNN related topics, i.e., in total more than 15000 papers per year. It is supposed to have more now. AlexNet which broke the record in 2012 now is rarely deemed as an important reference approach for experimental comparison. Therefore, it is impossible to review all related papers. In this paper, due to the space limitation, we strive to cover the most relevant, representative, and latest works[2] which, we believe, are adequate for readers to fully understand the development, motivation, and latest trend of these DL techniques applicable for edge systems and to have a full view of these techniques. Finally, we provide some of our thoughts about designing DNNs for edge devices based on some observations and experiments we have conducted.

Scope of this paper: Edge computing is an emergent research topic, consisting of many interesting and challenging research problems, like edge caching [30] and computation offloading [31], [32]. Prior review or surveys have profoundly discussed these topics. In this paper, we only focus on deep learning techniques which aid in developing edge intelligence systems. For other related topics, we refer interesting readers to other excellent literature [23], [30], [31].

The reminder of this paper is organized as follows:

- Section II introduces the preliminaries of DNN models to facilitate the understanding of techniques presented in subsequent sections.
- Section III discusses works designing novel DNN architectures for light-weight DNN models.
- Section IV reviews network compression methods, including network pruning, quantization, and knowledge distillation.
- Section V discusses NAS techniques aiming to design and customize efficient DNN models for resource-constrained systems, like some edge systems.
- Section VI discusses adaptive DNN models which may adapt the model computation under dynamic environment.
- Section VII demonstrates our thoughts on future edge DNN designs based on our observation and experimental results.
- Section VIII concludes this paper.

Fig 2 shows the detailed structure of this paper.

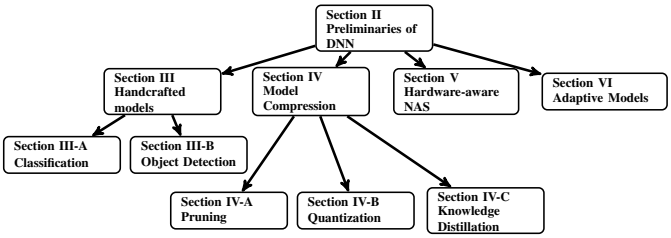

Fig. 2. The schematic structure of this paper

II. Preliminaries

In this section, we introduce some basic knowledge of DNNs for better understanding the techniques and approaches discussed in subsequent sections. In this paper, we mainly review the models and techniques proposed for computer vision applications and there are two reasons for this. First, currently computer vision models are the major application for edge intelligence systems; Second, although some recent studies start to investigate how to design lightweight natural language processing (NLP) models for edge devices like, [33] [34] [35], they are far from mature like models and techniques for computer vision tasks. Hence, in this paper, we use DNN models of computer vision as the major example for our presentation.

For computer vision applications, when referring to DNNs, we mean convolutional neural networks (CNN). Throughout this paper, we may use DNN and CNN interchangeably. CNNs usually have many convolutional layers, pooling layers, activation layers, and a couple of fully connected layers [3]. Figure 3 shows a simple CNN with two convolutional layers and one fully connected layer. The convolutional layer is the core ingredient in CNNs, which extracts the patterns/features from input data at different granularity. Convolutional layers account for the major resource and time cost of a CNN model. To better illustrate how the convolutional layer works, we give some terminologies about convolutional layers as follows:

- **Kernel** – A kernel is a 2D square matrix with size like 3×3, 5×5, 7×7 and a convolutional operand.
- **Filter** – A filter is a collection of kernels with size of $k \times k \times c$, where k is the kernel size and c is equal to the number of input channels. A filter is convolved with all input channels to derive a new channel/feature map/activation map.
- **Feature map/channel/activation** – Feature map, channel, and activation have the same meaning in CNNs. A feature map is a 2D feature matrix generated by a filter.

As demonstrated in Figure 3, the input of the first convolutional layer is an image with n_1 channels, and each channel is a 2D array with height h_1 and width w_1. Correspondingly, there are n_1 kernels in each filter, and the kernel size is $k_1 \times k_1$. There are n_2 filters, and thus n_2 feature maps will be generated after the first convolution operation and the size of the first feature maps is $(h_2 \times w_2 \times n_2)$, where h_2 and w_2 are the height and weight of the first feature maps, respectively. If the padding is used during convolution, then $h_1 = h_2$ and $w_1 = w_2$. Otherwise, $h_1 > h_2$ and $w_1 > w_2$. For more details, interesting readers are referred to [3].

Usually, a convolutional layer is followed by a pool layer and an activation layer. In Fig. 3 , we omit activation layers. Activation functions are used to introduce non-linearity into neural network, and the common activation functions are rectify linear units (ReLu) [36], sigmoid, etc. The pooling layer down-samples the feature map to reduce the spatial size

[2]We include peer-reviewed papers until 2020

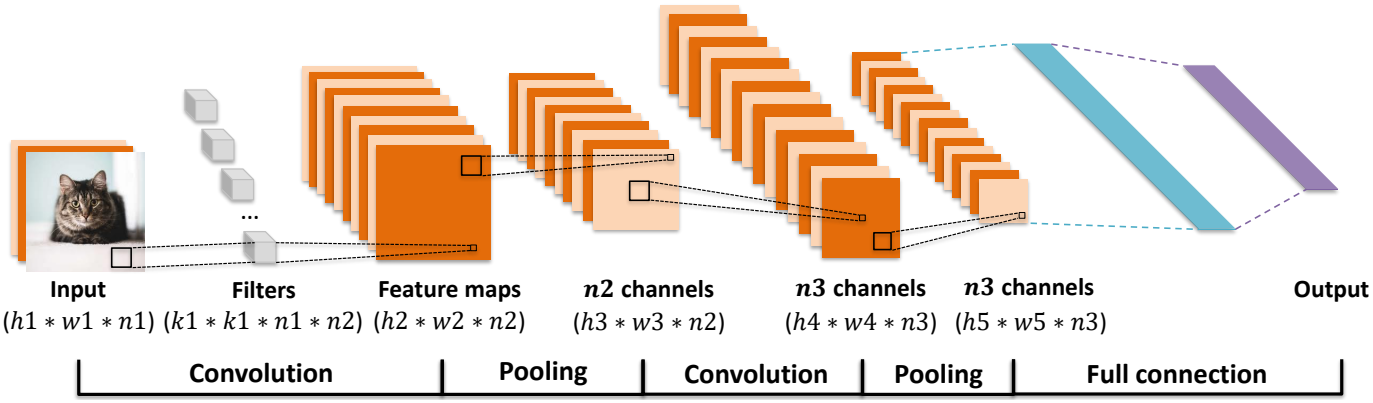

Fig. 3. The overview of a DNN/CNN model

of the feature map and increase its receptive field. The max pooling [37] and the average pooling are the two common pooling methods. A DNN model usually ends up with a couple of fully connected layers, which is used to fuse the feature information from the last convolutional layer and predict the classification of the input image. To improve the performance and training speed, modern DNN models also have other operation layers, like batch normalization layer [38], squeeze-and-excitation layer [39], etc.

III. Lightweight Network Design

AlexNet [1] marks the milestone of rapid development of DNN models while researchers find that the bigger a DNN model is, the better accuracy the model can provide. As a result, this incentivizes the emergence of increasingly complicated DNN models [40]. However, the highly computational complexity of DNN models hinders them from being efficiently deployed on resource-constraint devices, e.g., edge or IoT devices. A promising way to solve this problem is to design novel neural architectures/operators which are more efficient on edge systems while not compromising accuracy. In this section, we review the works which target to manually design lightweight models. For these hand-craft models, we classify them into two categories based on their applications: *classification* and *object-detection*. These two applications are the major deployment scenarios for edge intelligence systems.

A. Classification

Classification is the fundamental and most critical task in computer vision and it is also the first application in which DNNs demonstrated its huge potential [1]. Moreover, classification models play a core role in other computer vision tasks, such as semantic segmentation and object detection, where classification models serve as the backbone to extract features from images and provide essential feature information for segmentation and detection.

SqueezeNet [41] is one of the early works towards designing DNNs for resource-constrained hardware. The core idea in SqueezeNet is a new computational module, *Fire module*, in which convolutional operations are split into the squeeze layer and expand layer and some 3×3 convolutional operations are replace by low-complexity 1×1 convolution operations. SqueezeNet achieves AlexNet-level accuracy while greatly reducing the model complexity. The applicability of SqueezeNet is validated on real FPGA with small on-chip memory. Gschwend *et al.* [42] develop a variant of SqueezeNet and implement it on an FPGA. It shows the SqueezeNet-like model can be entirely fit within the FPGA on-chip memory. As the result, it significantly eliminates the off-chip memory access overhead and in turn improves the inference performance.

MobileNet series [43]–[45] are another prominent DNN models targeting resource-limited devices. MobileNet [43] replaces the conventional convolutional operation with more efficient depth-wise separable convolution operation proposed in [46] to reduce the computational cost. Depth-wise separable convolution operation factorizes a conventional $k \times k \times n$ convolution into a $k \times k \times 1$ depth-wise convolution and a $1 \times 1 \times n$ point-wise convolution. Each input channel is convolved with the depth-wise convolution operator and the point-wise convolution linearly combines all results from depth-wise convolution to generate one channel/feature map. Depth-wise separable convolution can significantly reduce the computational complexity, thus shortening inference time on edge devices. On the top of MobileNet, MobileNetV2 [44] adds linear bottleneck and inverted residual block to improve both accuracy and performance. The latest MobileNetV3 [45] combines NAS and NetAdapt [47] to design a more accurate and efficient network architecture.

TABLE I
Comparison of hand-crafted models. Each model has a scaling factor to scale up the number of channels. Here we only report the results of the scaling factor=1.

Model	Year	Parameters	FLOPs/MACs	Accuracy
SqueezeNet [41]	2016	1.24M		60.4%
Mobilenet [43]	2017	4.2M	/569M	70.6%
MobilenetV2 [44]	2018	3.4M	/300M	72%
MobilenetV2-large [44]	2018	6.9M	/585M	74.7%
MobilenetV3-small [45]	2019	2.5M	/56M	67.4%
MobilenetV3-large [45]	2019	5.4M	/300M	75.2%
ShuffleNet [48]	2018	3.4M	/292M	71.5%
ShuffleNetV2 [49]	2018	2.3M	146M/	69.4%
EfficientNet-B0 [50]	2019	5.3M	390M/	77.1%
GhostNet [51]	2020	5.2M	141M/	73.9%

ShuffleNet [48] deploys *group convolution* and *channel shuffle* to reduce the computational complexity while retaining high accuracy. ShuffleNetV2 [49] empirically observes four principles for designing efficient DNNs and proposes *channel split* to improve accuracy and performance. GhostNet [51] proposes a *ghost module* based on an observation that some features in convolutional layers are highly correlated. Thus, it first uses standard convolutional operation to obtain a few intrinsic features and then generates more features from the intrinsic features with cheap linear operations.

EfficientNet [50] investigates the influence of three scaling dimensions in DNN models, i.e., depth/layer scaling, channel scaling and resolution scaling, and proposes a compound scaling method such that given a DNN model and a target computational complexity (i.e., FLOPs) it can effectively adjust the three dimensions of a model to improve its predictive performance. The scaled models achieves a comparable or better accuracy while having fewer parameters.

Discussion: Table I summarizes the models discussed in this section. Significant progresses have been made in lightweight DNN models for classification task, and classification DNN models are also the key component for other CV tasks, like segmentation, detection, etc. The manual design of lightweight DNN models is considerably dependent on experts' knowledge and also needs a time-consuming hyperparameter exploration. As *automated machine learning* (AutoML[3]) [52] techniques, like NAS, hyperparameter optimization, emerge to aid in designing DNN models, designers can put their focus on designing effective and efficient DNN modules or operations. Then the new operations can be used as the fundamental elements to generate new models for edge systems. For instance, the latest MobileNetV3 exploits NAS with the novel architectures from MobileNetV2 and MnasNet [53] to find accurate and efficient DNN model for mobile setting. We think this is becoming a mainstream trend to adopt NAS with novel lightweight operators to design edge DNN model. However, existing methods overlook the impact of deployed hardware, on which some operators cannot be supported or effectively executed. *Thus, for the future DNN classification models, especially for edge systems, it is important to design models in a hardware-aware fashion.*

B. Object Detection

Object detection is another vital field in DNN research. Besides predicting the category of an input image, object detection locates objects in the input image by drawing bounding boxes. Generally, we can classify object detection methods into two categories: one-stage method and two-stage method. One-stage methods predict object classification and localization in one single forward pass. YOLO [64] and SSD [65] are two widely-used examples of one-stage methods; Two-stage methods first deploy a backbone CNN network (usually for classification) to extract features from input images and then a detection part uses the features extracted from the backbone network to localize objects. RCNN [66], Fast RCNN [67], and Faster RCNN [68] are representative examples of two-stage methods.

To boost the efficiency of object detection on resource-limited devices, some works strive to reduce the computational complexity of backbone part of object detection models, like Tiny-dsod [58] for DSOD [69], Tiny-SSD [59] for SSD [65], and Tiny-YOLO [57] for YOLO. Other efforts combines SSD framework with lightweight CNNs, like MobileNet, ShuffleNet, SequeezeNet, etc to improve efficiency. SequeezeDet [56] implements an object detection model by using SequeezeNet [41] as the backbone to improve the efficiency while not significantly compromising accuracy. Pelee [60] combines an improved SSD framework with its optimized PeleeNet to improve efficiency of object detection. Since one-stage methods are less computational than two-stage methods, all of these methods mentioned above target one-stage method for improving efficiency of object detection DNNs.

Few efforts strive to improve efficiency of two-stage methods. Li *et al.* [61] replace the heavy-head in RCNN framework to speed up execution. Qin *et al.* [62] observe a good configuration of input resolution, backbone network, and detection head can readuce the complexity of two-stage methods while maintaining competitive accuracy. Thus, they propose ThunderNet in which a variant of ShuffleNet, dubbed SNet, is proposed to implement an efficient DNN for object detection. EfficientDet [63] is the counterpart of EfficientNet in object detection, where a new bi-directional feature pyramid network is proposed to combine with EfficientNet to generate an efficient object detection detector.

Discussion: Table II summarizes all object detection works discussed in this section. Although object detection can be considered as an extension of classification, there lack adequate efforts to address the efficiency issues of object detection for resource-constrained systems. The two-stages methods provide high accuracy at the cost of efficiency, and even on powerful GPUs they cannot guarantee real-time constraints (e.g., 25fps for video). The one-stage approaches can trade off accuracy for efficiency, but the practical deployment on edge devices is still behind the expected performance [70]. In addition, we notice that only two works evaluate their designs on edge settings (on ARM CPU, mobile GPU or low power accelerators) and as pointed in [49] using the *indirect* metrics lke FLOPs or MACs cannot directly translate to the relevant performance metric, i.e., latency and throughput. *Therefore, for the future model design, evaluating models' direct performance on targeting platforms will be a necessity. In addition, for edge systems, the trade-off between speed and accuracy should be application or context dependent, so a good benchmark and design guide should be developed for practitioners [71].*

IV. Network Compression

Novel lightweight models provide us a solution to efficiently deploy DNN models on edge devices. However, designing a novel architecture is really challenging due to its large design space and complicated parameter tuning. Unfortunately, the majority of DNN models are designed to pursue better

[3]https://https://www.automl.org/

TABLE II
HAND-CRAFTED OBJECT DETECTION MODELS SUMMARY. ALL INFORMATION OF THIS TABLE IS FROM THE ORIGINAL PAPER OR THE PAPER WHICH COMPARE THESE MODELS. EXPERIMENTAL RESULTS ARE UPON TWO WIDELY-USED BENCHMARK, PASCAL VOC2007 [54] AND MS COCO [55].

Model	Year	Methods	Input	Parameters	FLOPs/MACs	mAP(PASCAL)	mAP(COCO)	Tested Platform
SqueezeDet [56]	2017	One-stage	1242×375	-	9.7B/	-	-	GPU
Tiny-YOLO [57]	2018	One-stage	416×416	15.12M	3.49B/	57.1%	-	GPU
Tiny-DSOD [58]	2018	One-stage	300×300	1.15M	1.12B/	72.1%	23.2%	GPU
Tiny-SSD [59]	2018	One-stage	300×300	1.13M	/571.09M	61.3%	-	GPU
Pelee [60]	2018	One-stage	320×320	5.98M	1.21B/	70.9%	22.4%	GPU/Edge GPU
Light-Head RCNN [61]	2017	Two-stage	800×1200	-	5.65BM/	75.1%	-	GPU
ThunderNet [62]	2019	Two-stage	320×320	-	4.61M/	75.1%	-	GPU/Edge CPU
EfficientDet [63]	2020	Two-stage	-	8.1M*	11B*/	-	43.0%*	GPU

*EfficientDet has models with different complexity. We report the results of EfficientDet D2 which may be applicable to edge systems.

accuracy without consideration of resource constraints of edge systems, where complex DNN models have a few hundred layers and several billions of parameters to achieve competitive accuracy. As indicated in [72], DNN models usually have significant redundancy in terms of weights and parameters. Then, an interesting question is raised:

> *Can we reduce the complexity of DNN models by removing these redundancy without greatly compromising their predictive performance?*

Network compression tackles this problem by removing the redundancy of over-parameterized networks. Network compression techniques generally fall into three categories: *network pruning* which removes the redundant weights and channels of over-parameterized DNNs, *quantization* which uses fewer bits to store DNN weights and intermediate results (e.g., float point 32, FP32 to integer 8, INT8), and *knowledge distill* which learns a small and compact (student) model from a large and over-parameterized (teacher) model. It is worthy noting that the three approaches are not mutual exclusive to each other and in many cases they are combined to maximally compress the redundant models. In this section, we discuss three approaches in details.

A. Network Pruning

The main motivation behind network pruning is that DNN models are usually over-parameterized in terms of weights and channels [72], and eliminating these redundancy within the model can hugely reduce the computational complexity and storage requirement. Many network pruning methods are proposed in the past 5 years, and we like to classify them into two major branches based on the pruned structure: *non-structure pruning* and *structure pruning*.

1) Non-Structure Pruning: Non-structured pruning technique, widely known as *weight pruning*, conducts a fine-grained operation by removing irrelevant weights in over-parameterized DNN models, as shown in Fig. 4. It removes individual weights within a kernel or individual neurons within a fully connected layer. *Non-structure pruning* can significantly reduce the number of parameters and memory footprint.

Weight pruning on neural network can be traced back to 1990s, [73], [74], where pruning on fully-connected neural networks was investigated. Deep Compression framework [75], [76] is one of the pioneers in DNN model compression.

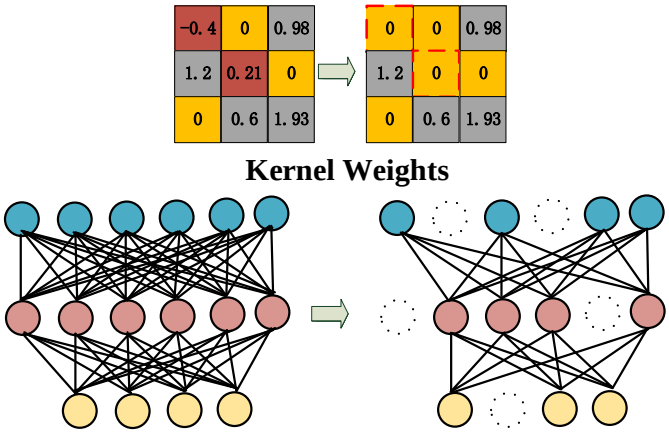

Fig. 4. Non-structure Pruning

Deep Compression uses three steps, *prunning*, *quantization*, and *Huffman Encoding*, to compress over-parameterized DNN models, such as VGG [77], AlexNet [1]. The pruning step removes the weights with magnitude lower than a threshold and corresponding connections. Then, quantization reduces the bit-width of weights to reduce the model size (More details about quantization in the subsequent section). Finally, Huffman encoding further compresses the weight storage. Experimental results show deep compression can significantly reduce the model size with no or negligible accuracy loss.

Inspired by the success of deep compression, many new methods are proposed to further improve the efficacy of *non-structure pruning*. Molchanov *et al.* [78] extend variational drop-rate to each weight of a DNN model, and weights with high drop-rate are deemed irrelevant and thus can be removed for model compression. Different from above-mentioned approaches which conduct *non-structure pruning* directly on pre-trained networks, NeST [79], that is inspired by the development of human brain, adopts a grow-prune scheme, where NeST first makes a sparse seed DNN model bigger and more complex and then prunes some irrelevant weights from the grown model to generate the final compact model. Zhang *et al.* [80] formulate the weight-pruning problem as a non-convex problem which can be solved by *alternating direction method of multipliers* (ADMM) method [81].

All methods discussed above target to reduce the model size via *non-structure pruning*. However, for edge devices, power and energy are also important metrics to consider. Yang *et al.*

TABLE III
COMPARISON OF NON-STRUCTURE PRUNING METHODS

Methods	Pruning method	Metrics	year
Deep Compression [76]	Threshold/Huffman Coding	Model size	2015
Molchanov *et al.* [78]	Variational Drop Rate	Model size	2017
Yang *et al.* [82]	heuristic	Energy	2017
Zhang *et al.* [80]	ADMM	Model size	2018
NeST *et al.* [79]	Grow-prune	Model size	2019

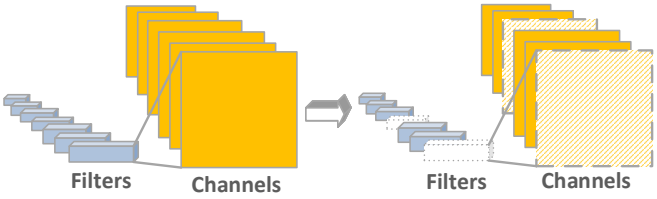

Fig. 5. A simple visualization of filter pruning

in [82] propose an energy-aware pruning method, in which they strive to reduce the energy consumption of DNN models by *non-structure pruning*. The core idea behind their approach is to order layers according to their energy consumption and then it prunes weights according to that order.

Although *non-structure pruning* can significantly reduce memory footprint and multiply-accumulate (MACs) of DNNs, *such reduction does not directly translate to latency improvement*. This is because *non-structured pruning* generates sparse structures which lead to irregular access pattern. The irregular pattern of sparsified DNN models needs special formats, e.g., compressed sparse row and compressed sparse column, to store sparse matrices. The off-the-shelf hardware and software cannot efficiently execute those compressed formats, so specialized hardware and software libraries are required to execute sparsified DNN models [83], [84].

2) Structure Pruning: Structure pruning, on the other hand, prunes network by maintaining its regular pattern. To keep regularity, *structure pruning* completely removes some channels and filters, that have least impact on the model's prediction as shown in Fig. 5. Since *structured pruning* does not lead to irregular pattern, the compressed network pruned by *structure pruning* can directly accelerate its inference on off-the-shelf hardware platforms without specialized software library support. Therefore, it has been receiving growing attention in recent years.

The common process of *structure pruning* is (1) *defining a pruning criterion*; (2) *selecting pruned channels according to the criterion and goal*, such as compression ratio and the number of MACs or FLOPs; and (3) *fine-tuning the pruned model*, i.e., retraining the pruned network to retain accuracy. Works in *structure pruning* define different criteria and adopt different methods to select the pruned channels, while minimizing accuracy loss. In terms of pruning methods, we can classify them into two categories: *training-based* vs *inference-based*.

- **Training-based**: The pruning is conducted during the training procedure, where a sparsity constraint is exposed and a compact network is directly learned from a big and over-parameterized network;

TABLE IV
THE CLASSIFICATION OF STRUCTURE PRUNING METHODS.

Methodologies	Pruning methods	Scope	Process	Year
Li *et al.* [87]	Inference-based	Layer	Rule-based	2016
Hu *et al.* [88]	Inference-based	Global	Rule-based	2016
SSL [89]	Training-based	Layer	Rule-based	2016
He *et al.* [90]	Inference-based	Layer	Rule-based	2017
ThiNet [91]	Inference-based	Layer	Rule-based	2017
DeepIoT [92]	Training-based	Layer	Learing-based	2017
DCP *et al.* [93]	Inference-based	Layer	Rule-based	2018
SFP [94]	Training-based	Layer	Rule-based	2018
Huang *et al.* [95]	Training-based	Global	Rule-based	2018
AMC [96]	Inference-based	Layer	Learning-based	2018
NetAdapt [47]	Inference-based	Layer	Learning-based	2018
Gate Decorator [97]	Inference-based	Global	Rule-based	2019
LFPC [98]	Inference-based	Layer	Learning-based	2020
HRank [99]	Inference-based	Layer	Rule-based	2020

- **Inference-based**: The prune method reduces the redundant channels from a pre-trained model according to defined rules;

In terms of pruning scope, we have *layer pruning* vs *global pruning*.

- **Layer pruning**: The pruning process is applied to the network layer by layer for finding the pruned network satisfying a defined target;
- **Global pruning**: The pruning process is applied to the whole network for finding the best pruned network while satisfying a defined target;

The pruning process is either *rule-based* or *learning-based*.

- **Rule-based**: The pruning is conducted according to some defined rules, like heuristic algorithms;
- **Learning-based**: The pruning is conducted by a learning algorithm, such as reinforcement learning [85], evolutionary algorithms [86] and gradient-based optimization.

Li *et al.* [87] adopt a global pruning method where all filters are sorted in terms of absolute weight sum and then filters with low magnitude are pruned and related channels are all removed. He *et al.* [90] present a layer pruning method using LASSO regression and reconstruction error to select pruning channels. Hu *et al.* [88] observe that a fraction of activation weights in DNNs are zero and these zero weights imply the corresponding filters are likely to be redundant and can be pruned, and they thus propose using Average Percentage of Zeros (APoZ) of a filter as the criteria to select pruned channels. Instead of using the information from the currently pruned layer for selecting pruned channels, ThiNet [91] proposes to exploit the information from the output of the next layer to determine pruned filters. Discriminate-aware channel pruning (DCP) in [93] relies on the discrimination of each filter to select the pruned channels. The main concept is that the discriminated channels provides more relevant information or features to retain accuracy and then the channels which are inadequately discriminated can be pruned for complexity reduction. You *et al.* [97] propose *gate decorator* module to replace the batch normalization module in DNNs to select pruned filters globally. Most *structure pruning* methods adopt a uniform pruning criterion for all layers, but different layers have different functions, thereby likely benefiting from employing different pruning criteria at different layers. Recently, He *et al.* [98] propose LFPC to learn an optimal pruning

criterion for each layer by using a gradient-based method. HRank [99] empirically finds that filters with low-rank is less informative than those with high-rank, and thus uses this observation to prune the unimportant channels.

Wen *et al.* [89] propose SSL to have a training-based pruning method, learning a compact and sparse model from a pre-trained model. Liu *et al.* [100] propose an approach called network slimming, which takes wide and large networks as input models, but during training, insignificant channels are automatically identified and pruned afterwards. Soft filter pruning (SFP) [94] deploy a training-based pruning method to prune a complex network. Huang *et al.* [95] propose the concept of sparsity scaling factor for each filter which is learned during to training, and then filters with scaling factor 0 are removed. Yao *et al.* [92] train an recurrent neural network (RNN) to determine the dropout probability of each filters and prune the network layer by layer according to drop probability.

Most of the above-mentioned *structure pruning* methods are rule-based, i.e., some heuristic algorithms devised according to their own criterion. In contrary, some work employ learning algorithm to automatically prune network model. AMC [96] proposes to use *reinforcement learning* to automatically prune channels of each layer. Anwar *et al.* [101] prune a DNN model at feature level, kernel level, and intra-kernel level (i.e., weight), where they deploy *evolutionary algorithm* to find the best combination of different pruning granularities. However, searching in a discrete space using RL and EA is really costly, so the gradient-based method is recently proposed to find an optimal pruning criterion for each layer [98].

To determine how many channels should be pruned, the above-mentioned works use indirect metrics like FLOPs or compression ratio to prune network. Nevertheless, the reduced FLOPs and compression ratio cannot directly translate to the performance improvement. In addition, a diverse of hardware accelerators have emerged for boosting the execution of DNNs, but these various systems demonstrate different capability to handling network complication. Hence, some pruning studies directly target the direct metric upon a specific hardware, e.g., latency. NetAdapt [47] proposes an automated framework to prune filters in different layers such that the pruned model can be adapted to a target platform. To optimize the latency on a target platform, NetAdapt builds up a look-up-table (LUT) for different operations and layers, so instead of measuring latency on the real platform, it can quickly estimate the latency based on the model architecture and LUT. Yu *et al.* [102] introduce SIMD-aware pruning framework which employs different pruning strategies for different underlying hardware, like weight pruning for low-parallelism CPU and filter pruning for high-parallelism.

Discussion: Table III and IV summarize methods discussed in this section. Pruning is the well-studied topic in model compression and many methods have been proposed to advance the network pruning [103]. These pruning methods show promising results, capable of compressing a DNN model, in some cases more than 40x, while only degrading the prediction accuracy slightly [76]. To effectively adopt model pruning for edge intelligence systems, the existing works suffer from two flaws: 1) the theoretical foundation behind pruning lacks, and thus there is a debate whether we should prune a complex model or directly train a compact model for resource constrained hardware platforms. Some recent research [104] [105] [106] [107] strives to empirically or theoretically find an answer for this question. Since pruning is a time-consuming procedure involving large model training and iterative pruning-retraining procedure, a theoretical foundation or proof may drastically change the way we use model pruning to design edge DNN models; 2) Almost all of pruning methods are *hardware-agnostic* and depend on *hardware-independent* metrics, like MACs and FLOPs. Since different hardware architectures demonstrate different degrees of parallelism, the state-of-the-art methods may unnecessarily prune models without performance improvement (e.g., latency reduction) while reducing the capacity of DNN models which is proven to have a significant impact on model's accuracy [108]. Therefore, we need to devise pruning method with hardware-awareness such that the model can be tailored for various hardware.

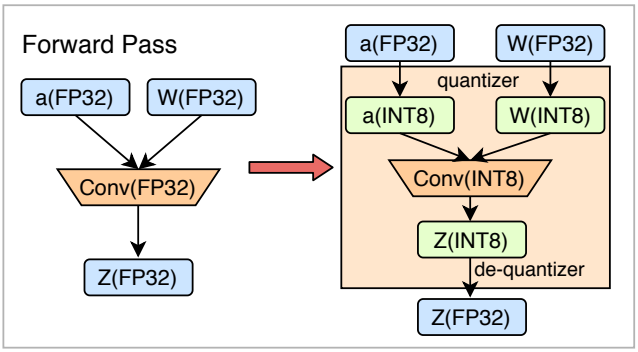

Fig. 6. An example of convolution quantization.

TABLE V
THE CLASSIFICATION OF QUANTIZATION METHODS.

Methodologies	Bits	Weights	Activations	Year
Gupta *et al.* [109]	16-bit	Yes	No	2015
Q-CNN [110]	1-bit	Yes	No	2016
BinaryConnect [111]	1-bit	Yes	No	2016
Ternary [112]	2-bit	Yes	No	2016
DoReFa-Net [113]	Arbitrary	Yes	Yes	2016
XNOR-Net [114]	1-bit	Yes	Yes	2016
INQ [115]	5/4-bit†	Yes	Yes	2017
ABC-Net [116]	1-bit	Yes	Yes	2017
Zhu *et al.* [117]	2-bit	Yes	No	2017
Jacod [118]	8-bit	Yes	Yes	2018
Bi-Real-Net [119]	1-bit	Yes	Yes	2018
LQ-Net [120]	Arbitrary	Yes	Yes	2018
TQT [121]	8-bit	Yes	Yes	2019
HAQ [122]	(1∼8)-bit§	Yes	Yes	2019
HAWQ [123]	2/4-bit†	Yes	Yes	2019
Jung *et al.* [124]	(2∼4)-bit	Yes	Yes	2019
Zhuang *et al.* [125]	2-bit	Yes	Yes	2020
XOR-Net [126]	2-bit	Yes	Yes	2020

a/*b*-bit † denotes the method quantizes the network weights for a bits while operating intermediate activation feature maps for b bits.
($a \sim b$)-bit§ represents the method quantizes the network weights or activations from a bits to b bits.

B. Quantization

Network pruning reduces the complexity of DNN models by removing redundant weights or channels. However, the

state-of-the-art models have more than billions of parameters and at the same time during inference a model produces a large portion of intermediate results (activation/feature maps) which usually occupy a large memory space. As a result, the huge memory requirement prohibits DNN models from implementing on memory-limited edge devices [127], [128]. For example, ResNet-50 [7] has 26 million parameter weights, generates 16 million activations in one inference, requires around 168 MB memory space, and needs at lease 3GB/s memory bandwidth. It is not difficult to see that it is unlikely to deploy these state-of-the-art models to edge devices, which have limited storage and computational resources.

In this case, *quantization* becomes a promising approach to address the aforementioned issue, which encodes full-precision (FP32) weights and activations with low-precision ones (e.g., FP16, INT8, binary) while preserving even the same level of accuracy. Some early work has shown that using FP16 to train DNN models can reduce the computational cost while retaining accuracy [109]. Quantization significantly benefits DNN models on resource-limited devices, and it is capable of fitting the whole model into on-chip memory of edge devices such that the high overhead occurred by off-chip memory access can be mitigated. In addition, since operations with low-bit representation usually consume less energy and execute faster, quantization reduces energy consumption and latency as well on some hardware platforms [129]. In this section, we discuss some state-of-the-art quantization studies.

Wu *et al.* [110] propose a unified quantization framework, which improves the quantization performance by minimizing the estimation error of each layer's response. On this basis, an error correction training strategy is included. Jain *et al.* [121] propose a trained uniform quantization method for accurate and efficient neural network inference on fixed-point hardware. Jacob *et al.* [118] quantize models into 8-bit integer and propose a quantize-aware training method to eliminate the accuracy loss caused by quantization. Instead of quantizing all weights in a DNN, IQN [115] adopts a group-wise quantization method to gradually quantize weights of a DNN model. The advantage of IQN is that it is able to derive the quantized model without accuracy loss.

The above works uniformly convert all weights and activation into the same low-bit representation, but many emergent hardware and accelerators support mixed precision operations, e.g., Nvidia Turing GPU architecture supports 1-bit, 4-bit, 8-bit and 16-bit arithmetic operations [130]. This provides a more effective and flexible way to quantize weights and activations of a DNN model. Wang *et al.* [122] propose a hardware-aware automated quantization approach, namely HAQ. HAQ exploits reinforcement learning to select different quantization width for each layer upon a target hardware. Additionally, the hardware architecture is involved into the learning loop, so that it can directly reduce the inference latency, energy and storage on the target hardware. HAWQ [123] also considers to quantize a full-precision model into a mix-precision model where the quantized precision is determined for each layer based on Hessian matrix.

Some works use binary or ternary quantization to maximally compress DNN models. Then, binarized or ternarized network can use cheap bit operation to boost the efficiency on dedicated hardware [131]. In [111], Courbariaux *et al.* propose the binaryconnect, which targets to transform the full-precision weights into the binary format. A very straightforward binarization method would be based on the sign function:

$$w_b = \begin{cases} +1 & \text{if } w \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

where w_b is the binary weight and w is the full-precision weight. Due to the fact that this is a deterministic operation, averaging the discretization over the many input weights of a hidden unit could compensate for the loss of information. An alternative that allows a finer and more correct averaging process to take place is to binarize stochatically, which helps to improve the model generalization capability:

$$w_b = \begin{cases} +1 & \text{with probability } p = \delta(w) \\ -1 & \text{with probability } 1 - p \end{cases} \tag{2}$$

where δ is the *hard sigmoid* formula:

$$\delta(x) = \text{clip}(\frac{x+1}{2}, 0, 1) = \max(0, \min(1, \frac{x+1}{2})) \tag{3}$$

Intuitively, applying a binarized method is an easy way to quantize the full precision weights. However, this will be harder for the training process to converge due to the highly discrete parameter space, which drastically degrades the model performance.

Later, Rastegari *et al.* [114] present an efficient approximation strategy, which constrains the full precision weights to $+1$ and -1 instead of directly rounding them. With a scaling factor α, an convolutional operation can be approximated as follows:

$$\mathcal{I} * \mathcal{W} \approx \mathcal{I} * (\alpha \mathcal{B}) = \alpha \mathcal{I} * \mathcal{B} \tag{4}$$

where $\mathcal{I}$ is the input image and $\mathcal{B} \in \{-1, +1\}$ is the binarized weight. Other binarized methods are proposed to improve the degraded accuracy caused by binarization like [113], [116], [119]. XOR-Net [126] takes the implementation of BNNs into account and simplifies the number of instructions used in binary dot product to accelerate BNN inference on edge devices.

Inspired by [114], Li *et al.* [112] propose ternary weight networks (TWNs) with weights constrained to $\{-1, 0, +1\}$, which minimizes the Euclidian distance between full precision weights $\mathcal{W}$ and the ternary weights $\mathcal{W}^t$ along with a scaling factor α. Here the quantized weights are obtained with a threshold-based ternary function:

$$\mathcal{W}_i^t = \begin{cases} +1, & \text{if } \mathcal{W}_i > \Delta \\ +0, & \text{if } |\mathcal{W}_i| \leq \Delta \\ +0, & \text{if } \mathcal{W}_i < -\Delta \end{cases} \tag{5}$$

where Δ is a positive threshold parameter. Thus, the optimization objective can be formulated as follows:

$$\alpha^*, \mathcal{W}^{t*} = \arg\min J(\alpha, \mathcal{W}^t) = ||\mathcal{W} - \alpha \mathcal{W}^t||_2^2 \tag{6}$$

By addressing the above convex optimization problem, we can obtain the approximately optimal α^* and Δ^*:

$$\begin{cases} \alpha_\Delta^* = \frac{1}{I_\Delta} \sum_{i \in I_\Delta} |W_i|, \\ \Delta^* = \arg\max \frac{1}{|I_\Delta|} (\sum_{i \in I_\Delta}) |W_i|^2 \end{cases} \tag{7}$$

where I_Δ denotes the number of elements in $\{i||\mathcal{W}_i| > \Delta\}$. On top of TWNs, Zhu *et al.* [117] further introduce two independent quantization scaling factors for positive and negative weights in each layer, respectively, to improve accuracy of ternary quantization.

Quantization are usually achieved via post-training quantization which is regarded as the most prevalent method currently. The mainstream DNN frameworks like Pytorch [132] and Tensorflow [133] all support post-training quantization. Fig. 6 illustrates the procedure of post-training quantization. First, we should find a proper encoding algorithm, which quantize both full-precision (e.g., FP32) activation results a and networks weights w into the low-precision (e.g., INT8). Then, the operation (e.g., Convolution) is performed as usual, where the derived outputs will be further relaxed to the full-precision format which can be conducted together with the scaling factor (e.g., α in Eq. 4) with respect the quantizer. Post-training quantization is flexible and can be broadly applied to existing DNN models. However, since the weights and activations are quantized into discrete values, we cannot use stochastic gradient descent to update the weights. Consequently, the quantized models may suffer from significant accuracy loss. Therefore, some studies [120], [124], [125] aim to effectively train a low-precision, compact model. Zhang *et al.* [125] adopt an auxiliary full-precision model to facilitate the training of its quantized counterparts.

Recently, the robustness of DNN models, i.e., robust to adversarial examples, is a burgeoning topic in DNN research [134]. For edge systems, robustness is a critical metric especially for some safety-critical systems, like self-driving cars, UAVs, robots, etc. The failure or mispredication may lead to catastrophic results. Recently, Lin *et al.* [135] study the effect of quantization on the DNN robustness and propose the Defensive Quantization (DQ) that addresses the robust issue of quantized models, where DQ can maintain adversarial robustness and model performance at the same time. Moreover, Gong *et al.* [136] consider to quantize network with the objective of reducing energy consumption.

Discussion: Table V summarizes the quantization approaches discussed in this section. Quantization has become a standard means to compress memory-hungry DNN models for resource-constrained edge systems. Many vendors develop their own tools and hardware to effectively quantize models and support efficient execution of quantized models, like Nvidia TensorRT[4], Tensorflow Lite[5], OpenVino[6], etc, and some real-world applications based on quantized models are emerging such as [137]. For edge intelligence systems, we have seen a new trend that quantization will work with other model compression techniques as well as the new NAS methods to derive the most efficient and compact models, for example, AQP [138] which combines NAS, quantization, and pruning to design efficient DNNs. Such method will become a new standard to design edge DNN models.

[4]https://developer.nvidia.com/tensorrt

[5]https://www.tensorflow.org/lite

[6]https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html

TABLE VI
THE CLASSIFICATION OF KNOWLEDGE DISTILLATION METHODS

Methodologies	Distillation Transfer	Number of Teachers	Year
Remero *et al.* [139]	From intermediates	Single	2014
Hinton *et al.* [140]	From logits†	Single	2015
Zagoruyko *et al.* [141]	From intermediates	Single	2016
Tarvainen *et al.* [142]	From logits	Multiple	2017
Polino *et al.* [143]	From logits	Single	2018
Ravi *et al.* [144]	From logits	Single	2019
Li *et al.* [145]	From logits	Single	2020
Chung *et al.* [146]	From intermediates	Single	2020
Liu *et al.* [147]	From intermediates	Multiple	2020

† Following [140], we use logits to denote the knowledge from the output of the neural network, *i.e.*, from the last layer.

C. Knowledge Distillation

Knowledge distillation is another technique to conduct model compression, where a more compact student model can learn the knowledge from a complicated and powerful teacher model. Bucila *et al.* [148] first propose the concept of knowledge distillation, and Hinton *et al.* [140] generalize knowledge distillation and apply it to DNNs.

The core idea of knowledge distillation is to train a compact model (student) with the assistant of a complicated, pre-trained model (teacher). During training, the student model exploit the conventional method to train the model and obtain a loss according to the one-hot class distribution, e.g., $[0, 0, 1, 0]$, namely *hard targets* and at the same time the knowledge from the teacher model is distilled and transferred to the student model by calculating a new loss in which the target is the probability distribution of predicted class $\mathcal{P}$ from the teacher model, e.g., $[0.1, 0.21, 0.6, 0.09]$, namely *soft target*. Nevertheless, the probability of the correct class dominates the probability distribution generated by the teacher network (e.g., $[0.97, 0.1, 0.0, 0.2]$), which significantly limits the knowledge transferring capability. To alleviate this issue, Hinton *et al.* [140] propose *softmax temperature* in which temperature T is to soften the generated probability distribution. Intuitively, a larger T leads to a 'softer' probability distribution (e.g., $[0.4, 0.2, 0.2, 0.2]$). Hence, we are able to formulate the softmax with temperature as follows:

$$\mathcal{P} = \{p_i | \frac{\exp(\frac{z_i}{T})}{\sum_j \exp(\frac{z_j}{T})}\} \tag{8}$$

when T is set to 1, it is the original softmax function. Please note that we refer the softmax function with temperature T as δ_T for simplicity. Therefore, we can formulate the overall loss function as:

$$\mathcal{L}(x; W) = \alpha * \mathcal{F}(y, \delta_1(z_s)) + \beta * \mathcal{F}(\delta_T(z_t), \delta_T(z_s)) \tag{9}$$

where $\mathcal{F}$ denotes the cross-entropy function. α and β are two balancing factors. z_s and z_t represent the output logits from student model and teacher model, respectively. y is the ground

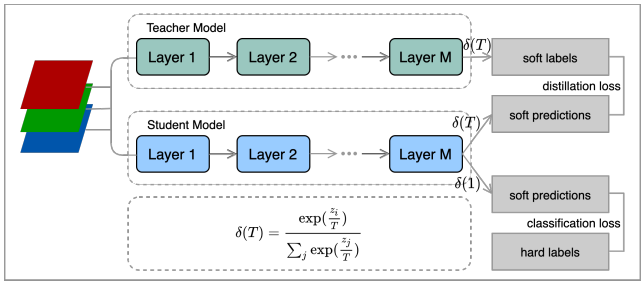

Fig. 7. Overview of Knowledge Distill.

truth. An illustration about how knowledge distillation works is shown in Fig. 7.

After [140], many efforts are made towards improving the performance of knowledge distillation. The work in *et al.* [142], [147] extend the number of teacher models from one to multiple. However, there exist performance difference among those teacher models. To tackle this issue, they propose to assign different weights for each teacher models, and then weighted-average probability distributions from different teachers are applied to supervise the student model. Combined with quantization, Polino *et al.* [143] introduce the *quantized distillation*, which leverages distillation during the training process by incorporating knowledge distillation loss. Ravi [144] introduces a neural projection approach to design and train efficient on-device neural networks. Preceding to the prediction, input instances are transformed into binary representations, which significantly reduces the memory consumption. Afterwards, the prediction weights are learned by knowledge distillation to achieve higher generalization capability. In [145], Li *et al.* propose to use knowledge distillation to efficiently compress models, where the uncompressed model and compressed model are considered as a teacher-student pair. This new method can avoid the time-consuming fine-tuning after pruning and achieve data efficiency.

The above works only use the knowledge from the outputs of of the last layer in the teacher model. *Can the intermediate knowledge help to obtain a better model?* Remero *et al.* [139] adopt knowledge distillation to train a compact model, namely FitNets. The main idea in FitNets is to train a deeper and thinner student with the knowledge transferred from the shallower and wider teacher model. Different from the previous works, the knowledge in FitNets is not only from the final outputs but also from intermediate feature representations of the teacher model. By doing so, the student model in FitNets mimics or imitates the teacher model from different granularity levels. Similarly, Zagoruyko *et al.* [141] introduce the attention transfer strategy to mimic the attention maps of a powerful teacher network, which proves to improve the performance of the student network.

Discussion: Table VI summarizes the works discussed in this section. As identified in literature [149], knowledge distillation provides several benefits for small network models. *Accuracy:* distilling knowledge from large networks can improve the accuracy of small models which may be directly applicable to edge systems. *Transferability [150]:* Knowledge distillation demonstrates better transferability for the small models, i.e., for a given dataset, learning the small model from knowledge distillation outperforms learning it from scratch. This feature is really instrumental to design lightweight models for edge systems, because for some edge systems installed in special contexts, there lacks a huge amount of high-quality data to train an accurate model. Thus, knowledge distillation facilitates training of a competitive, compact model with small dataset. Moreover, knowledge distillation can also be used to design robust networks [151]. Knowledge distillation shows several benefits for edge systems and we envision knowledge distillation will be gradually integrated with other techniques like NAS to derive accurate, compact networks.

V. Hardware-Aware Neural Architecture Search

Model compression provides a '*large-to-small*' method to generate an efficient and compact model from a complex model for edge devices. Although it has shown its success in reducing latency and model size, the accuracy of the pruned model is inherently upper bounded by the pre-trained model, i.e., the compressed model cannot have better accuracy than the pre-trained model. Moreover, network compression involves a costly and time-consuming compression-retraining procedure to retain the accuracy of the pruned model. At the same time, as we are witnessing the shift of DNN designs from manual design to automatic search, i.e., neural architecture search, which has demonstrated its capability to design more accurate DNNs without tedious parameter tuning, this immediately raises an appealing question:

> *Can NAS directly design hardware-efficient, accurate neural architectures?*

An increasing attention is paid to hardware-aware NAS and some works exploit NAS to design hardware-efficient DNNs. Tan *et al.* [53] propose a hardware-aware NAS framework, dubbed MnasNet, in which both latency and accuracy are formulated into the reward function of reinforcement learning algorithm and the latency is directly measured from the target mobile device. The reward is shown in Eq. (10).

$$\text{ACC}(m) \times \left[\frac{\text{LAT(m)}}{T}\right]^{w} \tag{10}$$

where m is the obtained network, T is the expected latency and w is a variable to adjust the weight of latency in this reward. ACC and LAT are the real accuracy and latency of network m, respectively. MnasFPN [158] is an extension of MnasNet which, instead of searching classification model, targets directly searching a network for object detection. Dai *et al.* [165] present ChamNet which deploys evolution algorithm (EA) and three predictors, i.e., energy predictor, accuracy predictor and latency predictors, to effectively and efficiently search for a DNN model on a target platform.

The RL-based or EA-based NAS frameworks explore the optimal architecture in a large, discrete search space, so they demand huge amount of computational resource and take thousands of GPU days [166] to search for a neural architecture.

TABLE VII
COMPARISON OF DIFFERENT HARDWARE-AWARE NAS APPROACHES. HERE, COST REFERS TO THE TIME SPENT TO SEARCH THE NETWORK, WHILE WE USE *GPU Hour* AS UNIT.

	Hardware Evaluation	Search method	Cost (GPU days)	Metrics	Target	Edge Systems
MnasNet [53]	Measure	RL	40000	Latency	Image Classification	Yes
ProxylessNAS [152]	Predictor	gradient	200	Latency	Image Classification	Yes
EdgeNAS [153]	Predictor	gradient	-	Latency	Image Classification	Yes
SPNAS [154]	LUT	gradient	0.2 †	Latency	Image Classification	Yes
FBNet [155]	LUT	gradient	-	Latency	Image Classification	Yes
DenseNAS [156]	LUT	gradient	3	Latency	Image Classification	Yes
NAS-FPN [157]	-	RL	-	-	Object Detection	Yes*
MnasFPN [158]	Measure	RL	-	Latency	Object Detection	Yes
NAS-FCOS [159]	-	RL	-	-	Object Detection	No
Auto-FPN [160]	-	gradient	-	-	Object Detection	No**
AdversarialNAS [161]	-	gradient	24	-	GAN	No
SpArSe [162]	-	gradient	24	Memory	GAN	Yes
OFA [163]	Measure***	-	-	Mem/Lat	Image Classification	Yes
MCUNet [164]	Measure***	-	-	Mem/Lat	Image Classifica tion	Yes

† This low search cost is due to that SPNAS only searches for 8 epochs on a subset of ImageNet.
*Although NAS-FPN is a hardware-agnostic approach, NAS-FPN devises a lite version for resource constrained systems.
** Auto-FPN incorporate hardware-agnostic resource constraint like FLOPs, which cannot translate the runtime complexity upon target edge systems.
*** These two methods train an over-parameterized and large network which is used to sample different small networks, and the over-parameterized network is designed without consideration of hardware. However, sampled networks can be directly measured on devices.

As estimated in [152], MnasNet needs 40,000 GPU hours to search for the optimal network. It hinders users with limited resource to search for a DNN upon their target hardware devices. Thus, some studies aim to reduce the search cost for hardware-efficient NAS. ProxylessNAS [152] uses a gradient-based method to design hardware-aware neural architectures. Instead of measuring real latency on the target platform, it presents a latency-predictor to facilitate exploration of the optimal configuration for neural architectures. It greatly reduces the search cost to 200 GPU hrs while finding efficient models with competitive accuracy. Li *et al.* [167] consider both latency and accuracy as their NAS objectives and use the concept of *partial order pruning* to reduce the search space such that it can significantly reduce the searching time and achieve a good trade-off between accuracy and latency. RCNAS [168] formulates the resource-constrained neural architecture search problem as a submodularity function problem which is known to be NP-Hard but has good heuristic algorithms to approximately solve this problem. FBNet *et al.* [155] introduces DNAS, which is based on the differential neural architecture search (DARTS) method [166], but instead of just searching for an optimal cell in DARTS, DNAS searches for an optimal setting for each layer within the network. DNAS also takes latency as their goal, where a look-up-table is set up for latency prediction. EdgeNAS [153] proposes a novel NAS method to search efficient and competitive DNN model for resource constrained edge devices, where a latency predictor is trained from data collected from various architectures on target hardware devices and the latency predictor is integrated into the DARTS-similar NAS framework for efficiently designing DNN models. Besides, SPNAS [154] introduces a single-path paradigm, which formulates kernels with different sizes into one *big* kernel. Similar to DNAS, SPNAS constructs a LUT in terms of the runtime latency of different kernels upon target hardware, which will be incorporated into the differentiable loss function (similar to DNAS and EdgeNAS). All of the above-mentioned approaches mainly search for the best operators for cells or layers where their width and depth are fine-tuned manually. However, as shown in [50], the width and depth of a DNN have a critical impact on its accuracy and latency. Fang *et al.* [156] propose DenseNAS which not only searches for the optimal architectures but also their width and depth configuration. The successful application of NAS on image classification inspires researchers to explore the potential application of NAS on other CV tasks, like NAS-FPN [157] and NAS-FCOS [159] for object detection and AdversarialNAS [161] for GAN. However, these methods only consider the accuracy and ignore the performance like latency and power consumption, which are critical for edge intelligence systems. Besides, Auto-FPN [160] aims to search for a compact FPN with low FLOPs count, but FLOPs cannot necessarily reflect the runtime performance upon target hardware (see Fig. 11). Similar to MnasNet [53], MnasFPN [158] directly measures the runtime latency on target hardware, thereby significantly increasing the search cost.

Besides latency, other metrics are also considered in hardware-aware NAS frameworks. SpArSe [162] targets networks which can be fit into micro-controllers which have small memory footprint and less computation capability, where NAS and pruning techniques are combined to design small-memory networks. Cai *et al.* [163] propose an *once-for-all* (OFA) framework to train a large super network and then sample different size of networks from the super network to fit the different hardware platforms. The advantage of OFA is that it just needs to train once to generate many different DNNs which can be directly applied on diverse hardware platforms, thus greatly reducing the training cost and CO_2 emission. Lin *et al.* [164] propose MCUNet aiming to design DNN models for microcontrollers. To fit computation-intensive DNN models on microcontollers, MCUNet consists of two key parts, TinyNAS, a NAS framework to search for models satisfying different constraints, such as memory, latency and TinyEngine, an efficient inference library.

Discussion: Table VII summarizes the works discussed in this

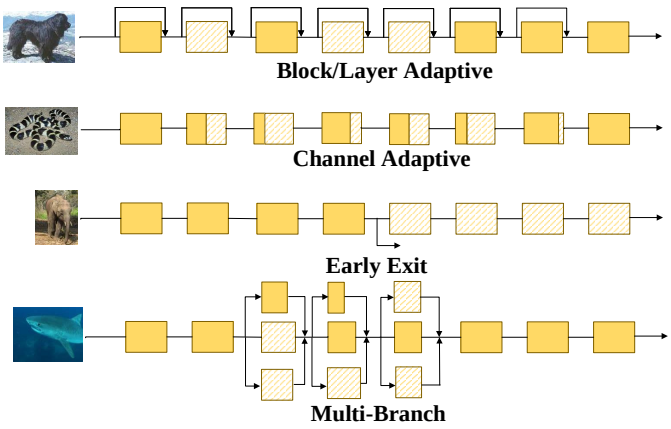

Fig. 8. Visualization of four different methods for adaptive models

section. NAS defines a new paradigm to design DNN models, frees DNN designers from tedious hyperparameter tuning and automates the costly DNN design procedure. Especially, in future edge era, more and more customized DNN models for specific tasks will be developed and implemented on different hardware platforms, e.g., edge systems with intermittent power supply [169]. Nevertheless, for edge intelligence systems, the existing NAS methods suffer from two flaws. First, the majority of existing methods use *platform-independent* metrics, i.e., FLOPs and MACs as the constraint to design the model. However, the same model demonstrates significant difference on different hardware [170]. Therefore, such designs may generate inefficient models for targeting hardware. Second, the existing NAS methods mainly target the models with high accuracy while ignoring other important system metrics, e.g., energy and latency. This leads to the searched model with low efficiency and high energy consumption. *For edge intelligence systems with diverse tasks and hardware architectures, we need hardware-aware NAS methods to efficiently tailor competitive DNN models for a specific hardware platform. Moreover, to achieve an optimal design for future edge intelligence systems, NAS may need to take advantage of other model compression techniques like network pruning, quantization, and knowledge distillation, to have a holistic and efficient design paradigm, e.g., [138].*

VI. Adaptive Models

In previous sections, we discuss the different ways to design lightweight DNN models for edge systems, all of which generate a static DNN model, i.e., the model does not change its execution behavior during run-time. However, during the run-time, DNN applications share the computational units and communication bandwidth with other applications and as a result the availability of computing and communication resources may significantly affect DNNs' inference time. In addition, some edge systems are energy-constrained, battery powered or supplied by sustainable energy, like solar energy (intermittent computing [171]), and in these cases energy variance will change the system's status, such as scaling down frequency and power which affects the inference time of DNN applications as well.

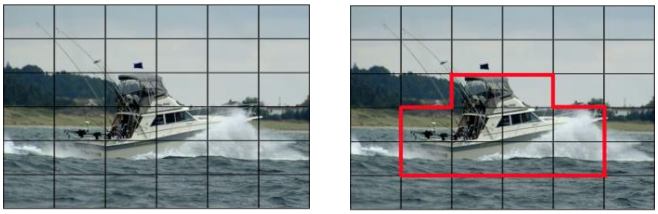

Fig. 9. Attention-based methods only conduct convolution operations on 'important' areas to reduce the computational cost, where the red line highlights the area of interest.

Such variance may influence the quality of service (QoS) of applications without rigorously temporal requirement, like voice recognition, face recognition, machine translation, etc [20] and on the other hand may lead to catastrophic consequence for those with rigorously temporal requirement, (i.e., real-time requirement), like autonomous driving, UAV, etc. Execution variance of DNN applications require DNN models to be adaptive to the different input data and system status (like low power mode) for guaranteeing certain QoS or real-time performance. Adaptive DNN models will be useful and practical for edge systems under dynamic environments and prior review articles rarely discuss this topic in a systematical way.

We, in this section, review some important techniques of adaptive DNN models which are applicable to edge systems. Some early works identify that for different input images, not all channels or layers of a DNN model are needed to make accurate prediction [172]. This key observation serves the technical foundation for adaptive models, i.e., a network may selectively activate its channels and layers per input basis. The partially activated network can achieve higher efficiency than the originally full network without accuracy loss. Some initial works on this topic are called *conditional computation* [172], [173], where the main objective of *conditional computation* is to enhance the capacity of a network while not significantly increasing the computational cost. However, for edge systems, the efficiency is our top priority, and we briefly classify adaptive models into five categories:

- **Block/Layer Adaptive**: This method selects a portion of blocks/layers to conduct DNN inference as shown in Fig 8;
- **Channel Adaptive**: This method deploys a portion of channels of each layer to conduct DNN inference as shown in Fig 8;
- **Early Exit**: This method uses the intermediate result from early layers to conduct prediction and skips the left layers in the network as shown in Fig 8;
- **Multi-branch**: This method has diverse kernels for extracting features and uses a combination of kernels to conduct DNN inference as shown in Fig 8;
- **Attention**: This method employs attention mechanism [174] to find the importantly spatial locations of images and only conduct computation-expensive convolution on these area as shown in Fig 9.

Runtime Network Pruning (RNP) [175], a channel adaptive

method, is conceptually similar to *structure pruning* discussed in Section IV-A. However, different from the methods in Section IV-A which aim to produce a *static* model at design time, RNP is a *runtime and dynamic* pruning method, which uses RL to learn a policy to determine which filters should be used according to the input data. Feature Boosting and Suppression (FBS) [176] exploits channel saliency to judge which channels can be skipped, where a low overhead predictor is presented to predict channel saliency according to feature maps of the previous layer. Bejnordi *et al.* [177] propose a channel-gate module to partially activate channels according to input. SlimmableNet [178], [179], also a channel adaptive method, proposes a method to train a network with different channel width configurations and the network can actively vary its channel configuration on the fly.

Some works exploit attention mechanism to improve computational efficiency. Attention technique mimics human visual systems to locate their focal point on the important area and up-weights the areas of interest to improve the model accuracy. Attention has shown its successful application in CV [174] and NLP [180]. From efficient perspective, instead of up-weighting the important areas, some works avoid the computation-expensive convolution on the less important areas to improve the efficiency of DNN inference. SBnet [181] divides image into blocks with fixed size, and only convolves with blocks of interest. SACT [182] exploits adaptive computation time (ACT) technique [183] to vary the computation per spatial location of input images. Verelst and Tuytelaars [184] use Gumble-softmax [185] to train a pixel-wise gate to identify the region of interest while Chen *et al.* [186] propose to combine attention with multi-branch technique to improve the capacity of lightweight models. CGNets [187] is conceptually similar to attention mechanism, where few feature maps are used to identify the regions of interest within intermediate activations and the convolutional computation is only applied to the regions of interest. CGNets also provide a hardware implementation for accelerating its design.

Some works observe that for simple input data, the model can use intermediate results from early layers to predict the class without affecting the accuracy. This is called *early-exit*. Adaptive Neural Network [188] learns a policy to determine whether given an input data the model can use intermediate results from early convolutional layers to make accurate prediction. Based on this strategy, they also extend their policy to select a model from a set of models with different accuracy/latency trade-off. BranchyNet [189] uses *early-exit* to skip some layers for easy input images, where a threshold is given to evaluate the confidence of predictions from early-exit.

Another common method to achieve adaptivity is *multi-branch* which is conceptually analogous to mixture of experts [190] [191], where each branch has some convolution kernels and the multi-branch network selects some branches/convolution kernels per input to conduct inference. HydraNet [192] presents a multi-branch network which can select the best k branches to conduct inference on per-input basis. Condconv [193] proposes a conditional convolutional layer which combines different convolution kernels based on the input. Dynamic deep neural networks (D^2NN) [194], a bit different from [192], [193], formalize a network as a directly acyclic graph (DAG) which has different operations for each node within DAG and the network selects an effective and efficient execution path from DAG according to its input. D^2NN is trained in an end-to-end manner with assistant of RL.

TABLE VIII
COMPARISON OF DIFFERENT ADAPTIVE DNN MODELS

Methods	Adaptive Scope	Training	Year
RNP [175]	Channels	RL	2017
Benjordi *et al.* [177]	Channels	Gradient	2019
SlimmableNet [178]	Channels	Gradient	2019
FBS [176]	Channels	Gradient	2018
CGNets [187]	Channels	Gradient	2019
AdaptiveNN [188]	Early-exit	Gradient	2017
BranchyNet [189]	Early-exit	Gradient	2017
D^2NN [194]	Multi-branch	RL	2018
Condconv [193]	Multi-branch	Gradient	2019
HydraNets [192]	Multi-branch	Gradient	2018
SkipNet [196]	Layers	RNN/RL	2018
BlockDrop [197]	Layers	RL	2018
SBnet [181]	Attention	Gradient	2018
Verelst and Tuytelaars [184]	Attention	Gradient	2020
Chen *et al.* [186]	Attention	Gradient	2020

ResNet [7] is found to be tolerant to the removal of some blocks or layers without affecting the predictive accuracy [195]. Therefore, some works investigate how to bypass blocks/layers of a ResNet-similar network. SkipNet [196] uses previous layer's activation to determine whether the subsequent layer is required for the inference, where RNN and RL are used to control the executed blocks . Similarly, BlockDrop [197] uses RL to train a policy network to determine the block configurations used for different input images. ConvNet-AIG [198] learns a gate module for each block of ResNets to select the inference blocks conditioned on input images.

Discussion: Table VIII summarizes the works discussed in this section. Adaptive models are attracting more attention from researchers due to the computation limit of resource constrained systems and highly dynamic environments. The advantages of adaptive models are twofold: 1) Adaptive models provide a flexible way to achieve trade-off between accuracy and efficiency on the fly; 2) for some lightweight models adaptive models can increase the capacity of models, thereby improving the accuracy of the model without increasing the computational cost (because it only needs to partially activates the network). The existing adaptive models mainly consider the dynamics of input images, i.e., fewer channels or layers for '*easy*' images and more channels or layers for '*complex*' images. However, on edge systems, some system dynamics, memory contention, cache miss, bandwidth unavailability, etc, also impact the execution of DNN application and this part is ignored in current research outcomes. *For edge intelligence systems, it is of importance to take into account both dynamics, generating an adaptive edge intelligence system which can guarantee a certain level of QoS or meet real-time requirement under dynamic environments.*

VII. DISCUSSION AND ENVISIONS

DNN-based AI applications are increasingly integrated into our life and work and will greatly revolutionize the way we

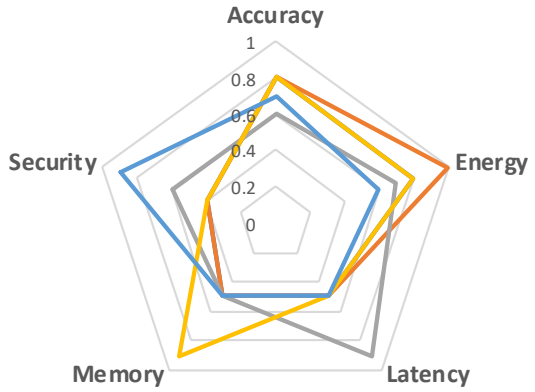

Fig. 10. The real design space for Edge intelligent systems. There are various scenarios which have different design concerns.

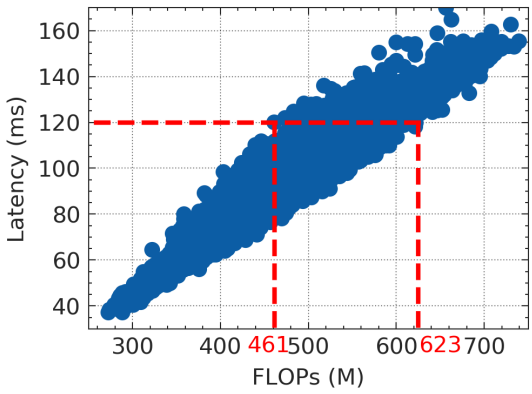

Fig. 11. Latency vs FLOPs: The latency measurements of 10000 different architectures with different FLOPs count. The experimental platform is Nvidia Jetson Xavier, a widely used edge platform for robots and autonomous driving.

live and work. Some AI applications rely on super computing power to complete complex tasks while others will operate in proximity to data and end-users to help us live 'smarter' and work 'intelligently'. Edge intelligence, deemed as one of the most important AI trends, will make the proximity AI possible and accessible. In the first DNN decade (2011-2020), researchers around the world have designed many compelling DNN models, applicable to various domains, like nature language processing [199], AI-assisted medicine [200], robots, etc. As reported in [201], over the years to come, the training/inference ratio of DNN models will increase to 1:5 from current 1:1 and enterprises will gradually add AI services to their core business so that they can profit from AI research and in turns AI applications can benefit the whole society. We can envision that the next important development for DNNs will be the practical deployment, especially like edge devices. To achieve this, we may need new DNN design methods, novel hardware [129] and the seamless cooperation between software and hardware.

In this section, based on our observations, we identify some important topics which we believe will play a pivotal role in pushing DNNs to edge.

A. *Different metrics oriented DNN models*

The accuracy improvement has been the highest priority in DNN research, where different models are justified by their accuracy increase, likewise a couple of percentiles. However when they are deployed on real systems, more metrics such as latency and power also matter. Moreover, DNN models are found to be vulnerable to adversary attack [202], so security is emerging as another new design concern for ML systems. Therefore, when designing DNN models for the emerging edge intelligent systems, the objective should be focused not only on accuracy but also on other critical metrics to have an overall consideration. More specifically, edge DNN models are really application-dependent or context-dependent, and it needs to find a good trade-off within the multi-dimension design space as shown in Fig. 10. As seen in Fig. 1, edge computing spans a wide spectrum. Some scenarios like UAVs, self-driving cars, robots, etc, have restrict requirements for accuracy, latency, and security, hence we may need a design point which is able to strike the balance within the design space. To do so, we need to correctly use metrics or define new combination metrics for edge DNN systems.

For example, DNN research mainly adopts FLOPs as the indicator of model complexity, while FLOPs are used as constraints for network design and compression. However, the number of FLOPs may not directly translate to its latency, because DNN models have diverse architectures which may demonstrate different effectiveness on different hardware platforms. Fig. 11 shows the latency of 100000 network architectures generated using DARTS [166] which are measured on Nvidia Jetson Xavier [203], where we can see that the models with the same latency may differ in the number of FLOPs by up to 26% (from 461 FLOPs to 623 FLOPs with latency 120ms) and the models with the same FLOPs perform different latency ranging from below 80ms up to 120ms. Therefore, we should carefully use the indirect metric to guide edge DNN design. In addition, design space for DNN is too large to have an exhaustive search. More design concerns will exacerbate this design complex issue. However, we still lack the measurement of the trade-off between each metric and this will lead to either high searching cost or sub-optimal design result. Thus, it will be necessary to define some combined metrics which can quantitatively evaluate the trade-off between different metrics, like energy-delay-product for conventional applications on CPU.

B. *Hardware-software co-design*

The high complexity of DNN models has spurred hardware architecture innovation to boost DNN training and inference over the past five years. The academia and industry have a consensus that the breakthroughs in hardware architecture will bring DNNs to a new level and boost DNN adoption [15] [205]. However, different hardware accelerators features diverse underlying characteristics, and the majority of DNNs were designed without consideration of underlying hardware features. We would like to call it *hardware-agnostic design*.

It is known that accuracy of DNN models are highly related to its width (channels or feature maps) and depth (layers or blocks) [50], DNN models designed without hardware

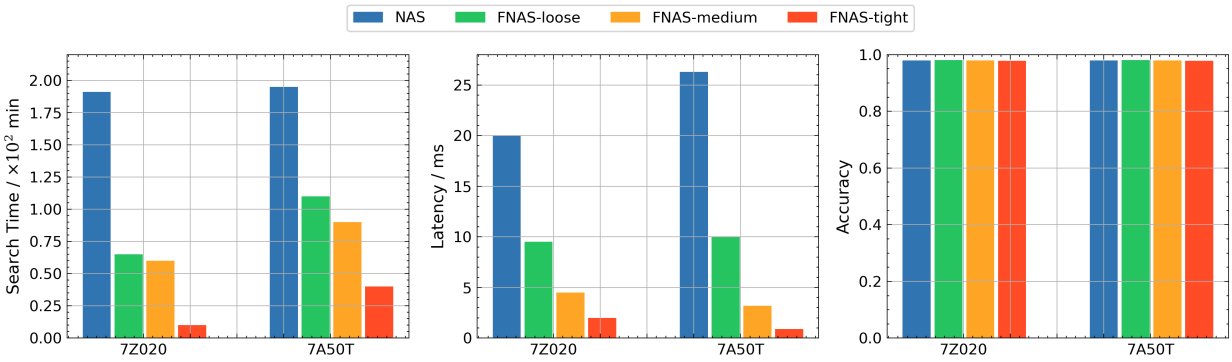

Fig. 12. Effectiveness of Hardware-Software Co-Design Paradigm as shown in FNAS [204]. FNAS-loose, FNAS-medium, and FNAS-tight denote three different design patterns in terms of the resource constraint.

consideration may not fully utilize the underlying hardware. Fig 13 shows the latency of four state-of-the-art DNN models, Inception v3 [206], ResNet50 [7], MnasNet [53] and MobileNet V3 [45] on three edge devices, *Intel Neural Computing Stick 2* [207], *Nvidia Jetson TX2* [203], *Google Edge TPU* [21] and one desktop GPU, *Nvidia Quadro GV100*. For MobileNet V3 on *Edge TPU*, we use optimized models provided by Google, which has more MACs than the original models (990M vs 210M). From the experimental results, we see that MnasNet and MobileNet V3 on *Edge TPU* perform very low latency, even lower than the high-end GPU, because these two models are specifically optimized for *Edge TPU*. These two models on *Edge TPU* also consume higher power (5W) than Inception V3 and ResNet50 (4.6W). Based on the observations, we conjecture that since these two models are designed and optimized for *Edge TPU*, they are able to better utilizes the parallelism of underlying hardware, thereby having lower latency and higher power consumption. In addition, as shown in [170], the same neural architecture demonstrates up to 62x difference on modern mobile devices in terms of inference time.

Some research strives to have a hardware-aware NAS, like FNAS [204], which directly incorporates resource constraints during implementing DNNs on FPGAs. The experimental results of FNAS are illustrated in Fig. 12. 7Z020 and 7A50T denote the low-end and high-end FPGAs, respectively. FNAS can design models according to different resource constraints, i.e., FNAS-loose, FNAS-medium, and FNAS-tight. Thank to its hardware-aware method, FNAS can achieve the same accuracy level under different resource constraints, while significantly reducing the search cost and inference latency.

These together signal the importance of *system-level hardware-software co-design*. The first decade of 21th century witnessed the emergence of system-level design methodologies [208] for multicore system design. To alleviate the increasing complexity of multicore systems, different software and hardware co-design methods were proposed to elevate the design level to system-level by modeling software and hardware so that some tedious and error prone procedures can be avoided. The history may repeat for the emergent edge intelligent systems. *The large and complex design space of edge intelligent systems need hardware-software co-design to facilitate the effective and efficient design and implementation of edge intelligent systems.*

To achieve co-design, we need to determine an effective design space for DNN models which will be helpful to reduce the costly design time. We have seen some recent efforts towards defining the effective design space for DNN models [209], [210]. At the same time, effective hardware modeling techniques are needed. We need metrics like Roofline [211] to guide the direction in finding efficient network upon a target platform and a standard benchmark which can fairly and quantitatively evaluate various DNN models on new hardware accelerators, like MLPerf [212] and ParaDNN [213]. In addition, to better utilize the underlying hardware, some DL compilers may need to be integrated into the co-design framework, like TVM [214] and patDNN [215].

C. Lightweight models for other applications

Currently, the majority of works regarding deep learning on edge systems target computer vision tasks, i.e., *image classification* and *object detection*. We have seen some successful CV-based edge intelligent systems, such as face recognition, object tracing on UAVs, navigation on robots, video analytic systems [216], etc. However, we also see the success of DNN models in other domain, like NLP, machine translation, etc. These models, like BERT [217], are known to be highly complex, even more complicated than DNN models for image classification, and a recent study in [9] raises a concern regarding the environment effect of training complex NLP models. As diverse applications will be increasingly implemented on edge systems, we need new methods or frameworks to design lightweight DNN models for domains other than computer vision. Like [218], an efficient point-voxel CNN is proposed for efficient 3D learning, and this model can help to implement 3D AR/VR applications and SLAM [219] of autonomous driving on edge systems. Li *et al.* [220] recently present a method to compress generative adversarial nets (GAN) [221]. By means of compressed GAN, some GAN-based applications, e.g., style transfer, image synthesis, etc, can be efficiently implemented on edge systems. Some recent works study to compress the complex NLP models such that they can be deployed on resource-constrained edge systems [33] [34] [35]. Only few efforts are made towards designing lightweight DNN models

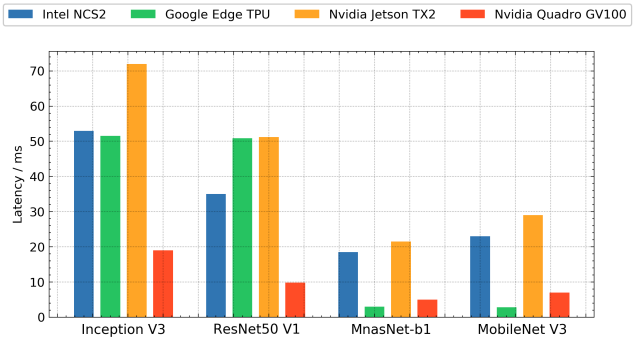

Fig. 13. Latency of four state-of-the-art DNN models on four devices.

of other domains for edge systems, but there is a huge potential to exploit such models on edge systems.

D. Learning on The Edge

In this paper, we review many techniques aiming to design lightweight models for edge intelligence systems, where the models are assumed to be trained on powerful servers but are deployed on edge devices for inference. However, due to high communication overhead, on-time model update and possible leakage of confidential data, some edge systems prefer to train the model locally, i.e., *on-device learning* [222]. On-device learning is a challenging task, because training is a more computation-intensive and memory-hungry procedure compared to inference, whereas edge systems are resource-constrained in many settings. Moreover, limited energy supply of some edge systems will make this issue even more difficult. Although few efforts have been made towards efficient learning at the edge, such as [223], [224], learning at the edge is still at its early stage. There still remain a lot of issues to be addressed in this topic. Some new methods, software frameworks and underlying libraries are needed to facilitate the effective and efficient training on edge systems with consideration of limitation and constraints imposed by edge systems, e.g., [128].

E. The data challenge at edge

The success of DNN models heavily relies on high-quality and large-scale datasets, such as ImageNet, but for some edge applications, e.g., edge surveillance systems in wild life, defect detection in manufacturing process, etc, it is difficult or expensive to collect massive amount of data and label them to train a good model. Thus, the majority of edge intelligence systems without large-scale dataset exploit *transfer learning* [225] to learn a competitive model, where a model trained with large-scale dataset is provided to extract features and then the classification layer (fully-connected layer) is further fine-tuned according to domain-specific dataset such that the model can be adapted for the new domain.

However, since, during the long-term operation, edge systems are likely to collect data with different distributions from the original training data or data pertaining to a new class which is not included in the original training data, we may need to update the model on edge systems in order to provide better prediction performance or infer a new class. On one hand, edge systems can update models locally by using *incremental learning* [226] [227], where techniques in [228] are deployed to improve the accuracy for new data and infer unknown classes. On the other hand, a group of edge systems is able to help each other to improve models by using *federated learning* [229]. As discussed several times in this paper, data privacy is one motivation of edge systems, so it may be impossible to collect data from different edge systems and share them with each other. *Federated learning* (FL) [229] is proposed to attack this issue, where instead of sharing the data with a centralized server clients (e.g., an edge system) in FL only share the learned gradient with others which will not leak data. Federated learning is a promising solution to share the information among several models or data sources while still keeping the confidential of data. Thus, FL can be used as a powerful tool to connect edge intelligence systems to improve their intelligent ability. Recently, few works study to employ incremental learning [227] and federated learning [230] [231] with edge systems, but the research in this context is still in its infancy. The breakthrough in this area will pave the way of ubiquitously adoption of edge intelligence systems in our life.

VIII. Conclusion

The convergence of edge computing and artificial intelligence leads to the concept of edge intelligence. Edge intelligence is in its early stage and needs sustainable efforts. This paper mainly surveys DL techniques which will facilitate the efficient deployment of DNN models on edge systems, i.e., lightweight models, network compression, hardware-aware NAS and adaptive models. We provide some of our thoughts about edge intelligence systems and hope this paper can help researchers from edge computing community to understand the state-of-the-art DL techniques and to explore new opportunities in edge intelligence era. As stated in [40], DL algorithms are approaching the computational limits of computing systems and this probably indicates that designing efficient DNN models will soon become a standard not only for edge systems but also all AI systems. Then, designing efficient DNN models will become a mainstream.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[8] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[9] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 3645–3650.

[10] V. Kepuska and G. Bohouta, "Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home)," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2018, pp. 99–103.

[11] S. Baruah, P. Lee, P. Sarathy, and M. Wolf, "Achieving resiliency and behavior assurance in autonomous navigation: An industry perspective," *Proceedings of the IEEE*, 2020.

[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[13] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[14] "Ieee computer society's top 12 technology trends for 2020," https://www.computer.org/press-room/2019-news/ieee-computer-societys-top-12-technology-trends-for-2020, access: 2020-5-14.

[15] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[16] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "Diannao family: energy-efficient hardware accelerators for machine learning," *Communications of the ACM*, vol. 59, no. 11, pp. 105–112, 2016.

[17] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition engine with sparse lstm on fpga," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 75–84.

[18] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "[dl] a survey of fpga-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 1, pp. 1–26, 2019.

[19] M. Feldman, "The era of general purpose computers is ending," *The Next Platform*, 2019.

[20] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia *et al.*, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 331–344.

[21] "Google edge tpu," https://cloud.google.com/edge-tpu/, accessed: 2020-06-25.

[22] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017. [Online]. Available: https://arxiv.org/abs/1611.01578

[23] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2020.

[24] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[25] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

[26] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[27] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[28] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[29] J. Dean, D. Patterson, and C. Young, "A new golden age in computer architecture: Empowering the machine-learning revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21–29, 2018.

[30] D. Liu, B. Chen, C. Yang, and A. F. Molisch, "Caching at the wireless edge: design aspects, challenges, and future directions," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 22–28, 2016.

[31] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[32] J.-J. Yu, M. Zhao, W.-T. Li, D. Liu, S. Yao, and W. Feng, "Joint offloading and resource allocation for time-sensitive multi-access edge computing network," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2020, pp. 1–6.

[33] Z. Wu*, Z. Liu*, J. Lin, Y. Lin, and S. Han, "Lite transformer with long-short range attention," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=ByeMPlHKPH

[34] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019.

[35] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," *arXiv preprint arXiv:2004.02984*, 2020.

[36] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[37] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," in *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*. IEEE, 2011, pp. 342–347.

[38] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[39] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[40] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," 2020.

[41] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[42] D. Gschwend, "Zynqnet: An fpga-accelerated embedded convolutional neural network," *Swiss Federal Institute of Technology Zurich: Zürich, Switzerland*, 2016.

[43] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[44] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[45] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1314–1324.

[46] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[47] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 285–300.

[48] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[49] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 116–131.

[50] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 6105–6114.

[51] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "Ghostnet: More features from cheap operations," *arXiv preprint arXiv:1911.11907*, 2019.

[52] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in neural information processing systems*, 2015, pp. 2962–2970.

[53] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2815–2823.

[54] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[55] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[56] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 129–137.

[57] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[58] J. L. Yuxi Li, Jianguo Li and W. Lin, "Tiny-DSOD: Lightweight object detection for resource-restricted usage," in *BMVC*, 2018.

[59] A. Womg, M. J. Shafiee, F. Li, and B. Chwyl, "Tiny ssd: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection," in *2018 15th Conference on Computer and Robot Vision (CRV)*. IEEE, 2018, pp. 95–101.

[60] R. J. Wang, X. Li, and C. X. Ling, "Pelee: A real-time object detection system on mobile devices," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 1963–1972. [Online]. Available: http://papers.nips.cc/paper/7466-pelee-a-real-time-object-detection-system-on-mobile-devices.pdf

[61] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun, "Light-head r-cnn: In defense of two-stage object detector," *arXiv preprint arXiv:1711.07264*, 2017.

[62] Z. Qin, Z. Li, Z. Zhang, Y. Bao, G. Yu, Y. Peng, and J. Sun, "Thundernet: Towards real-time generic object detection on mobile devices," in *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[63] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 781–10 790.

[64] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[65] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[66] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[67] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[68] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[69] Z. Shen, Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue, "Dsod: Learning deeply supervised object detectors from scratch," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1919–1927.

[70] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[71] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7310–7311.

[72] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.

[73] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605.

[74] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*. IEEE, 1993, pp. 293–299.

[75] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[76] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *International Conference on Learning Representations (ICLR)*, 2016.

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

[78] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2498–2507.

[79] X. Dai, H. Yin, and N. K. Jha, "Nest: A neural network synthesis tool based on a grow-and-prune paradigm," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1487–1497, 2019.

[80] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[81] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[82] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5687–5695.

[83] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[84] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[85] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[86] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001, vol. 16.

[87] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *International Conference on Learning Representations*, 2016.

[88] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[89] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.

[90] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[91] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[92] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, 2017, pp. 1–14.

[93] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 875–886.

[94] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 2234–2240.

[95] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 304–320.

[96] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[97] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 2130–2141.

[98] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang, "Learning filter pruning criteria for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2009–2018.

[99] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao, "Hrank: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1529–1538.

[100] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[101] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, pp. 1–18, 2017.

[102] J. Yu, A. Lukefahr, D. J. Palframan, G. S. Dasika, R. Das, and S. A. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 548–560, 2017.

[103] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *arXiv preprint arXiv:2003.03033*, 2020.

[104] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rJlnB3C5Ym

[105] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rJl-b3RcF7

[106] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu, "Pruning from scratch," *arXiv preprint arXiv:1909.12579*, 2019.

[107] E. Malach, G. Yehudai, S. Shalev-Shwartz, and O. Shamir, "Proving the lottery ticket hypothesis: Pruning is all you need," *arXiv preprint arXiv:2002.00585*, 2020.

[108] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[109] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International Conference on Machine Learning*, 2015, pp. 1737–1746.

[110] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[111] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[112] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[113] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[114] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[115] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[116] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Advances in Neural Information Processing Systems*, 2017, pp. 345–353.

[117] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[118] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[119] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, "Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 722–737.

[120] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382.

[121] S. R. Jain, A. Gural, M. Wu, and C. Dick, "Trained uniform quantization for accurate and efficient neural network inference on fixed-point hardware," *arXiv preprint arXiv:1903.08066*, 2019.

[122] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[123] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 293–302.

[124] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4350–4359.

[125] B. Zhuang, L. Liu, M. Tan, C. Shen, and I. Reid, "Training quantized neural networks with a full-precision auxiliary module," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1488–1497.

[126] L. H. K. D. Shien Zhu and W. Liu, "Xor-net: An efficient computation pipeline of binary neural network inference on edge devices," in *The 26th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2020.

[127] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in *Proceedings of the 2015 international workshop on internet of things towards applications*, 2015, pp. 7–12.

[128] H. Cai, C. Gan, L. Zhu, and S. Han, "Tiny transfer learning: Towards memory-efficient on-device learning," *arXiv preprint arXiv:2007.11622*, 2020.

[129] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[130] J. Burgess, "Rtx on—the nvidia turing gpu," *IEEE Micro*, vol. 40, no. 2, pp. 36–44, 2020.

[131] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 65–74.

[132] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[133] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[134] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[135] J. Lin, C. Gan, and S. Han, "Defensive quantization: When efficiency meets robustness," *arXiv preprint arXiv:1904.08444*, 2019.

[136] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan, "Mixed precision neural architecture search for energy efficient deep learning." in *ICCAD*, 2019, pp. 1–7.

[137] G. Chen, H. Meng, Y. Liang, and K. Huang, "Gpu-accelerated real-time stereo estimation with binary neural network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2896–2907, 2020.

[138] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "Apq: Joint search for network architecture, pruning and quantization policy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2078–2087.

[139] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv preprint arXiv:1412.6550*, 2014.

[140] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[141] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," *arXiv preprint arXiv:1612.03928*, 2016.

[142] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results," in *Advances in neural information processing systems*, 2017, pp. 1195–1204.

[143] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018.

[144] S. Ravi, "Efficient on-device models using neural projections," in *International Conference on Machine Learning*, 2019, pp. 5370–5379.

[145] T. Li, J. Li, Z. Liu, and C. Zhang, "Few sample knowledge distillation for efficient network compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 639–14 647.

[146] I. Chung, S. Park, J. Kim, and N. Kwak, "Feature-map-level online adversarial knowledge distillation," *arXiv preprint arXiv:2002.01775*, 2020.

[147] Y. Liu, W. Zhang, and J. Wang, "Adaptive multi-teacher multi-level knowledge distillation," *Neurocomputing*, vol. 415, pp. 106–113, 2020.

[148] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 535–541.

[149] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4133–4141.

[150] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[151] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 582–597.

[152] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://arxiv.org/pdf/1812.00332.pdf

[153] X. Luo, D. Liu, H. Kong, and W. Liu, "Edgenas: Discovering efficient neural architectures for edge systems," in *International Conference on Computer Design*, 2020.

[154] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-path nas: Designing hardware-efficient convnets in less than 4 hours," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2019, pp. 481–497.

[155] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 734–10 742.

[156] J. Fang, Y. Sun, Q. Zhang, Y. Li, W. Liu, and X. Wang, "Densely connected search space for more flexible neural architecture search," *arXiv preprint arXiv:1906.09607*, 2019.

[157] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "Nas-fpn: Learning scalable feature pyramid architecture for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 7036–7045.

[158] B. Chen, G. Ghiasi, H. Liu, T.-Y. Lin, D. Kalenichenko, H. Adams, and Q. V. Le, "Mnasfpn: Learning latency-aware pyramid architecture for object detection on mobile devices," *arXiv preprint arXiv:1912.01106*, 2019.

[159] N. Wang, Y. Gao, H. Chen, P. Wang, Z. Tian, C. Shen, and Y. Zhang, "Nas-fcos: Fast neural architecture search for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 943–11 951.

[160] H. Xu, L. Yao, W. Zhang, X. Liang, and Z. Li, "Auto-fpn: Automatic network architecture adaptation for object detection beyond classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 6649–6658.

[161] C. Gao, Y. Chen, S. Liu, Z. Tan, and S. Yan, "Adversarialnas: Adversarial neural architecture search for gans," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5680–5689.

[162] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers," in *Advances in Neural Information Processing Systems*, 2019, pp. 4978–4990.

[163] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HylxE1HKwS

[164] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," *arXiv preprint arXiv:2007.10319*, 2020.

[165] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia *et al.*, "Chamnet: Towards efficient network design through platform-aware model adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 398–11 407.

[166] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[167] X. Li, Y. Zhou, Z. Pan, and J. Feng, "Partial order pruning: For best speed/accuracy trade-off in neural architecture search," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[168] Y. Xiong, R. Mehta, and V. Singh, "Resource constrained neural network architecture search: Will a submodularity assumption help?" in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1901–1910.

[169] G. Gobieski, B. Lucia, and N. Beckmann, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 199–213.

[170] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, and L. Van Gool, "Ai benchmark: Running deep neural networks on android smartphones," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 0–0.

[171] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel, "Intermittent computing: Challenges and opportunities," in *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[172] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[173] A. Davis and I. Arel, "Low-rank approximations for conditional feedforward computation in deep neural networks," *arXiv preprint arXiv:1312.4461*, 2013.

[174] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[175] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Advances in neural information processing systems*, 2017, pp. 2181–2191.

[176] X. Gao, Y. Zhao, Ł. Dudziak, R. Mullins, and C.-z. Xu, "Dynamic channel pruning: Feature boosting and suppression," *arXiv preprint arXiv:1810.05331*, 2018.

[177] B. Ehteshami Bejnordi, T. Blankevoort, and M. Welling, "Batch-shaping for learning conditional channel gated networks," *arXiv*, pp. arXiv–1907, 2019.

[178] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, "Slimmable neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=H1gMCsAqY7

[179] J. Yu and T. S. Huang, "Universally slimmable networks and improved training techniques," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1803–1811.

[180] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[181] M. Ren, A. Pokrovsky, B. Yang, and R. Urtasun, "Sbnet: Sparse blocks network for fast inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8711–8720.

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

[183] A. Graves, "Adaptive computation time for recurrent neural networks," *arXiv preprint arXiv:1603.08983*, 2016.

[184] T. Verelst and T. Tuytelaars, "Dynamic convolutions: Exploiting spatial sparsity for faster inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2320–2329.

[185] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[186] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu, "Dynamic convolution: Attention over convolution kernels," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 030–11 039.

[187] W. Hua, Y. Zhou, C. M. De Sa, Z. Zhang, and G. E. Suh, "Channel gating neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 1886–1896.

[188] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, 2017, p. 527–536.

[189] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[190] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[191] S. E. Yuksel, J. N. Wilson, and P. D. Gader, "Twenty years of mixture of experts," *IEEE transactions on neural networks and learning systems*, vol. 23, no. 8, pp. 1177–1193, 2012.

[192] R. Teja Mullapudi, W. R. Mark, N. Shazeer, and K. Fatahalian, "Hydranets: Specialized dynamic architectures for efficient inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8080–8089.

[193] B. Yang, G. Bender, Q. V. Le, and J. Ngiam, "Condconv: Conditionally parameterized convolutions for efficient inference," in *Advances in Neural Information Processing Systems*, 2019, pp. 1307–1318.

[194] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," in *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. AAAI press, 2018, pp. 3675–3682.

[195] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *Advances in neural information processing systems*, 2016, pp. 550–558.

[196] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 409–424.

[197] Z. Wu, T. Nagarajan, A. Kumar, S. Rennie, L. S. Davis, K. Grauman, and R. Feris, "Blockdrop: Dynamic inference paths in residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8817–8826.

[198] A. Veit and S. Belongie, "Convolutional networks with adaptive inference graphs," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 3–18.

[199] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[200] E. J. Topol, "High-performance medicine: the convergence of human and artificial intelligence," *Nature medicine*, vol. 25, no. 1, pp. 44–56, 2019.

[201] I. P. Network, "Monetizing ai: How to get ready for 'inference at scale'." [Online]. Available: https://itpeernetwork.intel.com/ai-inference-at-scale/#gs.6ojv36

[202] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[203] "Nvidia jetson systems," 2020. [Online]. Available: https://developer.nvidia.com/embedded/develop/hardware#family

[204] W. Jiang, X. Zhang, E. H.-M. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[205] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[206] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[207] "Intel neural compute stick 2," 2020. [Online]. Available: https://software.intel.com/en-us/neural-compute-stick

[208] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. P. Stefanov, D. D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1517–1530, 2009.

[209] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo, and P. Dollár, "On network design spaces for visual recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1882–1890.

[210] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 428–10 436.

[211] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, p. 65–76, Apr. 2009. [Online]. Available: https://doi.org/10.1145/1498765.1498785

[212] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang *et al.*, "Mlperf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.

[213] Y. E. Wang, G.-Y. Wei, and D. Brooks, "A systematic methodology for analysis of deep learning hardware and software platforms," in *The 3rd Conference on Machine Learning and Systems (MLSys)*, 2020.

[214] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated end-to-end optimizing compiler for deep learning," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 578–594.

[215] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, p. 907–922.

[216] T. Hussain, K. Muhammad, J. Del Ser, S. W. Baik, and V. H. C. de Albuquerque, "Intelligent embedded vision for summarization of multiview videos in iiot," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 4, pp. 2592–2602, 2019.

[217] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[218] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 965–975.

[219] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[220] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, "Gan compression: Efficient architectures for interactive conditional gans," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 5284–5294.

[221] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[222] S. Dhar, J. Guo, J. Liu, S. Tripathi, U. Kurup, and M. Shah, "On-device machine learning: An algorithms and learning theory perspective," *arXiv preprint arXiv:1911.00623*, 2019.

[223] Y. Wang, Z. Jiang, X. Chen, P. Xu, Y. Zhao, Y. Lin, and Z. Wang, "E2-train: Training state-of-the-art cnns with over 80% energy savings," in *Advances in Neural Information Processing Systems*, 2019, pp. 5138–5150.

[224] Y. Wu, Z. Wang, Y. Shi, and J. Hu, "Enabling on-device cnn training by self-supervised instance filtering and error map pruning," *arXiv preprint arXiv:2007.03213*, 2020.

[225] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[226] T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang, "Error-driven incremental learning in deep convolutional neural network for large-scale image classification," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 177–186.

[227] J. Shin, S. Choi, Y. Choi, and L.-S. Kim, "A pragmatic approach to on-device incremental learning system with selective weight updates," in *Proceedings of the 57th Annual Design Automation Conference 2020*, 2020, pp. 1–6.

[228] Z. Chen and B. Liu, "Lifelong machine learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 12, no. 3, pp. 1–207, 2018.
[229] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.
[230] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.
[231] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.